Hasso Plattner Institute for Digital Engineering
at the University of Potsdam
Enterprise Platform and Integration Concepts Research Group
Prof. Dr. h.c. mult. Hasso Plattner

Situational Interventions and Peer Feedback in Massive Open Online Courses

Narrowing the Gap Between Learners and Instructors
in Online Programming Education

Dissertation
submitted in partial fulfillment
of the requirements for the academic degree of

Doctor of Natural Sciences
(Dr. rer. nat.)

in the scientific discipline of
Practical Computer Science

to the
Digital Engineering Faculty
at the University of Potsdam

by
Ralf Teusner, M.Sc.

Potsdam, January 8th, 2020

Supervisors

Prof. Dr. h.c. mult. Hasso Plattner
Hasso Plattner Institute for Digital Engineering

Prof. Dr. Falk Uebernickel
Hasso Plattner Institute for Digital Engineering

Prof. Dr. Helmut Krcmar
Technical University of Munich

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-50758
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-507587

Abstract

Massive Open Online Courses (MOOCs) open up new opportunities to learn a wide variety of skills online and are thus well suited for individual education, especially where proficient teachers are not available locally. At the same time, modern society is undergoing a digital transformation, requiring the training of large numbers of current and future employees. Abstract thinking, logical reasoning, and the need to formulate instructions for computers are becoming increasingly relevant. A holistic way to train these skills is to learn how to program. Programming, in addition to being a mental discipline, is also considered a craft, and practical training is required to achieve mastery. In order to effectively convey programming skills in MOOCs, practical exercises are incorporated into the course curriculum to offer students the necessary hands-on experience to reach an in-depth understanding of the programming concepts presented. Our preliminary analysis showed that while being an integral and rewarding part of courses, practical exercises bear the risk of overburdening students who are struggling with conceptual misunderstandings and unknown syntax. In this thesis, we develop, implement, and evaluate different interventions with the aim to improve the learning experience, sustainability, and success of online programming courses. Data from four programming MOOCs, with a total of over 60,000 participants, are employed to determine criteria for practical programming exercises best suited for a given audience.

Based on over five million executions and scoring runs from students' task submissions, we deduce exercise difficulties, students' patterns in approaching the exercises, and potential flaws in exercise descriptions as well as preparatory videos. The primary issue in online learning is that students face a social gap caused by their isolated physical situation. Each individual student usually learns alone in front of a computer and suffers from the absence of a pre-determined time structure as provided in traditional school classes. Furthermore, online learning usually presses students into a one-size-fits-all curriculum, which presents the same content to all students, regardless of their individual needs and learning styles. Any means of a personalization of content or individual feedback regarding problems they encounter are mostly ruled out by the discrepancy between the number of learners and the number of instructors. This results in a high demand for self-motivation and determination of MOOC participants. Social distance exists between individual students as well as between students and course instructors. It decreases engagement and poses a threat to learning success. Within this research, we approach the identified issues within MOOCs and suggest scalable technical solutions, improving social interaction and balancing content difficulty.

Our contributions include situational interventions, approaches for personalizing educational content as well as concepts for fostering collaborative problem-solving. With these approaches, we reduce counterproductive struggles and create a universal improvement for future programming MOOCs. We evaluate our approaches and methods in detail to improve programming courses for students as well as instructors and to advance the state of knowledge in online education.

Data gathered from our experiments show that receiving peer feedback on one's programming problems improves overall course scores by up to 17%. Merely the act of phrasing a question about one's problem improved overall scores by about 14%. The rate of students reaching out for help was significantly improved by situational just-in-time interventions. Request for Comment interventions increased the share of students asking for help by up to 158%. Data from our four MOOCs further provide detailed insight into the learning behavior of students. We outline additional significant findings with regard to student behavior and demographic factors. Our approaches, the technical infrastructure, the numerous educational resources developed, and the data collected provide a solid foundation for future research.

Zusammenfassung

MOOCs (Massive Open Online Courses) ermöglichen es jedem Interessierten sich in verschiedenen Fachrichtungen online weiterzubilden. Sie fördern die persönliche individuelle Entwicklung und ermöglichen lebenslanges Lernen auch dort, wo geeignete Lehrer nicht verfügbar sind. Unsere Gesellschaft befindet sich derzeit in der sogenannten „digitalen Transformation“. Von vielen Arbeitnehmern werden in diesem Zusammenhang zunehmend Fähigkeiten wie abstraktes Denken und logisches Schlussfolgern erwartet. Das Erlernen einer Programmiersprache ist eine geeignete Möglichkeit, diese Fähigkeiten zu erlangen. Obwohl Programmieren als geistige Disziplin angesehen wird, ist es zu einem gewissen Grad auch ein Handwerk, bei dem sich das individuelle Können insbesondere durch stetige praktische Anwendung entwickelt. Um Programmierkenntnisse effektiv in einem MOOC zu vermitteln, sollten daher praktische Aufgaben von vornherein in den Lehrstoff des Kurses integriert werden, um die vorgestellten Konzepte geeignet zu vertiefen und zu festigen. Neben den positiven Aspekten für die Lernenden weisen praktische Programmieraufgaben jedoch auch ein erhöhtes Frustpotential auf. Kryptische Fehlermeldungen und teils unbekannte Syntax überfordern insbesondere diejenigen Teilnehmer, welche zusätzlich mit konzeptionellen Missverständnissen zu kämpfen haben.

Im Rahmen dieser Arbeit entwickeln und analysieren wir mehrere Interventionsmöglichkeiten um die Lernerfahrung und den Lernerfolg von Teilnehmern in Programmier-MOOCs zu verbessern. Daten von über 60.000 Teilnehmern aus vier Programmier-MOOCs bilden die Grundlage für eine Analyse von Kriterien für geeignete Programmieraufgaben für spezifische Teilnehmergruppen. Auf Basis von 5 Millionen Codeausführungen von Teilnehmern leiten wir Schwachstellen in Aufgaben und typische Herangehensweisen der Teilnehmer ab. Die Hauptschwierigkeit beim Lernen in einer virtuellen Umgebung ist die durch physische Isolation hervorgerufene soziale Entkopplung. Jeder Teilnehmer lernt alleine vor einem Bildschirm, ein gemeinsamer Stundenplan wie im klassischen Schulunterricht fehlt. Weiterhin präsentieren bestehende online Kurse den Teilnehmern in der Regel lediglich universell einsetzbare Lerninhalte, welche in keiner Weise auf die jeweiligen Bedürfnisse und Vorerfahrungen der individuellen Teilnehmer angepasst sind. Personalisierte Lerninhalte bzw. individuelles Feedback sind in MOOCs aufgrund der großen Anzahl an Teilnehmern und der nur kleinen Anzahl an Lehrenden oft nur schwer bzw. gar nicht zu realisieren. Daraus resultieren wiederum hohe Anforderungen an das individuelle Durchhaltevermögen und die Selbstmotivation der MOOC-Teilnehmer. Die soziale Entkopplung manifestiert sich sowohl zwischen den Teilnehmern untereinander als auch zwischen den Lehrenden und den Teilnehmern. Negative Folgen sind ein häufig verringertes Engagement und damit eine Gefährdung des Lernerfolgs. In dieser Arbeit schlagen wir als Gegenmaßnahme skalierbare technische Lösungen vor, um die soziale Interaktion zu verbessern und inhaltliche Schwierigkeiten zu überwinden.

Unsere wissenschaftlichen Beiträge umfassen situationsabhängige Interventionen, Ansätze zur Personalisierung von Lerninhalten, sowie Konzepte und Anreize zur Verbesserung der Kollaboration der Teilnehmer untereinander. Mit diesen Maßnahmen schaffen wir es, kontraproduktive Blockaden beim Lernen zu lösen und stellen damit einen universell einsetzbaren Ansatz zur Verbesserung von zukünftigen Progammier-MOOCs bereit.

Die aus unseren Experimenten gesammelten Daten zeigen, dass bei Programmierproblemen gewährtes Feedback von anderen Teilnehmern die Gesamtpunktzahl innerhalb des Teilnehmerfeldes durchschnittlich um bis zu 17% verbessert. Bereits das Formulieren des jeweiligen individuellen Problems verbesserte die Gesamtpunktzahl um etwa 14%. Durch situative Interventionen konnte weiterhin der Anteil der Teilnehmer, die nach Hilfe fragen, um bis zu 158% gesteigert werden. Die gesammelten Daten aus unseren vier MOOCs ermöglichen darüber hinaus detaillierte Einblicke in das Lernverhalten der Teilnehmer. Wir zeigen zusätzlich Erkenntnisse in Bezug auf das Verhalten der Teilnehmer und zu demografischen Faktoren auf. Die in dieser Arbeit beschriebenen Ansätze, die geschaffene technische Infrastruktur, das entworfene Lehrmaterial, sowie der umfangreiche gesammelte Datenbestand bilden darüber hinaus eine vielversprechende Grundlage für weitere zukünftige Forschung.

Acknowledgement

A PhD thesis is usually seen as an individual endeavor. Nevertheless, it is also a team effort to some extent. Given the length of this activity over several years, this endeavor is only possible with continuous support along the way. I am truly grateful for all of you who have encouraged, challenged, and inspired me.

First, I sincerely thank my supervisor Hasso Plattner for supporting me to choose a field of research I truly believe in. Many thanks also to Matthias Uflacker, who guided me along the way and provided valuable feedback at all stages. My colleagues at the EPIC research group deserve a great thank you for providing an intellectually challenging and pleasant atmosphere.

Thank you also to the openHPI team for showing a can-do attitude whenever facing unexpected challenges. My master students Philipp Giese, Nicholas Wittstruck, Kai-Adrian Rollmann, Thomas Hille, Sebastian Serth, and Lukas Boehme deserve my sincere thanks for positively challenging me along the way and being good company in exploring the different areas of online education.

Probably the most intense and also most rewarding phase of this research was developing and conducting the numerous MOOCs forming the basis for my experiments. I am truly grateful for being able to spend most of these challenging times together with Ann Katrin Kuessner, Thomas Staubitz, and Christiane Hagedorn.

Finally, my foundation to confidently reach out for yet unknown endeavors is the steady support of my friends and family. In particular, I am most grateful to my parents Michael and Ute, and my sisters Lena and Sandra for supporting my decisions and fueling my “inveterate optimism”, leaving me to look back with a smile - already thinking of the next challenges to face.

Potsdam, January 8th, 2020

Ralf Teusner

Contents

1 Introduction

Programming skills, as part of digital literacy, have the potential to fuel a new age of enlightenment. Looking back, the ancient Greeks established the first formalized educational concepts[1] and opened up education for broader, but limited societal circles: free, wealthy males. This led to cultural breakthroughs in numerous disciplines, including politics, physics, and philosophy. In the Middle Ages, access to education expanded considerably, enabled by letterpress printing and school systems. The slow but constant spread of literacy as well as scientific interest among the broader public fueled and was an integral part of the Age of Enlightenment [132], building the foundation of our current prosperity.

Nowadays, society is once more facing a dramatic change: industrialization and its common jobs are fading away, whereas careers in the digital age often require advanced problem solving skills and abstract thinking.

To allow for an inclusive society and therefore participation, appropriate training becomes a fundamental necessity. A major challenge society is facing when applying traditional education approaches is that parts of the society are excluded if they are no longer attending school or university.

1.1 MOOCs and Their Potential to Educate the Masses

Massive Open Online Courses (MOOCs) are educational online courses that allow for open-access of an unlimited number of students. Compared to the high tuition fees of Ivy League universities, these courses, which are usually free of charge, open up education to a large range of the population, independent of one's financial background or age. Furthermore given the low barriers to entry with regards to prior knowledge and the technical ability to scale to thousands of participants, MOOCs are a suitable approach to "educate the masses".

Nevertheless, MOOCs also face several substantial challenges, such as a perception of anonymity and the increased gap between instructors and students. Most prominently, this gap is induced by the virtual environment and the tremendous disparity in numbers between a small teaching team (usually less than ten persons) and enrollment numbers which can reach tens of thousands of participants. Additionally, MOOCs face relatively low completion rates compared to class-

[1] The so-called "paideia".

room training, low permeability for specialized topics and unsolved questions concerning knowledge retention and employer recognition.

The phenomenon of MOOCs, starting in fall 2011 with the two massively popular MOOCs “Intro to Artificial Intelligence”[2] by Sebastian Thrun and Peter Norvig, and “Machine Learning”[3] by Andrew Ng, has caused great expectations, financially as well as transformatively. The so coined “MOOC revolution” [25, 45, 73] has been proclaimed by investors, journalists as well as researchers. Six to seven years later, it is safe to say that MOOCs, like many other former trending technologies, followed Gartner’s hype cycle [95] and, depending on whom one asks, are currently going through the “slope of enlightenment” [32] or already reached the “plateau of productivity” [134].

The exaggerated expectations towards MOOCs stylized it as a savior to education in general, eventually solving part of humanity’s greatest challenges and bringing wealth and stability through free education also to developing societies. Others feared that MOOCs might impair or even kill universities in the long term, especially smaller, non-Ivy-League ones [179]. The general opinion was mostly, that universities will survive. Furthermore most publications and essays agreed, that traditional education will be disrupted and forced to reinvent itself [70, 104, 125], or as John Hennessy put it, “There’s a tsunami coming” [7]. Neither the glorious nor the dystopian visions have come true so far. In order to assess the general impact MOOCs have contributed to education until now, we first outline their most common usage scenarios. Subsequently, we briefly showcase the advantages of MOOCs, before we outline their drawbacks.

MOOCs complement and extend traditional classroom education. They allow for easier adoption of so-called “blended learning” and “flipped classroom” settings [16, 82]. Furthermore, they bring in an additional variety of viewpoints, presentation styles, as well as didactical approaches.

Employed funding models depend on the organizations offering the courses. Non-profit MOOCs, often created by universities or other public institutions, are usually based on pre-existent learning material and further funded via additional research budgets. Commercial for-profit MOOC providers have found viable business models for their offerings in the meantime. They mostly enhanced their paid offerings from just advertising supplemental services (e.g. prioritized support or additional training) to offering a coherent series of multiple courses within a given knowledge area. These courses complement each other and their successful completion, together with a payment for the assessment, then leads to a certification called a nanodegree. With these nanodegrees, commercial MOOC providers attempt to improve the recognition of the acquired knowledge among companies within their hiring processes.

Considering the effects towards education in general, the impact is regarded as positive overall, while pre-existing drawbacks unfortunately remain. On the positive side, MOOCs offer access to high-quality education regardless of individual wealth and location (as long as a stable internet connection and a capable play-

[2] see `https://eu.udacity.com/course/intro-to-artificial-intelligence--cs271`. This link and all subsequent links, together with their last access dates, can also be found in the appendix.

[3] see `https://www.coursera.org/learn/machine-learning`

back device are available). They free education from a mostly limited timeframe in one's life and from the location of the campus.

However, MOOCs also have to face the very same problems traditional education is facing: low social permeability. On top of that, MOOCs also suffer from additional threats: a high dropout rate (or otherwise put, a low completion rate), mostly unknown retention of knowledge [161], and a low or unknown recognition of students' achievements.

Completion rates are generally low, and participants mostly do not come back for additional classes after they finish (or quit) their first MOOC [127]. Additionally, the audience reached by MOOCs is mostly pre-educated and wealthy. People living in countries that show a lower or very low Human Development Index (HDI), could not be reached at scale [127]. This sounds plausible, as it might either be that the prerequisites (internet and capable device) cannot be met, or priorities are simply not focused on education, as long as one's basic needs are not met.

It is important to note, that MOOCs have outgrown traditional e-learning, as they are (and have to be) more than just "learning material made available online". Traditional e-learning, focusing on content distribution, gathering of submissions an organization of small to medium-sized classes, should not be mistaken for online learning in massive context. Otherwise, this common misconception omits or even negates the very nature that allows MOOCs to become a place of individual growth, the online interaction between the students and the resulting academic discourse.

1.2 Pathways to Digital Literacy

The term digital literacy is not yet defined clearly. The common understanding is that the term means possessing knowledge to use digital media, in particular the internet. What skillset, however, to be regarded as necessary to properly "use" the internet, is open for debate. Some argue that knowledge of advanced search techniques is part of this skillset, others opt for the ability to evaluate online content. To further add to that, it could also include the ability to alter the medium, allowing one to create new content and to extend existing content. The most extensive definition and reasoning available as of now has been compiled by David Buckingham, building on his definitions on the term "media literacy" [18]. He argues that digital literacy builds upon the four areas of media literacy being applied to a digital medium, for example, the internet or a computer game. For the internet, the first area, representation, covers the ability to interpret the content and address its source as well as reliability. The second area, language, comprises understanding words and grammar of the communication medium - in terms of the internet, this includes concepts like hyperlinks and the structure of web sites. The third area, production, means recognizing concepts of advertising and its effects on the credibility of the source. Finally, the fourth area, audience, focuses on the approach the medium takes towards the consumer.

Given the necessity to understand the mechanics driving our modern world based on data and algorithms, fundamental building blocks like understanding the concepts of variables and basic control structures are therefore a vital part of achieving digital literacy. In order to understand and master these concepts thoroughly, it is best to practically experiment with them to see their effects in different situations. And doing this means learning programming.

According to the literature, the most effective way of acquiring programming knowledge is having a skilled and experienced tutor sitting right next to the student [37, 49, 130]. Requiring novices to describe their individual solutions and asking them to explain details about program flow further helps to ensure a thorough understanding of the concepts being taught.

Our research provides novel insights into the area of online programming education and offers approaches as well as evidence on how to tackle the aforementioned issues.

The main research topic of this thesis is to narrow the technical and social gap regarding communication and learning support between students and instructors within programming MOOCs. In order to approach this goal, we investigated three broader research dimensions. We will later relate our more fine-grained contributions to these dimensions, depending on their most prominent characteristics.

1. **Effects of Collaboration:** Making MOOCs more personal by using technical measures, including video telephony and asynchronous question answering, is likely to increase individual positive involvement, sense of belonging and knowledge retention. Thus, we examine the requirements of successful collaboration and measure the effects on performance metrics such as completion rates and course scores.

2. **Personalization of Content:** Individualizing course content towards specific students' needs potentially improves knowledge adoption and students' satisfaction. Likewise, we measure the effects on students motivation and exercise scores.

3. **Scalability of Approaches:** Scaling individual feedback and learning support within MOOCs poses several challenges. Overcoming those is possible either by technical automation or by crowdsourcing. We describe the technical implementations of our approaches and further outline steps that helped the audience adopt and embrace our proposed crowdsourcing measures.

1.2.1 Improving the Learning Process

After the initial development, our efforts to strengthen student collaboration and content personalization were applied and continuously improved within several large scale programming MOOCs. The iterative process shown in Figure 1.1 allowed us to gain a thorough scientific understanding of the induced effects on the one hand and on the other hand ensured a lasting, practical impact for the students, specifically fitting the given setup.

The central starting point for improving learning outcomes is a good understanding of the students' situation within the course. All changes being applied to the "normal" course flow are intended to improve the learning experience and outcomes of students. However, interventions also bear the risk of impeding students' experience, thus requiring us to act only on an informed basis gained from suitable metrics which, for example, indicate that a student is struggling with an exercise. Depending on the cause of the undesired struggle, it may be either appropriate to affect individual students (intervene), or adjust the material that is offered in general (adapt). Thus only based on a solid understanding of what is happening within a MOOC in general and what might hinder an individual student in particular, it is possible to purposefully intervene or adapt the difficulty of the content.

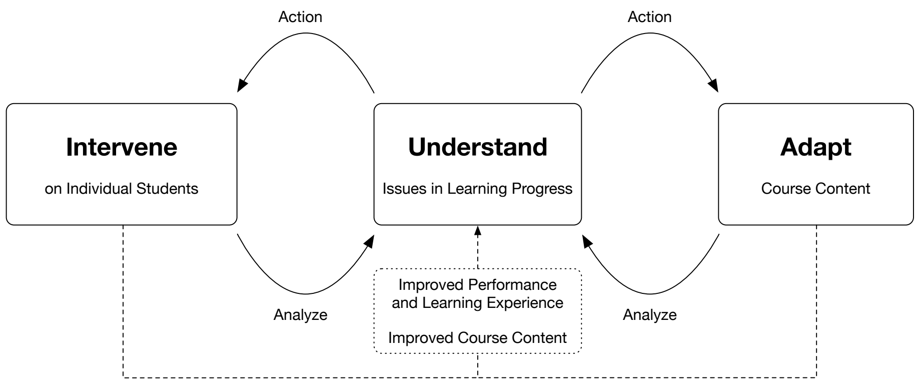


Figure 1.1: Overall concept of improving the learning process.

In addition to providing an understanding of students' experiences, the metrics we have gathered also allow us to assess the effects of applied measures afterwards. Interventions that have been conducted or changes to the course material thus subsequently add to the instructors' understanding of issues, as they can track and analyze effects of the changes. This general, iterative model reflects the overall structure we followed to enact our implemented approaches within the research context of programming education in MOOCs.

1.2.2 Research Context

My contributions include general learnings backed by several experiments conducted in a series of MOOCs (13 MOOCs in total).

Concerning methodology, all challenges have been tackled from different viewpoints and with different approaches. The analyzed MOOCs were run in the timeframe from autumn 2012 to spring 2018 and provide qualitative insights as well as quantitative results. While the first MOOCs on "In-Memory Data Management" did not address the primary goal of programming education, they provided initial enrollment, activity and completion rates and further enabled us to gather initial technical as well as didactical experience with this new medium. The first programming-centric MOOCs, teaching Python to school children, provided insights into students' coding behavior as well as the acceptance of automated grading on the basis of unit tests. Although these MOOCs were aimed at school kids, interest was sparked among all age groups. Subsequent programming courses, teaching Java over a timespan of between four to six weeks, therefore were designed for beginners of all ages.

Research-wise, we enhanced the survey and quiz-based data collection with metrics gathered on the code execution platform CodeOcean (further described in Section 4.1. Within several shorter, two to four-week-long workshop MOOCs, we measured the impact of different content difficulty (i.e. completion rates for exercises on beginners or advanced level), didactical approaches (i.e. guided learning vs. project- and problem-centered learning) and content amount. In addition, several experiments were run to further understand students' motivations and validate acceptance of our approaches. Ad-hoc clustering of student groups allowed us to further analyze students' learning behavior and the impact of individualized motivational mailings. Within CodeOcean, we tested user acceptance of automatic answer suggestions for programming issues and video conferencing.

The main experiments for the central element of this thesis were designed building on the gathered insights and finally tested within two independent Java MOOCs run in 2017 and 2018. In the first course run in 2017, the so-called "Request for Comments" feature and the "just-in-time interventions" were initially tested for acceptance and impact. To revalidate the initial findings and answer follow up questions, the experiments were extended and rerun in another course in 2018. To widen the scope of students who were covered in this study and to ensure that our experiments were conducted with a previously unaffected audience, the course was conducted in English.

Our overall aim is to improve the effectiveness and motivation of students through collaboration, by adding to distance education what it lacks most: personal interaction.

1.3 Contributions

This work contributes the following advances to the current state of research:

- **Statistical evaluation of numerous hypotheses on the effects of employed collaborative measures (Effects of Collaboration)**
  Scientific evaluation of the implemented measures within several programming MOOCs resulted in a comprehensive collection of students' performance data. We posed several hypotheses prior to each course iteration and employed randomized A/B-testing with control groups in order to validate our assumptions. The published findings showing statistical significance advance the status quo in MOOC research. Additionally, several null findings are specifically outlined to further enlarge the general knowledge base and provide data for fellow researchers to compare against.

- **A scalable mechanism to provide individualized feedback and help with programming problems (Effects of Collaboration)**
  Enabling students to reach out for help and expressing their question towards their fellow students is a crowdsourcing approach to provide individual feedback to arbitrary programming problems. Our mechanism called "Request for Comments" enriches students' questions with the associated source code, the program output, and errors that occurred. It furthermore forwards the question to fellow students who are presumably capable of providing competent help.

- **First work to implement video-tutoring in programming education on a massive scale (Effects of Collaboration)**
  We developed a system that allows students to connect to tutors via video telephony directly from within the coding environment. It addresses the specific needs of programming MOOCs, e.g., by additionally live-sharing the student's progress and output with the tutor. Furthermore, our solution runs on our servers in order to comply with GDPR requirements and does not require an additional account in contrast to most third-party tools.

- **A mechanism to provide students with content tailored to their needs (Personalization of Content)**
  Similar to a tutor suggesting additional training exercises to mend specific weaknesses, we contribute an approach to automatically determine student-specific weaknesses concerning conveyed concepts. Based on these weaknesses, we suggest additional learning content, in our case non-graded bonus exercises.

- **Novel means to intervene on struggling students during their work (Scalability of Approaches)**
  Intervening on struggling students while they are engaged in their studies promises to be the best situation to avoid additional context switches and achieve the best effect to counter frustration. With situational popups, encouraging students to either request help or take a break, we increase students' adoption of our interventions and thus improve learning results.

- **Metrics to classify and group programming students (Scalability of Approaches)**
  Finding appropriate criteria to classify students into subgroups is necessary to accurately apply and improve potential interventions. The determination of the prior knowledge of students, for example, bears intricacy if required for large audiences. Gathering of the necessary information has to be conducted without obtrusion to ensure high audience coverage and prevent students' resentment.

- **First work to introduce (semi-)automated detection of weak learning material (in MOOCs) (Scalability of Approaches)**
  Anomalies in students' performance metrics (e.g. a much higher mean working time in an exercise) indicate potentially flawed content. We propose a system to automatically detect material provoking such outliers. Our approach provides course instructors with community feedback on these exercises and allows for timely correction to limit the negative impact on students' motivation.

- **Improvement of an open-source programming platform, scalable for MOOC requirements and capable of supporting large scale scientific experiments (Scalability of Approaches)**
  CodeOcean, initially conceptualized and implemented within a master's thesis at the HPI, has been advanced from prototype status to a field-tested and dependable tool applicable to MOOC audiences larger than 20,000 enrolled students. Furthermore, numerous technical improvements have been integrated into the system, e.g., to support synchronous user input/output, provide detailed statistics for instructors, and allow for convenient management of large numbers of exercises.

1.4 Published Results

Substantial parts of the following conference papers, originally published with ACM or IEEE, have been reused in this thesis. These parts contribute to the respective motivation, concept, implementation, evaluation, related work, and conclusion chapters of this thesis. Reused text parts are listed in the preambles of the respective chapters. The corresponding publications containing reused sections are first-authored by the author of this thesis. All substantial components of the work, including conceptualization and experiment design, have been directly carried out or led by the author of this thesis. Reprints were made with permission from the publishers (see Appendix 9.1).

The paper *Effects of Automated Interventions in Programming Assignments: Evidence from a Field Experiment* [160], published at ACM Learning@Scale in 2018, first introduced and described our concepts Request for Comments, just-in-time interventions, and tailored bonus exercises. Further, the results of the experiments run in the German Java MOOC "Java für Einsteiger (2017)" are presented in this paper. Our paper *What Stays in Mind? - Retention Rates in Programming MOOCs* [161], published at IEEE Frontiers in Education in 2018, contributes learnings gathered concerning the long-term effects of MOOCs. *On the Impact of Programming Exercise Descriptions - Effects of Programming Exercise Descriptions to Scores and Working Times On the Impact of Programming Exercise Descriptions* [158], published at IEEE LWMOOCs 2018, provides insights with regard to badly phrased task descriptions. The concept as well as corresponding results on the topic of video conferencing were first published as *Video Conferencing as a Peephole to MOOC Participants* [163] on IEEE TALE in 2017. Partial results of *Aspects on Finding the Optimal Practical Programming Exercise for MOOCs* [159] contribute to the concepts we use to measure students' learning success in order to evaluate our interventions. This work was published at IEEE FIE in 2017. The concepts behind our employed programming platform were first described in *CodeOcean - A Versatile Platform for Practical Programming Exercises in Online Environments* [146], published at EDUCON in 2016.

1.5 Outline

In the remainder of this document, we will follow the structure outlined in this paragraph. Chapter 2 sets the stage for the following approaches, by introducing the general background, status quo, and challenges of programming education within MOOCs. Chapter 3 classifies and showcases the different approaches we have implemented and evaluated in order to tackle the aforementioned challenges. Specific details about the technical realizations of the tools we developed are described in Chapter 4. It showcases the foundation for our experiments, the execution platform "CodeOcean", as well as the specific tools used in conjunction to conduct experiments and support the programming courses. Within Chapter 5, we detail the setup and outcomes of our experiments, including the methodologies we used, the observed results, and a reasoned discussion. We further add several findings of additional side experiments that confirm and expand current knowledge, but are not described in full detail in this thesis for the sake of conciseness. Chapter 6 provides an outlook of promising next steps. In Chapter 7 we present related work, before we conclude the conducted research with a summary of the achieved outcomes in Chapter 8.

2

Background

Having outlined the main issue of a social gap in programming MOOCs, this chapter will give an overview of the background and foundational conditions of our work. We first share the most common, established theories employed in the domain of education, before we elaborate on the differences between in-class and distance education. We outline factors that impact the learning outcomes and describe suitable metrics to trace potential effects of our experiments. Subsequently, in the context of measured metrics and gathered data, a short overview over ethical questions with regard to our experiments is given. Finally, this chapter highlights the specifics of programming education in online environments and presents a classification of relevant programming platforms, including the one that was used as the foundation for our experiments. Parts of Subchapters 2.1.2 and 2.2.1 have been published in [160], respectively additional parts of Subchapters 2.2.6 and 2.2.7 have been published in [159].

2.1 Theoretical Background

Revisiting the three research dimensions to be tackled in this research, it is evident that many of our questions to be addressed have already been worked on in the past and thus many proven theories already exist. This opens up an area to put the novel questions we address into context and relate our findings to prior results.

Considering our first research dimension, *effects of collaboration* on learning, different factors come into play: the foundational content at hand and its desired learning effect, the experienced difficulty level, as well as the design of the collaboration session itself. In order to classify the content, conveyed knowledge, and different process directions for learning effects, we will present Krathwohl's taxonomy. The concept of self-regulated learning divides the learning process into phases to additionally distinguish and outline students' meta-activities which are considered to improve the learning results. Vygotsky's idea of a zone of proximal development further contributes theory to justify and estimate optimal difficulty levels for learning.

An adjustment of the level of difficulty encountered by a student is possible through tutoring, a collaboration activity for which the resulting effects have been extensively investigated in the past.

The second dimension, *personalization of content*, is also affected and can be explained with Krathwohl's taxonomy, as well as the theory of self-regulated learning. An overview of prior approaches for personalized learning and a classification of recommender systems round up the background for our second dimension of research.

The third dimension, *scalability of approaches*, wraps around the other two dimensions. It is not an autonomous research direction itself, but adds unique requirements to potential approaches. The application of collaboration and personalization in the MOOC context introduces two additional circumstances, largely reducing the existing applicable approaches. The delivery of content online automatically implies a distance education setting, while the aspect of massiveness prohibits approaches requiring individual instructor interference. To account for this from the theoretical side, we outline the differences between in-class and distance education, with an additional focus on the mismatch between the numbers of students and instructors.

Summarizing, the theories and prior findings presented in the following subsections individually affect our three research dimensions in different aspects. Altogether, the theories will build the foundation for a well-grounded discussion of our experiments and the resulting findings in Chapter 5.

2.1.1 Krathwohl's Taxonomy

In order to foster lasting learning effects, we offer students the opportunity to try out the presented content and practically use the conveyed concepts. On the basis of the six hierarchical levels of reasoning skills in Bloom's taxonomy [11] (i.e. knowledge, comprehension, application, analysis, synthesis, and evaluation), educators developed learning material in order to advance their students. Although not presented this way by Bloom, the taxonomy is often depicted in a pyramidal shape, as the levels build on each other, and reflect progress in understanding and skills. On the basis of this model, learning material can be categorized according to the affected and trained levels of skills.

While helpful in general, the model also suffers from several shortcomings [143]. Criticized is the absence of representation of the type of skill or material that is being taught, leaving the model unable to distinguish between teaching of, e.g., facts and concepts. Another mentioned issue is the hierarchical nature of the model, resulting in disdain for the lower levels because of treating the knowledge level as inferior instead of being fundamental [94].

Krathwohl [84] revised the taxonomy and improved the model with respect to the mentioned weak points. Most visibly, Krathwohl added a second dimension to the original taxonomy. The existing "cognitive process" dimension is now accompanied by a "knowledge" dimension, distinguishing between the main categories of factual, conceptual, procedural, and metacognitive knowledge. Furthermore, Krathwohl also changed the substantive forms of the cognitive process dimension to verb forms, replacing knowledge (now being a distinct dimension) with "remember" and placing "create" after "evaluate". An overview of the scheme used to structure and classify educational content is depicted in Figure 2.1.

	Remember	Understand	Apply	Analyze	Evaluate	Create
Factual	Recall historic dates in exam					
Conceptual		Relation between speed and distance		Find stylistic devices in a poem	Interpret a poem	Finish a story
Procedural			Perform differential calculus on equation		Interpret a poem	Finish a story
Metacognitive						

Figure 2.1: Examples of learning items classified into Krathwohl's taxonomy.

When classifying items, a decision has to be made for each of the two dimensions. The options concerning the cognitive process are shown in the horizontal direction. Remember subsumes activities aiming to have students recall content and knowledge that was prior presented. Such activities are often applied in tests asking for facts, e.g., having students recall historic dates, such as the prototypical history question: "When was the Battle of Issus?"[4] (factual). Understand covers activities requiring to determine the meaning of facts or instructions, qualifying students to summarize, compare, and interpret the presented content later on (e.g. relationship between distance and movement speed in physics (conceptual)). Applying means to use a fact or procedure in an appropriate situation, e.g., perform differential calculus in math for a given equation (procedural). The process of breaking down presented information to its sub-parts is analyzing, e.g., finding and naming stylistic devices in a poem (conceptual). Making sense of such information and judge potential effects is regarded as "evaluate", as for example when interpreting the analyzed poem (conceptual and procedural). Create, the last cognitive process option, represents activities to create novel content on the basis of the conveyed knowledge, e.g., finishing an incomplete story (conceptual and procedural) [84].

The knowledge dimension further details the information conveyed. Factual knowledge is all kind of information to be learned by heart, such as vocabulary, terminology, formulas, parts of a system, or dates. Conceptual knowledge sets these facts into relation and therefore exists in the form of models, categorizations, or principles. Anything that empowers a student to perform certain workflows, such as algorithms, skills, and techniques, is subsumed as procedural knowledge. The last category, metacognitive, includes strategic knowledge as well as awareness of one's own cognition and skills [84].

Given those categories, almost all activities, content, or tasks of common education can be classified and placed in the matrix. In case of ambiguities, the respective item is either placed in multiple cells or just in the predominant cell. Placing all activities of a course into this taxonomy allows to gain a thorough overview of the focus areas as well as shortcomings of the presented content.

[4] The Battle of Issus took place 333 BC between the Hellenic League and the Achaemenid Empire.

While some shortcomings and critic remain, e.g., a lack of constructivist integration or the distinction between categories [143], the model receives wide acceptance and usage. For the purpose of guiding the creation of an xMOOC and estimating its didactical coverage, the revised taxonomy is well fitted. Not being over-complicated and having relevance in the field, therefore qualifies the model to explain aspects of our content, mainly the course videos, self-test quizzes, and the programming exercises. The lack of constructivist integration does not affect us, as we do not require a holistic model explaining all aspects at once. Also the separation between categories is not a major issue, if one is aware that the borders are blurry, items will be placed based on their primary learning goal, and the levels are non-exclusive. We will employ the taxonomy to explain the rationale behind our educational efforts when describing our course setup in Section 5.2.

2.1.2 Influencing Factors of Online Education

The general aim of this research is to improve online programming education. In order to achieve an optimal impact, it is helpful to outline which factors influence the quality and success of online education, which of those can be influenced by teaching teams and researchers, and which factors reside out of scope. Two of the most vital influences towards learning effort are the student's motivation and self-efficacy [10]. Improving those individual factors is not possible on a universal level, rather difficult on individual course or item level, and hard to measure in general, as it requires students to succeed in tasks they expect to fail [99]. For these reasons, we abstained from trying to target these factors directly, despite giving our best to keep these values high by offering appealing, high-quality content. As reasoned in Chapter 1, one of the central elements in programming education are the offered practical exercises. Although watching course videos requires students' attention, this is a passive activity apart of the mere consumption. In contrast, practical exercises involve active reasoning, application of knowledge, abstraction, and transfer of learning. Research shows that engaging in active activities reduces dropouts and has a higher impact on learning and learning results than passive activities such as reading [83]. Accordingly, solving the exercises is a critical activity in learning, both offering situations for success as well as mistakes, struggle, and frustration. From qualitative student feedback in course forums and quantitive data analysis [159], we know that extended struggle often leads to frustration and ultimately to dropping out of the course. High dropout numbers have often been argued to be one of the main issues of MOOCs [81, 114]. In general, all MOOCs show the same phenomenon that roughly between 70 and 90 percent of all enrolled students do not finish the courses [114]. Students take courses because of different reasons, such as general interest, job relevance, or the wish of earning a certificate [79]. It is difficult to find the exact reason why a student dropped out from a course, e.g., time constraints or lost interest, since the response rate of such surveys is generally low (between 12.5% [77] and 1% [174]). A general lack of time is the most common factor, as Kizilcec and Halawa stated in [77], with a share of 84% of participants mentioning that reason. Other reasons could be that the students had problems in the practical exercises, or felt bored [114]. In our research we particularly aim at students who are interested in finishing the course but felt overwhelmed with the exercises.

Dropout Prediction and Prevention

Dropout prediction has been a vivid field in MOOC research: data is being published as a foundation for further research[5], and the actual prediction is approached with different methods including sentiment analysis on course forums [172] and using statistical reasoning or machine learning [81] on students' clickstream data. Accuracy of the employed models relies on the used training paradigm. Training a model on the same course (post-hoc) achieves a high accuracy of about 90%, but is unusable for intervention purposes. Whitehill et. al showed that similar accuracies of around 86% could be achieved with training on proxy labels or data from many other courses [173]. Reasons for dropout are manifold: students might never have had the intention to complete a course, as they were solely interested in specific parts and happily leave the course after their information need was satisfied, many students drop out because of a lack of time, or overstrain with the course content.

Usually researchers take the last event, such as logging in, of a student as the date of a dropout [157, 181]. In order to detect dropouts, supervised learning techniques like support vector machines, hidden markov models or logistic regressions have been used [62, 69, 81, 156]. The exact features used to classify dropouts vary between the approaches but mostly consist of a mix of *clickstream data*, *grades*, *social network analysis* [181] and *biographical information*. While the classifiers are considerably accurate (85% to 91% within the same course [174]) the cited work often lacks suggestions on how to prevent students from dropping-out.

We want to address this issue by intervening on students who struggle with our exercises. As suggested by Taylor et al. [157], we focus on features which relate students to other students as such features are more predictive than average grades or login durations.

Zheng et al. report that also a missing sense of community decreases retention and thus increases dropout [185]. They furthermore state that "existing MOOC platforms do not provide features to promote community awareness". Although some students are actively trying to build up a sense of belonging, either by browsing student profiles or looking into the forum, the outcome was mostly negative. Being presented mostly anonymous nicknames and no interesting information, they afterwards refused to say "hi" in the forum or to further provide personal information themselves. The missing appraisal of fellow classmates and instructors diminishes the feeling of success and recognition. As a logical consequence, missing connections and trust in the community also limit help-seeking strategies.

In a study examining students' reasons for dropout based on data from over 20 MOOCs, Kizilcec and Halawa found four main clusters (in declining order of significance) [77]:

1. time issues
2. course difficulty
3. format and content
4. goals and expectations

[5] see `http://www.katyjordan.com/MOOCproject.html`.

Of these four clusters, only some are affectable by instructors. Students' time issues (1) reside without the reach influence of instructors. Mostly the same holds for the area goals and expectations (4). Rohloff et al. show that surveying students' goals might improve learning outcomes, however, they could not yet reach statistical significance [131]. Clearly communicating prerequisites clarifies expectations towards the course, and as this should be given for every MOOC, it cannot be further influenced. The areas course difficulty (2) and format and content (3) are under control by instructors and thus bear potential for experimentation and improvement. It is therefore our primary intention to reduce what we call "content induced dropout" by supporting students during their negative, critical phases.

To uncover students' general progress as well as encountered issues, learning analytics are employed in MOOCs. Educators lost the aforementioned "glimpse over the shoulder" of individual students, but in return received potentially huge amounts of clickstream data. The field of learning analytics is relatively young, as it relies on the availability of "big data" in the educational sector. Despite the vibrancy of the field, a corpus of generally accepted knowledge is already present (e.g. the Handbook of Learning Analytics [90]). Further, this area builds upon established disciplines including data mining, data visualization, and psychology [44]. Learning analytics, despite offering magnitudes of data, have to be grounded in theory to allow for directed research [177]. Drawing from an established area, a potential model for the research of this thesis can be derived from information seeking. Same as information seeking, learning is an effort to close individual knowledge gaps. Therefore learning models can be related to the existing models of information seeking. The many models for information seeking behavior mostly vary in their scope and complement each other [176]. Given the different focus, not just on retrieval of information but on the understanding of it and subsequent construction of knowledge, additional aspects are introduced for the learning context and existing aspects shift their importance.

A comparison of recent studies within MOOCs together with their respective underlying learning theories [178] finds that most researchers base their experiments on the theory of self-regulated learning (SRL). The ability to structure and reflect one's learning affects students' success in traditional online learning [14] as well as MOOCs [92]. The theory achieved relevance and is therefore in vibrant scientific discussion, resulting in a variety of models that structure the idea of self-regulated learning, most often by dividing it into phases [115]. According to the model of Pintrich [118], self-regulated learning consists of four phases named (1) forethought, planning, and activation, (2) monitoring, (3) control, and (4) reaction and reflection. Other models, including the one from Zimmerman [186], divide the activities into three phases, in this case named (1) forethought (task analysis, self-motivation), (2) performance (self-control, self-observation), and (3) self-reflection (self-judgment, self-reaction).

While naming and number of phases differ between these two most prominent models, the concepts expressed by the phases overlap. For example, it becomes apparent that the phases 2 and 3 of Pintrich's model are reflected in a merged fashion in phase 2 of Zimmerman's model. In the following, we will stick with the model from Zimmerman. In general, all models contain phases that comprise activities prior to an activity, working on the activity, and after working on the activity. The models are cyclic, reflecting that either each activity can be imagined as many sub-activities, or that each task may be interrupted and recommenced after some additional reflection.

For our research, especially the last two phases are of increased interest, with a clear focus on the performance phase. Figure 2.2 shows Zimmerman's model reduced to the aspects that will be covered within our research.

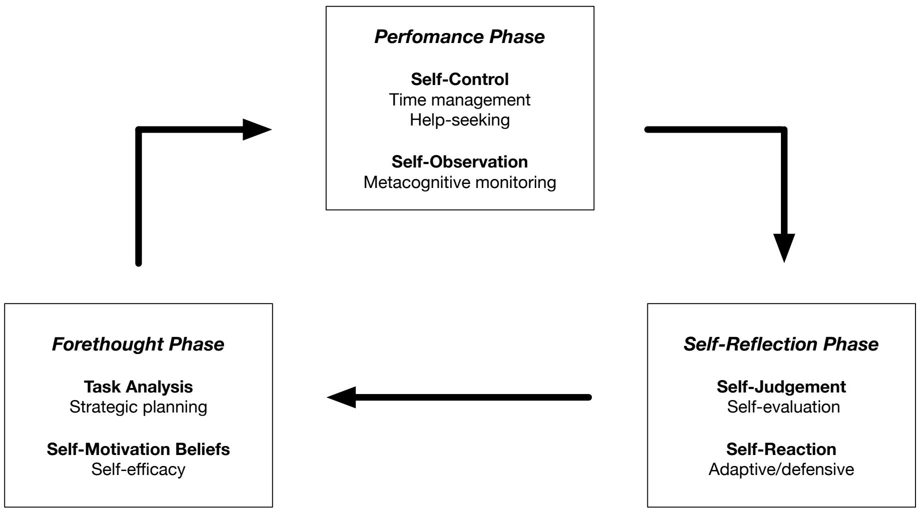


Figure 2.2: Cyclical phase model of self-regulation from Zimmerman and Moylan [187] reduced to aspects relevant for our research.

Students solving a programming exercise usually do not spend much time on a forethought phase, they just open the exercise and begin working on it, in this case by reading the exercise description. In case the student does prior planning on program design (e.g. structuring of loops), this counts as strategic planning in terms of task analysis. Also, prior experiences affect the self-efficacy, therefore being depicted in this phase. Within the performance phase, we plan to achieve the largest improvements for the students with our approaches. Especially the time management and help-seeking (a social form of information seeking) activities are within our interest. Metacognitive monitoring might be affected since success or the lack thereof will especially be remembered after a student seriously struggled. Coming to the Self-Reflection phase, students' self-evaluation as well as their adaptive or defensive behavior to the (temporarily) achieved outcomes are of interest for our research. Students' self-evaluation in our case will heavily be influenced by the achieved unit test results. From prior courses we know that most students persistently aim to achieve full score and will be demotivated to a large extent if they are missing the slightest fraction of the maximum score. However, also the step from zero points to at least some points usually comes with a boost in motivation and a positive self-evaluation of the student's progress. Especially when struggling, student's behavior should be adaptive to our belief, therefore we will implement measures to push students towards this behavior.

The cyclic flow of the model represents that subsequent phases will be affected by their predecessors. For example, strategic planning in the forethought phase will likely affect time management. Likewise, the outcomes of the performance phase will largely impact students' self-evaluation, and their reactions will subsequently affect their self-efficacy for future exercises.

The performance phase is of particular interest for us, as struggle can only happen if a student is actively working on an exercise. Students often do not know when to step back if they are focusing on an error [184]. To help, we aim to induce some sort of self-reflection with the goal to interrupt the current line of thoughts which lead to the situation where the student got stuck. Potentially, approaching the issue from another direction or seeking for additional input, for example by re-watching the lecture video or searching for additional examples on the web, might enable students to overcome the problems themselves.

The less established structure in online education increases the potential for struggle and requires more self-regulation from students [89, 155]. Compared to in-class education, there is further an absence of a given timely structure, as normally given by lecture hours, and educational guidance on individual level, as normally offered by the teacher or teaching assistants [78]. Consequently, demands towards students' qualities concerning self-motivation and determination are even higher.

2.1.3 Zone of Proximal Development

A central issue in distance education, that inherently also applies to MOOCs, is that students have to face problems mostly on their own. Whenever a student faces an issue in the learning progress, probably caused by a lack of prior knowledge, incomplete information presented in the learning material, or an occurring misconception, chances are high that the student might get stuck. Without an instructor to guide the learning efforts, the first necessary step is that students realize that they face a problem. For a lack of prior knowledge or the presented information being incomplete, the problem usually becomes apparent to the student naturally, whereas a misconception often remains undetected and might cause more severe problems in later learning stages, when advanced concepts base on misunderstood fundamentals.

Lev Vygotsky developed a model called "zone of proximal development" to reflect stages of development and learning processes for school children in the early 1920s [169]. His model has since been adapted and enhanced, and nowadays provides a basic distinction between task difficulties. The main distinction is between three continuous difficulty levels that present themselves for individual students: tasks that a student can do alone, tasks that a student can do with help and tasks that a student cannot do, even with help.

As an example, the task to write a "hello world" program, is a task a student can do alone after watching an introductory video that covered the basics of the respective programming language. A slightly more difficult task, to incorporate a call parameter for that program and print out a passed-in name to that "hello" program, is likely too demanding for a novice. However, on the basis of that initial hello world program, a novice is often able to enhance the program with the help of a more skilled tutor to solve the more difficult task. This task

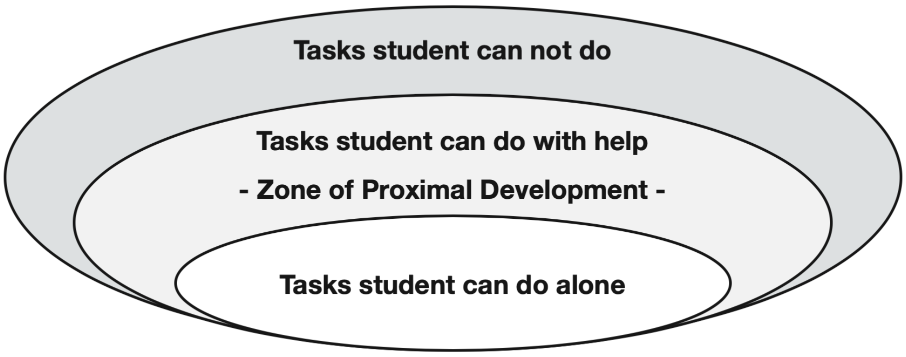

Tasks student can not do
Tasks student can do with help
- Zone of Proximal Development -
Tasks student can do alone

Figure 2.3: Zone of Proximal Development, according to Vygotsky [169].

would therefore fall in the so-called “zone of proximal development”, which is the area where the largest potential knowledge growth is expected in classical education. Whereas Vygotsky required more capable and experienced teachers for the helping teacher or peer in the zone of proximal development, Ohta claims that adult peers do not necessarily need to be more capable in order to provide assistance [113]. Vygotsky’s model was only slightly adapted by Otha on the basis of foreign language learning and is generally assumed to be applicable for in-class settings. However, the basic model currently does not incorporate the effects of literary sources that improve students’ comprehension with additional material, such as textbooks, worksheets, or in our case, videos and additional training exercises.

With two of our proposed interventions, we specifically aim to help struggling students into the zone of proximal development. The results of this thesis will thus further contribute to the yet open question concerning the applicability of Vygotsky’s model in the field of online education.

2.1.4 Effects of Tutoring

Tutoring describes the process of supporting an individual learner or a group of learners by another person. Collins English Dictionary describes it as “remedial or additional teaching, designed to help people who need extra help with their studies”[6]. So in a broad sense, the term tutoring covers all activities having a student receive additional help in understanding from either another human being or a computer agent.

In literature, different types of tutoring are presented: academic coaching, in-home tutoring, distance tutoring, online tutoring, and peer tutoring. General academic coaching aims to improve general skills to improve study success, such as time- and stress-management, research and (academic) writing skills. Specific academic coaching covers all help given to improve knowledge in specific subjects (e.g. quantum physics or theoretical computer science). In-home tutoring mostly comes in the form of private lessons to improve school grades and pass critical

[6] see `https://www.collinsdictionary.com/dictionary/english/tutoring`

tests. Distance tutoring, in Germany also often called “tele-tutoring”, covers all tutoring which is provided from a distance, be it via phone, email, moderated use-groups, or postal mail exchange of study documents. Online tutoring is the most prominent variant of distance tutoring and nowadays mostly supplanted all other forms of distance tutoring, given the low costs, general high flexibility, and low latency compared to the other variants. For graded assessments in non-it-related subjects, postal mail exchange is still common of today, but also these fields show a rise of alternative options to hand in students’ solutions digitally. Tutoring is classified as peer tutoring if the role of the tutor is not represented by the course instructor or a dedicated teaching assistant, but an individual of the student body. This setup comes with several advantages as well as disadvantages. Advantages include better scaling and lower psychological barriers to ask for help. A meta-study of Robins et al. further points out, that peer learning shows its benefits especially when dealing with practical tasks [130]. On the negative side, peer tutoring needs increased organizational setup, and the quality of tutoring cannot be guaranteed without monitoring [164].

Effects of tutoring are a well researched field since the early eighties, with results being presented and collectively supported in various meta-studies [5, 26, 27, 41, 59, 164]. All of the meta-studies agree that school tutoring programs increased academic performance for students being tutored. These students outperformed control groups on examinations, furthermore, they showed a more positive attitude towards the topic being dealt with. Additionally, students acting as tutors also showed positive effects towards performance as well as attitude.

For example, Cohen et al.’s meta-study of 65 independent comparative studies of tutoring shows that 87% of the studies measuring learning effects via scores report positive effects towards the students being tutored. Considering the tutors, even 90% of the studies report positive effects on their scores [27].

Having a closer look onto the subjects being taught, most studies deal with reading skills, elementary scientific topics, or understanding of mathematical principles. In general, the subject matters of reading skills and mathematical principles however clearly dominate the studies (>90%) [27].

Allen and Feldman conducted their study dealing with topics including reading, math, as well as science. They found that tutoring improved the results of tutors, even if they are deemed “low achievers” in the respective field [3]. In their study, they let “low achieving” fifth-graders tutor third-graders over a timespan of two weeks. They showed significantly better scores than a control group, which studied alone and did not convey any tutoring. The results of the tutees were affected positively, but not significantly according to their study. Albeit their low participant count of 7 fifth-graders, their research indicates that the process of trying to uncover others’ misconceptions has a positive impact on one’s own learning efforts, even if the current skill level of the tutor is low. Either by being exposed to different challenges or by re-visiting prior content, learning is reinforced.

Most existing studies and meta-studies on tutoring were enacted on elementary and secondary school students, given a face-to-face setting. The transferability of the results thus has to be ensured for the field of distance education. Furthermore, the applicability to a wider audience including adults currently working in their career has to be revalidated.

When comparing the effects of face-to-face tutoring to online tutoring, Price et al. found that online tutoring is deemed as inferior compared to face-to-face tutoring [119]. The type of tutoring analyzed in their study on students of the Open University is mostly academic coaching. The gathered ratings are based on CEQ (Course Experience Questionnaire) [122], RASI (Revised Approaches to Studying Inventory) [43], and a distance education centric subset of the AEF (Academic Engagement Form) [129] questionnaires. Price et al. conclude that the inferior rating is likely caused by students' inappropriate expectations towards online tutoring. Further, they express that the limitations induced by the online environment hinder the communication and interaction between tutors and students, which arise from technical aspects as well as missing experience for effective online communication.

2.1.5 Personalized Learning

The general setting of a student mainly interacting with a computer while learning not only causes the aforementioned downsides of reduced human interaction but also offers positive potential, in the MOOC case technically increased feasibility for personalized learning. With the aim of optimal learning outcomes in terms of understanding, student motivation, and knowledge retention, it is therefore helpful to counter the potentially decreased motivation of students caused by the lack of individual feedback and peer interaction with better-suited material.

Achieving a good instructional design thus not only includes a mindful design of the learning content concerning its structure, accessibility, and appropriateness for the audience, but also means that the available content should be tailored towards the individual learner.

Personalized learning as a term has not reached an agreed-upon definition in the scientific community yet, however, the common understanding and agreement is that it generally describes a setting that offers each student an individual learning path. This personalization can, for example, be carried out by a teacher in school classes. For larger settings that cannot be feasibly prepared and supervised by a teacher, technical systems can be employed to tailor the learning [17].

Adaptive learning is one approach to implement personalized learning in a digital environment. It is characterized by the fact that a system uses the data a student produces while working on educational content in order to subsequently adapt future content and exercises.

The research area of Intelligent Tutoring Systems (ITS), having evolved from the field of so-called computer assisted learning (CAL) covers this in specific. The general structure of ITS consists of four so-called models (or modules): the domain (or expert knowledge) model, the student model, the tutoring (or teaching) module, and the user interface module [109, 111, 112]. For the sake of comprehensibility, we will just shortly describe the models to the extent necessary for our concepts and refer to the existing literature for an in-depth coverage [112]. The domain model reflects concepts and steps to learn the desired skills, and therefore serves as a kind of blueprint for the optimal progress. The student model represents the state of the student with regard to the domain model. Initially, the system has no information about a student's prior knowledge and skills. As the student interacts with the domain content, the system traces the progress, thereby gathers data and builds up the student model. Therefore, the student model and the domain model are inherently interdependent, and the student model can be regarded as an "overlay" over the domain model [48]. This overlay approach allows to represent a student's knowledge as a subset of the domain model to be aimed for. Students' weaknesses then can be expressed as deltas of the student model from the domain model. Based on information from the domain model and the student model (i.e. how the optimal progress should be and how the student progressed), the tutoring model decides for steps to improve the learning outcome (e.g. suggests specific content). Lastly, the user interface model holds data to interact with the student using the system. This interaction can be direct, e.g., by alerting the student via popup windows containing text or images, or indirect, e.g., by lowering or increasing difficulty or deciding which additional content to be displayed next.

Recommender Systems

An often employed implementation of student interaction with an intelligent tutoring system to convey personalized learning offers different paths through the same given material, therefore only changing the order of content blocks, or allowing to skip known content. Another implementation variant is recommending content based on prior actions, for example a specific video which repeats a topic a student has shown deficits in.

Offering such student-specific content also belongs to the field of *"Recommender Systems"*. Nowadays, recommender systems are found in almost all places. A prominent example, which is also suitable to exemplify the first of different types of recommendation approaches, is online shopping.

Collaboration-based Recommendation, also called *Collaborative Filtering (ColF)*, is often used to recommend products. The system identifies users that are likely to share interests and suggests target users a number of potentially interesting items based on the data of similar users. The approach requires a large amount of user data on the one hand, but is independent of product specifics on the other hand. Collaborative Filtering can further be divided into two sub-approaches, that differ in the way how similarities are found.

1. *User-based Collaborative Filtering* starts from given user characteristics (e.g. age, current location) in order to find similarities among users to find potentially matching users. This approach is problematic for new user profiles which are lacking viable information.

2. *Item-based Collaborative Filtering* starts its calculations based on similar items, in order to find matching users. So if a new user is interacting with a product A, thereby expressing interest, the system determines users who also liked product A and is then able to generate recommendations based on the combined preferences of these users. This results in the often seen "users who liked X, also liked Y" recommendations in web-shops.

Content-based Recommendation, also known as *Content-based Filtering (ConF)*, directly relates users with content. Opposing to the Collaborative Filtering approach, the indirection via similar users is skipped. Consequently, Content-based Recommendation relies on information the user has already expressed concerning his or her interests. If we know a user likes cooking, but dislikes gardening, we can recommend products that are directly or indirectly related with cooking, but share little connection with gardening. In a learning environment, instead of products, we usually recommend different examples or exercises, hopefully better fitting students' interests and hobbies, creating a better identification this way. In order not to force users to explicitly express their interests, necessary information can also be collected during the normal interaction of the student with the system, e.g., via dwelling times on pages or clicks on respective website areas.

Additional to their predominant applicability in e-commerce, recommender systems have received wide acceptance in the field of education in general and technology-enhanced learning in specific [98]. In the context of education, the same concepts as in e-commerce are used, with the sole difference that products are replaced with training exercises or additional learning content.

2.2 Differences Between In-Class and Distance Education

All universities - respectively its staff - have a working knowledge about "how to teach", that is not externalized as books or checklists, but as experienced knowledge and know-how, carried by individuals [103]. Teaching staff and lecturers usually acquire basic didactic knowledge passively during their school education, or their bachelors and masters studies [38]. On top of that, the excitement as well as the curiosity of the lecturers or seminar teachers, sometimes being responsible for their class on their own, keeps the teaching fresh and dedicated. This implicit knowledge is usually passed on from generation to generation of PhD students. The knowledge further feeds from the experience of the professors and is further quality controlled by them [103]. However, this "folklore how to teach", comes to an abrupt end when the walls of the university are left behind. Distance education is, albeit still present on some higher educational institutions, a much smaller field, despite the often much larger reach and output. Another step further is distance education via electronic measures, the so-called "e-learning". E-learning is often used as a supportive measure to complement presence studies with extensive material or more complex exercises. Both distance education and e-learning require new skills and knowledge [103].

2.2.1 Drawbacks for Learning Programming at a Distance

Whenever dealing with distance education, it is important to remember that the core principles of education and teaching remain the same, but the surrounding conditions differ and may do so to a great extent. The most striking differences when comparing MOOCs with in-class courses are the number of students enrolled in MOOCs and the absence of direct personal communication. Both differences make it difficult for instructors to support struggling students. Offering individual feedback personally requires excessive time, not available to course conductors. A reasonable alternative might be peer assessments, but they usually require considerable effort to set up as well as a large amount of the students' time and can therefore only be applied once within a typical course runtime [86].

In-class education allows teachers to glance over the shoulder of their students to notice potential struggle, assess the current situation and give direct feedback. Within online courses, this glance over the shoulder is not possible on large scale. Therefore an automated and scaling solution is required to detect struggling students and supply them with helpful feedback and additional training options.

This research provides new insights and general learnings on various levels of programming assignments as well as online distance education. The primary drawback of MOOCs is the impaired student-teacher relationship between students and course conductors. On the one hand, students may develop a sense of familiarity with the teachers performing in the lecture videos. Probably caused by constant exposure and role distribution, participants may even develop admiration and kind of an imaginary friendship towards the lecturers, as often observed with YouTube stars or other so-called influencers [87, 101]. On the other hand, the lecturers only receive selective and mostly textual feedback, thereby alienating the student-teacher relationship into that direction.

2.2.2 The Rise of Collaborative Work and Problem Solving

User requirements and expectations towards nowadays software are high and diverse: software is expected to run stable, fast and efficient on a multitude of devices, offering intuitive user interfaces. These requirements usually cannot be met by a single developer. Therefore, different experts contribute to achieve the desired results.

Knowledge-intensive professions also involve a high need for communication, to onboard new team members to running projects or to inform team members about changes in architecture or design decisions [39]. Given the abstract nature of computer science, additional challenges arise: a common vocabulary regarding the wording has to be established, common diagram styles to convey structure and relationships, and on top of that shared behavioral patterns to orchestrate large numbers of individuals. When dealing with software, developers have to know their tools of trade, including a solid understanding of the underlying hardware, and knowledge of multiple programming languages, as well as common design patterns. This general, fundamental knowledge is universal and can be taught at schools or universities beforehand.

On top of that comes specific business and application knowledge, the so-called "business logic", and previously taken design decisions with regards to the modeled process and software. Several types of artifacts, such as class diagrams or sequence diagrams, created following the uniform modeling language, have evolved in order to externalize project-specific knowledge. These challenges arise in smaller scopes, given agile software development teams of usually less than ten persons, as well as larger scopes when coordinating dozens of those teams to work on standard-software. Having large companies trying to apply a so-called "follow the sun" approach for globally distributed software engineering to reduce time to market, additional challenges with regards to cultural differences arise [165].

Looking towards the "non-professional" developments within knowledge intensive areas, also semi-professional work, such as crowdsourcing, requires a common understanding of the task at hand, the methods used and the deliverables as well as the quality expected. Within the strictly non-for-profit areas, even more examples of collaborative work and problem solving have come up. Knowledge databases such as Wikipedia offering general information, Stackoverflow focusing on computer-science specific information, or even everyday knowledge collections such as the German platforms *fragMutti*[7], or recipe databases like *chefkoch*[8] were built by vivid communities only able to achieve their shared goals together. Many different incarnations of knowledge platforms with differing requirements, rules, and etiquettes contribute to modern everyday life and underline Pea's claim, that knowledge is commonly socially constructed [117].

Humankind in general seems to have found opportunities to contribute to a common shared goal, mostly out of altruism and reciprocity, giving back as gratitude for help prior received.

Another important incarnation of collaborative work in software development is so-called "social coding". Social coding is an approach of socio-technical development that emphasizes collaboration [33]. It emphasizes that modern software development is an activity performed in potentially large groups and that open communication and information sharing within those groups is essential for the success of the software. Therefore, aspects such as awareness about progress and development artifacts, have been reevaluated given the changed conditions, e.g., loosely-coupled teams of hundreds of individuals in open-source projects as opposed to hierarchical development units in companies. It emerged that visibility of artifacts did not overburden developers and users, but fostered interaction and focused communication, especially when questions could not be answered due to missing information or transparency [33]. With the increased importance of relations between developers in mind, supportive tools were developed to visualize the social connections within projects, allowing to better identify suitable collaborators, e.g., to pinpoint and fix a bug [135].

[7] see `https://www.frag-mutti.de`
[8] see `https://www.chefkoch.de`

The term “social coding” was mostly coined by GitHub[9], and comprised the opportunities to distribute, share and modify code as well as discuss it. Other platforms such as Atlassian’s BitBucket[10] and open source tools like GitLab[11] and Phabricator[12] share this understanding. Major software companies established products in this area to support development further gain influence, but the market has settled by now. GitHub emerged as the market leader and was acquired by Microsoft in 2018. The offerings by Google and Microsoft were archived in the years before, with Google Code being archived in 2016, and Microsoft’s CodePlex in 2017.

Discussion on those platforms is centered around parts of the source code, namely new commits and pull requests, instead of the entire codebase. The aim of these approaches is to maintain a stable and reviewed codebase, maintained by an active community of already skilled developers.

Our approaches share the aim to foster collaboration in the area of software development with social coding platforms, but our approaches differ in the area of application and thus surrounding conditions. Our main goal is to improve learning. In contrast to platforms like GitHub, the resulting artifact, usually source code, is of no importance for the success of the overall effort. Also the scope differs with the small, self-contained exercises presented in MOOCs. Opposed to the focus on commits, which represent an incremental improvement to a larger code base, the fundamental structure of the (usually short and simple) program is the primary interest in education. Also, when discussing commits, the underlying assumption in modern software development is that the supplied code is tested and working, in contrast to the submissions made in programming exercises. Thus the focus on social coding platforms is on discussing design decisions and complicated issues between trained software engineers, while discussions on education platforms will evolve around fundamental misconceptions, syntax errors, and grading issues.

When having a closer look into the establishment of online communities, it is helpful to distinguish between different types of online platforms, depending on the approach of presenting knowledge. Platforms revolving around foundational, structured knowledge usually conveying smaller pieces of knowledge with the aim to embed them into a greater picture usually follow some kind of a course syllabus. These platforms resemble the school model and are suitable for example when learning a new language. Other knowledge platforms, in contrast, offer specific, unstructured knowledge. An example for this approach of knowledge offering is a database allowing to look up specific cooking recipes. Communities mostly evolved around unstructured knowledge bases.

Reasons for this are twofold: First, unstructured collections offer more options and freedom to participate, as changes and additions have fewer side effects. Second, a corpus of suggested and ordered material inherently requires some instance of power to decide which parts are presented, in which length and which order, carrying the seed for dispute.

[9] see `https://github.com`
[10] see `https://bitbucket.org`
[11] see `https://gitlab.com`
[12] see `https://www.phacility.com/phabricator`

Therefore most MOOCs, requiring at least fundamental structure (e.g. a topic, a start and end date, as well as technical administration), are led by a team of instructors and thus face the aforementioned drawbacks with regards to community building at least partially.

Depending on the desired structure and model of learning within the course, MOOCs have been classified into the "frontal" e**x**tension MOOCS (**x**MOOCs) and the more community centered **c**onnectivist MOOCs (**c**MOOCs).

Connectivist MOOCs are built with certain principles in mind, favoring aggregated material from the community over pre-selected material of a teaching team. This leads to evolving material that is mostly remixed, remixable and repurposable. The material, therefore, acts as building blocks for future students, allowing for a so-called feed-forward. Depending on the implementation, course instructors give out initial reading material, or just research directions and potential tasks or learning goals. The MOOCs on so-called "rhizomatic learning" by Dave Cormier are an extreme example of cMOOCs, in which even the curriculum itself was set to debate under the topic "The community is the curriculum" [28]. Challenges for these format are the coordination of students with regards to tools, effort, and means of communication. Additionally, most students are not used to this way of learning, resulting in comparably lower enrollment and finishing rates, if applicable at all.

xMOOCs, short for extended MOOCs, are originally recorded university courses, that have been enriched with additional material such as multiple choice quizzes and further reading material. They resemble the traditional "ex-cathedra" teaching style with an instructor presenting content and students required to listen. Despite common criticism of the approach, this style of MOOCs offers benefits for students and instructors alike. First, student and instructor roles are well known from in-class education. Second, students can consume the offered material at their own pace. On top of this, students struggling with any material can revisit it at any time, without interrupting the course flow or impeding others. Furthermore, it allows switching in-class education to so-called "blended learning", combining individual content consumption and collective in-depth discussions [12]. Instructors are freed from the necessity to re-enact basic lessons over and over again, leaving time to further improve the material or to give feedback. Over the past years, the majority of xMOOCs has increased in accessibility and quality with regards to different aspects: most often, the recordings are no longer a byproduct of a regular on-campus lecture being recorded, but are an independent project with multiple stakeholders involved, established best practices and ongoing quality assurance [56, 58, 88, 144].

Summarizing, collaborative work is not only pervasive in computer science and programming, but also has become an integral part of education, and in specific MOOCs. The extent, to which students are on the one hand merely "consuming", or on the other hand actively contributing towards the course content, depends on the general course format and differs with each individual course. While cMOOCs will have a generally much higher contribution ratio, also xMOOCs can strongly benefit from collaborative efforts, for example in the form of peer feedback and vivid forum discussions.

2.2.3 Mismatch of Workload (Teaching Team, Students)

When conceptualizing and designing a MOOC, two core metrics have to be kept in mind for all decisions: the required time for each individual student and the available time of the teaching team. The enormous mismatch between the sizes of the two groups, usually less than five instructors and teaching team members on the one side, and thousands of students on the other side, makes intuitive reasoning impossible. Carefully determining the weekly combined effort for students to follow all offered activities, including videos, texts, self-tests, and exercises is necessary in order to prevent time induced dropouts and possibly allow some additional room for forum discussions. On the other hand, the teaching team has to allocate time for forum administration, bugfixing and content improvement, mail support, technical support, general announcements, and many more unforeseen activities. While some tasks require a fixed amount of time, e.g., weekly mailings, other efforts such as forum administration, scaling with the number of students enrolled, and maintenance efforts for technical stability, scaling with the complexity of the employed system, are hard to estimate.

As the required effort also depends on the experience of the teaching team members, we cannot give reliable estimation guidelines here. The necessity of a supportive audience being capable to help each other out on occurring questions in the forum becomes evident with the numbers presented in Table 2.1.

Course	#Students	#Middle	#Forum Posts	#Solutions	#Submissions
imdb2012	13,629	12,066	2,729	0	0
python2014	7,376	6,598	8,844	130,722	not recorded
python2015	17,401	7,840	6,395	119,047	2,373,204
java2015	11,581	10,607	11,101	254,972	2,881,780
java2017	10,402	8,781	7,673	237,724	3,580,143
java2018	21,693	18,873	6,222	229,124	3,397,509

Table 2.1: Course key metrics for selected programming MOOCs openHPI and openSAP, retrieved in June 2019. imdb2012 (non-programming MOOC) is included for comparison.

The number of students #Students reflects the absolute number of enrollments, the metric #Middle reflects the number of students enrolled at the day when half of the course's runtime has passed. It therefore better represents the active audience by excluding enrollments that happened after the course runtime (and, e.g., increased the enrollment number of python2015 by almost ten thousand). The number of forum posts easily reaches several thousands within each course. This implies that the most important forum questions can still be answered by the teaching team, but monitoring all questions asked and answering them in the expected detail induced by the relatively polished course videos, becomes almost impossible. When adding additional complexity, as with programming exercises containing student-specific errors that have not been localized yet, the possibly required time investment cannot be brought up, even not by a teaching team that is willing to spend their leisure time and weekends.

Technical approaches to scale the effectiveness of instructor tutoring exist [46, 53], leveraging the number of students' solutions an instructor can review at once by synthesizing similar solutions and outlining relevant differences, thus reducing visual clutter such as duplications and highlighting potential error sources. This approach is helpful in general, but still requires an extensive amount of time from instructors. It potentially delays the point in time where instructors will be overburdened, but cannot tackle the issue of scaling from the conceptional perspective.

Suitable solutions for MOOCs optimally should scale the offered learning support linearly with the respective audience. Peer-education and content-centric collaboration within the audience thus become fundamental cornerstones for MOOC grade scaling, given the high complexity caused by the multiplying factors of a large MOOC audience and the increased individual technical depth caused by practical programming exercises.

2.2.4 Factors Impacting Learning Outcomes

Yousef et al. conducted a survey to find the most important factors impacting the design quality of a MOOC [183]. Additionally to giving a good overview about important criteria in MOOCs and further providing a convincing categorization of these criteria, their paper outlines the categories "learning analytics" and "assessment" as the most important ones. In general, it is reassuring to find many of the most emphasized metrics being considered and well integrated into the technical foundations of our MOOC platform (e.g. user interface, provided features including download options, and slide- and teacher-view) as well as individual course design (e.g. video length or communication of learning goals). The highlight on the two categories "learning analytics" as well as "assessment" emphasizes once more that actual exposure to exercises and students' possibility to reflect on them are deemed as crucial for learning. Further examining these two categories, top mentioned metrics for learning analytics include "Provide recommendations and feedback for learners to improve their performance." and "Provide performance report to learners.", while the category assessment brings up "Each quiz should give feedback and/or show the correct answers.", "Provide integrated assessment within each task." and "Each assignment should have hints." within its top criteria. These top-ranked criteria thus encourage instructors to mindfully design the optimal learning progress of their students not just content-wise, but also from the perspective of a holistic learning experience.

He et al. additionally highlight that learner support is of high importance concerning students' motivation, retention, and success [61]. The availability and guidance of instructors or teaching assistants largely influences learning outcomes. Martin et al. further argue that timely feedback in online education is especially crucial to lead students to "higher levels of learning" [100].

Feedback, thus seen as a central and integral part of learning, however, cannot be incorporated conceptually into the video lectures, which are the most prominent element of MOOCs. This is due to the fact that videos are consumed passively by the students from the viewpoint of the platform. Students have to actually do something in order to be able to receive feedback on their progress. The most basic approach of pursuing that is adding short multiple-choice quizzes after content videos. Having students pick the correct answers serves mainly two purposes: first, they are required to reflect on the presented concepts, and second, they provide kind of a submission on which feedback can be given on. In the case of MOOCs and the previously mentioned size of the audience, this feedback is usually automated. The most basic version of feedback on a multiple-choice quiz is simply providing the information whether the chosen option was correct or incorrect. This binary result is the most important information from the technical viewpoint. But from a didactical point of view, this minimal feedback on the students' probably complex decision is insufficient and misses the majority of available potential for learning. A better alternative is to also provide a short description, what the correct answer is, why it is the correct one, and which reasons or line of thoughts lead to this answer. This allows the students to follow the steps that bring them to the correct solution, and comprehend the reasons for correctness. Questions designed that way therefore can provide additional insights, recapitulate presented concepts, illustrate their application, and offer students the possibility to reflect. Even better than only to build on the correct answer, is to also provide feedback on the wrong options. Explaining which misconceptions might lead to wrong options, and outlining why they are wrong, gives students the possibility to find and understand the point where they diverged from the correct reasoning. Additionally, explaining incorrect options yields a benefit also for those students who solved the question correctly: being presented with possible but incorrect options, further strengthens understanding, as often those options just are not chosen because one did not think of it.

The design of a simple multiple-choice question therefore determines its usefulness in learning. As a consequence, the benefit of it ranges between being perceived as a basic check whether a fact was remembered correctly to an opportunity to embed a presented concept into a scenario. Embedding a concept in a scenario allows to reason on the correct applicability and thus helps in pinpointing common pitfalls, preventing further struggle and subsequent misunderstandings.

The given example on multiple-choice quizzes is most suited to explain the general issues occurring when designing content with the main goal of providing automated feedback to students. Quizzes with predetermined options represent the most restricted and therefore controlled activities possible. These boundaries limit the directions students can take and therefore also limit the explanations to be provided.

Revisiting the criteria presented by Yousef et al. [183], well-crafted quizzes therefore provide performance feedback, integrated assessment, and also hints. Offering those aspects also on other activities without such clear borders imposes a set of challenges. Depending on the actual activity, for example an open-ended question requiring a student to provide an answer within a free-text field, different conceptional and technical measures are necessary. In the case of the open-ended question, one might either approach the issue with three different approaches. The first approach is through technical means by automatically searching for certain keywords in the answer. The second approach employs organizational means by applying some form of peer-assessment and peer-feedback. The third option is to guide and assist the student to perform self-assessment. Combining technical and non-technical means usually yields the best results. However, it becomes apparent that the potential approaches largely differ depending on the activity being performed by the students.

In our case of learning programming, we introduce practical programming exercises on top of the proven foundation of multiple-choice quizzes. Similar to open-ended questions, students are free to enter everything they want in their source code editor. This however opens up an unlimited field of possibilities to make mistakes and to encounter extended struggle.

Despite the possibilities for encountering troubles are endless in general, several factors narrow the problem space in our scenario and thereby reduce the necessary complexity to support students with helpful feedback. First, the domain of programming knows certain common errors or groups of errors that occur more often than others. Second, each programming exercise comes with an exercise description, intended to guide the student into the direction of the solution, thereby marking the way together with the course video placed before the exercise. Depending on the level of detail, the exercise description might even give a step-by-step instruction on how to solve the exercise, thereby imposing some sort of soft-boundaries. Third, the exercises come with template code, effectively reducing foreseeable issues, e.g., confusion occurring on mixing up syntax when students first work with classes and methods, without having gained a thorough understanding of the concepts behind them. Fourth and last, some errors are especially common for beginners, so that they are more or less expectable at certain stages in the course and can be countered to some extent with more extensive descriptions and comments in the respective exercises. The consecutive nature of exercises being presented in a course introduces students to a desired way of thinking in a step by step manner, thereby often limiting specific errors and misunderstandings to specific exercises.

2.2.5 Applicability to K-12 Education

The average MOOC student on our platform openHPI is male, between 30 and 50 years old, holds a bachelor's or diploma degree and has prior knowledge in the tech sector. He takes the courses out of individual interest and motivation, mostly in his spare time, which he is able to schedule freely [50, 51].

This drastically differs from the situation of school children. They are externally motivated to follow the content, do the learning mostly during their classes, and are accompanied by their teacher. The given weekly format of MOOC content does not account for time limits like 45-minute lessons in German schools and does not necessarily group the videos and exercises into self-contained units. Additional structuring is thus beneficial, enabling teachers to better motivate concepts and allowing them to further add a repetition at the end of a lesson to fortify acquired knowledge [63]. In general, MOOCs do work in school context (and are frequently used as they offer a high-quality foundation for class discussions), but they need to be stretched out over the course of a school year [152]. MOOCs offer material that teachers can work with, but do not allow the teachers to further tap into the additional benefits the platform offers, such as detailed analytics. To check the progress of their students, teachers have to glance over the shoulders of their students individually. This allows to spot individual errors, but is impeding aggregated insights such as common misconceptions. Despite the differences with regard to the general prerequisites in online-only MOOCs and therefore the main target direction of this thesis, the scenario of K-12[13] education is included on purpose. Accustoming school children to approaches for life-long learning and modern education tools is actively requested by teachers, desirable from the goal of a general and all-embracing solution, and furthermore offers additional insights concerning content consumption and real-life issues [138]. Major differences between the online scenario and the school scenario are:

1. Student motivation (internal vs. external)
2. Content segmentation (course weeks vs. school lessons)
3. Grading (automatic vs. individual by teachers)
4. Feedback (unknown individual vs. classmates and teachers)

In order to keep explanations concise, if not stated otherwise, all descriptions refer to the "normal" MOOC context.

2.2.6 Design of Programming Exercises

A central part of programming education are the practical exercises themselves, as they offer the best opportunity to deepen and fortify understanding. Such programming exercises, apart from potential technical challenges, come with a series of questions to be addressed, for example: which fraction of the students' time should they take (compared to video lectures and other course activities), which difficulty should be aimed for, how much guidance should be offered and how much repetition should be incorporated? The perceived difficulty of a task depends on previous knowledge, supplied hints, the required time for solving and the number of failed attempts the student made. Furthermore, the detail and accuracy of the problem description, the restrictiveness of the applied test cases and the preparation provided specifically for a given exercise also influence the perceived difficulty of a task.

In general, Massive Open Online Courses are, as massive and open implies, intended for broad audiences. An optimal practical programming exercise should therefore appeal to all students, challenge them but also be solvable with reasonable effort in a predefined timespan. The individual challenge thus relies on prior knowledge, and the term massive in MOOCs naturally implies that the students taking a course bring in a wide range of prior knowledge in many areas, be it directly connected, adjacent or unrelated with the topics covered in the course. While this spectrum of knowledge is a profitable foundation for discussions in the forum, there are also occasions where this range is hindering learning outcomes. In wide audience settings, like forum discussions, a considerably large group of students is reading questions written by a much smaller share of students expressing those questions. This automatically circumvents harmful knowledge gaps, as those students that fit for the actual information needs will interact, while all others act as silent bystanders and eventually also

[13] Abbreviation for Kindergarten to 12th grade.

learn by passively reading. Given more narrow settings, like discussions within small groups or peer-assessments, the knowledge gap between the students will not regulate itself. Such a skill gap can either be conducive, as an experienced user explains concepts that have not been understood beforehand and is lead to new thoughts by questions that have not come to mind before, or be cumbersome, as the experienced user might be bored by elementary questions. Also, two students being on an elementary level, are most likely not best suited to find the solution to their problem without further external help.

2.2.7 Skill Assessment

Knowledge differences can thus either be helpful or hindering, depending on the actual setting. In order to steer potential outcomes and gain benefits of the knowledge differences, the prior knowledge of the participants has to be assessed.

The assessment, in this case, is neither really formative, meaning that it is not intended to build feedback upon or serves as a starting point for an intervention, nor is it summative, as it is not used for grading. In order to coin our assessment, we call it an informative assessment, as it is primarily used to improve potential actions in the future. This is close to a formative assessment, however it lacks the necessity of an intervention afterwards.

For the following approaches and descriptions, we want to state that whenever we speak of an absolute skill expressed in numbers, we are aware that this numerical value cannot reflect the true knowledge, experience, and mastery of a topic. It is not intended to rank students in kind of a high score and to display these individual values. On the opposite, the skill levels will be used internally to optimize the learning outcomes of all students.

Assessing knowledge and skills is difficult in general. Companies spend huge amounts of money on elaborated approaches like headhunters and assessment centers to find the right candidates for job offerings as wrong decisions come at an even higher price. Commercial providers like AMCAT[14] or others build their entire business model around the assessment of skills in various areas. In contrast, the assessment within MOOCs does not have to offer such fine granularity and does not bear high financial risks if it is inaccurate, which makes the problem less severe. In our context dealing with programming education, we also focus on a rather technical area, which tends to have a better expressiveness in numbers than for example communication skills. However, the given MOOC setting also adds other difficulties. For example, participants cannot be bothered with long quizzes or too excessive or too delicate questions. As they take part in courses mostly based on intrinsic motivation and without direct career goals, posing too cumbersome hurdles will only result in participants skipping the questions or in the worst case quitting the entire course.

Having gained some insights on students' prior knowledge, the actual programming exercises come into focus. Programming exercises that were not chosen well for the individual participant have several downsides that can result in a variety of negative effects. Exercises being too easy will not challenge students enough.

[14] see `https://www.myamcat.com`

While easy success might increase motivation over the first few exercises, it will increase the risk of frustration when facing exercises with a higher difficulty, as one got used to passing without effort. Especially students having a higher prior knowledge than that being aimed for in the target audience will be bored by exercises being too easy. Losing already high skilled students may seem bearable in the context of classic distance learning, since having a homogeneous group of beginners allows to optimize the course videos and additional course material to their specific needs. In the context of MOOCs, losing experts however would be a huge drawback due to the resulting absence of their know-how and possible support.

Having advanced practitioners or even experts of a programming language within the field of students potentially yields massive benefits. In past courses we conducted, some motivated experts helped out on various occasions, ranging from pinpointing ambiguities in videos, wording and slides over answering upcoming questions to supplying suited links or even creating additional material. Last but not least, advanced and expert users answered forum posts in quality, length and speed simply impossible to the teaching team, as it was bound on other tasks such as technical support and additional content creation.

2.3 Ethical Considerations and Data Privacy

MOOC research heavily relies on user-generated data. This brings up the issue of data privacy from a legal viewpoint, as well as questions of ethical correctness from the philosophical viewpoint.

Prinsloo and Slade give an overview over the development of the discussions and aspects with regards to ethics in learning analytics [120]. While it is generally agreed that higher education institutions "have a right" to collect and use student information in order to ensure effective learning and support students, questions of consent and potential downsides have to be discussed.

Concerning user generated data, the information we deal with falls into two different categories: data supplied directly by the user and data gathered through user interaction with the system. So apart of the data that users enter on the MOOC platform directly, such as for example their email address, name, gender, birth date, highest degree, career status, and affiliation, the MOOC platform as well as the programming exercise platform automatically create usage data, as users interact with them. Intrinsically, also the majority of this data contains sensitive information about users, such as their achieved scores, time spent on exercises, and weaknesses in specific knowledge areas.

Upon registration, all users agree to our privacy policy as well as our code of conduct. The privacy policy[15] clearly states that the MOOC platform saves user data and that these data are used for research in compliance with the german law as well as the GDPR[16]. In the process of this research, we analyzed the data only in an anonymized and aggregated form. Furthermore, we only drew conclusions from aggregated data of reasonably large participant groups.

[15] see `https://open.hpi.de/pages/data-protection`
[16] Abbreviation for General Data Protection Regulation.

The field we are operating in is the area of educational experiments. In order to fundamentally prevent the possibility of assessing or even penalizing students on the basis of their demographics, all test group assignments were based on artificial, uniformly distributed variables. Randomly assigning users to experiment groups still raises the question, whether it is justifiable to probably support or hinder participants. We agree with Justin Reicht[17], who argues that these discussions should take place on all levels of education to preserve public trust through proactive research ethics. Discussions we had on scientific conferences brought up several times, that the experiments researchers are conducting in a controlled setting, have lower estimated effects than changes caused by chance in ordinary in-class education. Assigning a new teacher to a school class, or only sending in a supply teacher once, usually has stronger effects on class performance than enabling or disabling a certain feature in a MOOC. The actual difference is, that the effects of "changes by chance" are usually not measured and thus cannot be quantified.

Apart of that, we rolled out the experiments in several stages, aiming to only improve the support and therefore learning experience as well as success for parts of our audience. In order to measure the effects, we thus conducted A/B tests to quantify the potential differences in metrics caused by the experiments.

Specifically, the experiment which was considered the most controversial by us was the last stage of a series of three experiments. Letting a group of users ask for help and even encouraging them to do so, while at the same time preventing the possibility that they receive feedback from fellow users, intuitively seems wrong and potentially might even cause negative outcomes such as increased frustration and consequently lower scores. Therefore, we decided to conduct this experiment in 2018 only after we learned that we could not infer a negative impact of that behavior from former experiments in 2017.

Summing up, only experiments with a desired, potentially positive outcome were conducted.

2.4 Status Quo in Online Programming Education

The area of programming education is diverse, both in terms of normative didactical approaches as well as diverse situations in different countries. The overview presented here is kept intentionally short to outline the different approaches, viewpoints, and stakeholders without losing itself in details.

The first approach to structure the current situation of programming education is from the organizational viewpoint. Federal education guidelines differ significantly between countries, and while politicians agree that IT is an important and eligible discipline, progress in externalizing this in curricula is differing to great extent. Considering Germany, education is in the authority of the federal states, resulting in a fragmented status quo with various curricula, and therefore also resulting in a lack of performance norms and a comprehensive strategy for the

[17] see the blog article “The Ethics of Educational Experiments” published in 2014 on `http://blogs.edweek.org/edweek/edtechresearcher/2014/07/the_ethics_of_educational_experiments.html`.

future. Recently, German governance announced the “DigitalPakt Schule”[18], to support the digital transformation of schools.For historical reasons, the money will be invested to improve the infrastructure in schools. The federal government is not allowed to directly fund skilled practitioners or improve educational content itself. This shortage of content and know-how, therefore, leaves room for other stakeholders, mainly organizations.

Organizations come with own agendas, differing whether the organizations are for-profit, or non-profit ones. Businesses (for-profit organizations), including MOOC platforms, usually offer paid programming courses as part of their revenue model. Not requiring large and expensive tooling such as in mechanical engineering or medical preparations in medicine, opportunity costs are low, whereas potential and expertise required for scaling are given in that field. Therefore, evening classes, seminars, boot-camps, and private colleges, emerged alongside state-funded offerings. Non-profit organizations mostly focus on delivering introductory knowledge to young people or specific underprivileged groups in society. In between are for-profit organizations that sponsor smaller initiatives for marketing and social responsibility reasons that are consequently offered free of charge.

The second approach of structuring the education landscape focuses on the didactical methods. Differences between didactical approaches become most apparent when comparing the different prior knowledge that freshman students bring to university. While some computer science classes in school mostly dealt with application knowledge, using spreadsheets and text processing applications, others learned imperative and object-oriented programming. Foundational knowledge, including knowledge about computer architectures, data structures, and propositional calculus, may or may not be part of a specific course syllabus. The wide variety of potential focus points cannot be covered exhaustively in the time available for computer science classes in school. Therefore, the usual result is some basic knowledge of programming, mixed with some slight introduction into logic reasoning. Existing online offerings are avidly incorporated into classes on individual teacher basis [97], as well crafted material is often too time-consuming to be created individually. Practitioners favor that well-crafted offerings cover and integrate multiple knowledge areas, such as programming, data structures, and boolean logic at once [153].

When initially teaching programming skills, mainly one of three major options can be followed:

1. *imperative first*, aiming to explain program flow similar to a cooking recipe carried out step by step
2. *objects first*, aiming to explain program states and structures with responsibilities
3. *block-based approaches*, mainly following the imperative logic but further abstracting from program code

[18] see `https://www.bmbf.de/de/bund-und-laender-ueber-digitalpakt-schule-einig-8141.html`

In this thesis, we will focus on languages based on plain text source code, and put the imperative first as well as the objects first approach into practice. Our research was carried out with students of the MOOC platforms openHPI and openSAP, who used the code execution platform CodeOcean to practically learn programming. In the following, we shortly describe specifics of the MOOC platforms used for our research and classify the used code execution platform by comparing it with other offerings.

2.4.1 Coding Platforms

Auto-grading suites and online coding platforms exist on the internet since some decades. With the recent maturity of containerization solutions like Docker[19], their absolute number has increased tremendously. During the time of creating this thesis, several platforms have come up, as well as several platforms have ceased to exist or have been acquired by major education or tech companies. Nevertheless, the categorization of platforms we will introduce in the following paragraphs are expected to remain stable.

Former studies and surveys published before 2013 by Queir'os et al., Ihantola et al., and Douce et al. focus on technical aspects and integration options mostly [35, 67, 121]. With the upcoming of MOOCs happening just later, these comparisons do neither cover the currently most significant offerings nor the resulting new categories. Also, the most recently published review of automatic assessment tools we found from Ullah et al. published in 2018 [166] does not cover the aspect of massive open online courses. To our knowledge, recent comprehensive surveys in this particular direction thus have not been published yet. Therefore, we will shortly outline the most relevant fields in the context of this thesis, each accompanied by the most prominent implementations for the respective field.

In general, the existing platforms will be categorized into five main categories. The categories used here are *Auto-Graders*, *Code Challenges and Riddles*, *Recruiting Platforms*, *Bounty Hunt Marketplaces*, *Online IDEs* and *Course Platforms.* The separation between the categories, especially for the categories code challenge platforms and interview platforms is often difficult and was done depending on the main use case of the platform, i.e., recruiting is the main revenue source for interview platforms' business models.

[19] see `https://www.docker.com`

Auto-Graders

Auto-Graders are basically programs suites, that run students' submitted solutions against a collection of hidden and visible unit tests in order to provide automatic grading. Most online programming platforms comprise an auto-grader. Therefore, the solutions mentioned here are distinguished from the others because they specifically focus on the very task of auto-grading and do not provide additional content or an online Integrated Development Environment (IDE) alongside them. Auto-graders are mostly found in the academic field.

One of the first published auto-graders in the scientific domain is TRY, published by Kenneth Reek [124] in 1989. Based on the UNIX shell, it allowed students to submit their programs and enabled a secure execution of students' programs against supplied test files to support grading.

The next major step was provided with a graphical user interface and the support of different test categories in the tool ASSYST, developed by Jackson and Usher from the University of Liverpool in 1997 [68]. Despite being a very early tool, it already separated and supported assessment categories such as correctness (solution passes unit tests), efficiency (CPU time spent, number of statements executed), style (e.g. indentation, comment ratio), complexity (McCabe's cyclomatic complexity metric [102]), and test data adequacy (test coverage). A recent version of ASSYST is not available as maintenance has stopped.

Most similar is Web-CAT[20], being developed at the Virginia Polytechnic Institute and State University (Virginia Tech) since 2003 as an open-source auto-grader primarily intended to teach students test-driven development (TDD) [40]. It integrates via (LTI) and currently supports unit tests as well as static code analysis. Development activity on GitHub shows a stable software with some occasional bug-fixes during the last years, however, no new substantial features have been added recently.

JACK is a system for computer-aided assessments and exercises being developed at the University of Duisburg Essen. It focuses on Java and C++ exercises and provides capabilities for static and dynamic program checking [154]. Furthermore, it allows generating math exercises with parametrized content for school usage [136].

DOMJudge and Mooshak are auto-graders primarily intended for programming contests, focusing the presentation on leaderboards and ranking information.

The presented auto-graders are designed to support in-class education respectively coordinated contests. As they are "solely" auto-graders, they inherently lack capabilities to further support online-education, most notably a suitable online IDE with advanced feedback mechanisms surpassing the mere passing of error messages.

[20] Hyperlinks to the several coding platforms are summarized in the appendix in Table 9.7.

Code Challenges and Riddles

Code challenges, also called code riddles or programming puzzles, are mainly used for three reasons: as a personal challenge mainly for fun, for hiring situations, and to solve open issues from other parties being reimbursed with recognition or cash. The most notable platforms for fun challenges are project Euler offering mostly mathematical challenges, CodingBat offering Java and Python Challenges, CodeWars and Sphere Online Judge (SPOJ). Although slightly different in their approach, the “daily programmer” subreddit posting a challenge each day and the codegolf sub-exchange from Stack Exchange are worth to be mentioned here, as they have large community impact.

Another different approach to learning programming via code challenges are coding games, such as CodinGame, and CodeCombat. They embed algorithmic problems within game scenarios and are therefore often more appealing to younger people.

Usually, challenges can be solved in almost all established programming languages (including OOP languages such as Java, C++, Python, and Clojure as well as scripting languages such as Javascript or Lua). Free selection of the used programming language is possible, because code challenges often only test for correct output values, whereas the games require the students to call specific methods of their supplied “protagonist” objects.

These exercise collections mostly address already experienced programmers, as prior knowledge is required to interpret the abstract problem description with no additional learning material provided.

Recruiting Platforms

Especially with regards to interview and recruiting platforms, the current number of offered solutions is so diverse, that only a small selection of well-known tools is presented here. HackerRank, HackerEarth, Codility, CoderByte, Qualified, and CodeSignal basically all offer the same product: an online IDE with a shared editor for coding exercises in technical recruiting. Several recruiting platforms come with challenges (such as HackerRank’s Interview Preparation Kit, HackerEarth’s CodeMonk list, or the separate platform CodeWars mentioned above which is based on Qualified). The platforms encourage users to solve the challenges in order to test and improve their programming knowledge. From the company side, this is intended to do fingerprinting on their users, thereby generating valuable data to improve their matching processes and promote hiring chances.

Bounty Hunt Marketplaces

TopCoder and Bountify each offer a marketplace for developers to solve open issues against payment. These commercial crowdsourcing platforms separate from other task or job matching agencies through their sole focus on software development. Both platforms list code review and debugging tasks. With a focus on data science, Kaggle offers an additional marketplace revolving around data analysis problems.

As a free alternative, exercism allows to download source code with open issues to be solved. It focuses on code practice and mentoring for individual growth. Same as the commercial platforms, exercism primarily acts as a "marketplace" and does not come with a code editor.

Feedback on programming code is therefore a valuable asset being traded or gifted on public platforms.

Online IDEs

The mentioned online IDEs focus on technical aspects and feature support to distinguish themselves. Educational content is not offered in the projects itself. In this overview, we focus on text-based IDEs, omitting block-based IDEs like Scratch or its modifications BYOB, Chirp, or Snap! used on well-known platforms such as code.org. The reason for this is the fundamental different approach. Although we considered doing the initial introduction to abstraction via variables and control structures with block-based languages, we decided against this approach. This decision was mainly made due to the possible hurdle that students face when required to use a text-based IDE and the danger of scaring away both younger and older participants with a programming environment that is deemed to look too playful and childish.

The most relevant internal difference between different text-based online IDEs is the fact whether they run user code server-side or client-side. Client-side IDEs must execute all code locally in the browser, operating systems containing potentially malicious code in a sandboxed environment. This comes with two advantages: (1) it eliminates all security issues for potential course instructors when running unknown code and (2) keeps CPU load local, preventing peaks and overload situations on servers on imminent deadlines. As browsers can only run JavaScript in their execution engines, the most common approach is to compile student supplied code from other languages to JavaScript.

JSFiddle and CodePen are well known minimal javascript and CSS rendering solutions without any direct integrations into MOOC grading available. Such solutions are mainly used to showcase small solutions with regard to web-development problems. However, by additionally supplying obfuscated JavaScript that transforms students' output to a keyword to be entered in a MOOC quiz, these client-side solutions already offer some basic assessment options [145]. Quite similar is the live-editor of the Khan academy. It is used for introductory courses in the Khan academy and additionally offers a block-based editor frontend. Going further with regard to available programming languages, Skulpt offers a Python environment emulated via JavaScript, whereas Opal offers the same for Ruby and ClojureScript for Clojure. Weblinux from Rémi Sharrock even emulates a Linux environment in JavaScript, which is additionally able to compile C code [140].

Server-side IDEs offer a much higher variety of supported languages and tooling, like a debugger for example. Given the improved functionality, server-side IDEs often resemble the look of desktop IDEs to offer a coherent and intuitive user interface. The greater flexibility comes at the price to carry out CPU intensive workloads, such as compiling and running student-specific code, centralized on a server. On the other side, this also allows the platform operator to gather additional data to improve.

Gitpod, repl.it, and CodeAnywhere are commercial server-side code execution platforms. repl.it further offers a paid extension called “classroom” aimed at in-class usage providing functionality specific for teachers and their students. Codio is a commercial platform focusing on computer science education in high schools and universities, offering pre-built learning material as well as authoring tools for instructors. Institutions have to buy yearly licenses based on the number of students they wish to train.

Microsoft offers Small Basic Online free of charge, an online environment especially to help students transition from block-based coding to text-based coding using a variant of the BASIC programming language.

Next to commercial offerings, there are also open-source solutions for server-side online IDEs. Under the umbrella of the Eclipse foundation, there are three projects that count as general online IDEs: Orion, CHE, and Theia. Orion is the oldest of the three mentioned ones and basically consists of two parts: a client representing the IDE frontend and a server used for compiling and running code. The client and server, despite being two different modules, are however quite depending on each other, reducing flexibility to adjust or replace program parts. CHE builds upon Orion and uses the same code editor, but replaced parts of the server and shifted the focus towards managing workspaces. Workspaces are kept inside docker containers, making them easily shareable and transferable. Theia is the most recent project of the three mentioned, building on CHE technology. It uses the CHE backend server but replaces the frontend client. Older Java GWT based technology was replaced with a more recent TypeScript application, furthermore reducing the coupling between client and server with the aim to enable the frontend being used on other backends as well.

codeboard.io, is an academic-based open source project very similar to CodeOcean. It was started by Christian Estler at the ETH Zurich’s Chair of Software Engineering and published to Github on December 2015. Its main use cases are supporting programming MOOCs as well as on-campus education. From the user interface, it closer resembles the look of a full-fledged IDE such as Eclipse, and allows every user to create new exercises (or projects in their terms). It was used in an accompanying MOOC by the ETH Zürich on edX in 2015 and also by an Introductory Java MOOC by Universidad Carlos III de Madrid in 2018. Development seems to have stopped in January 2016, leaving it feature-complete and “‘MOOC proven” but without ongoing development.

PythonTutor has a different target direction: it allows to visualize pieces of code, including the program flow and object structures. Despite the name, it supports a variety of languages including Java, C++, and Ruby. Another key feature is the option to help other students struggling with their code in a synchronous session. Editor content and program output are synchronized, while offering the participants a chat session to communicate.

Summarizing, online IDEs are available in different variants, each with advantages and disadvantages. For our intended usage offering free Java and Python courses starting in 2015 and complying with German privacy policies, the requirements lead to the development of the program execution platform CodeOcean, further described in Chapter 4.

Programming Course Platforms

The largest course platforms solely focused on programming are Pluralsight's CodeSchool, Codecadamy, Treehouse, and CodeAvengers. All platforms require a subscription for productive usage (with monthly costs ranging between 20 and 40 USD). They each offer different trial models (Codecadamy offers small intro courses for free, while the other platforms offer trial periods between 7 and 30 days).

Content and production quality is high on all three platforms: they offer individually scripted, professionally edited videos and for some courses also coherent storylines. Each of these three platforms uses a customized online IDE, offering all required functionality. Treehouse also offers subscriptions including personalized feedback on code, priced at 199 USD per month.

A mostly free alternative is SoloLearn, usable without a paid subscription. Content presentation is mainly via text and quizzes, with occasionally embedded programming exercises. Stepik, a course platform mostly featuring Russian content, is another free alternative. Content is mostly delivered via text, with pictures and videos embedded occasionally only. For code execution, Stepik uses repl.it.

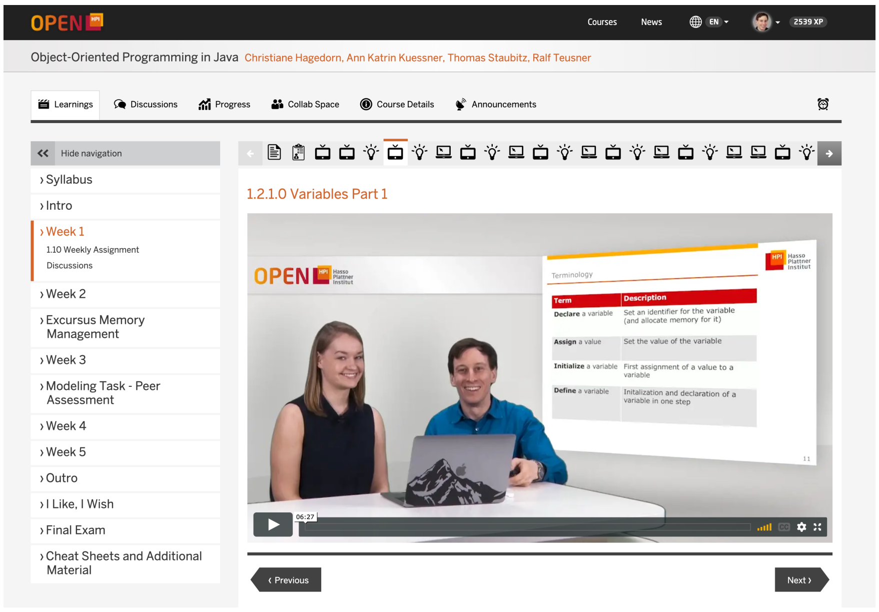
Programming course platforms thus either require a subscription fee to access high-quality material or only offer basic content without coherent course videos.

Definition: Programming MOOC

In the following, when we speak of a programming MOOC, we refer to a programming course satisfying all MOOC criteria. This means that the course is massive (in our case more than 5,000 students), open (available for everyone, usually free of charge) and delivered online. On top of that, a programming MOOC has to offer an opportunity to practically try out the conveyed concepts, thus it has to provide an online code execution platform (or a client-side browser alternative) with embedded exercises. Last, we demand the exercises to be an integral part of the grading mechanism, thereby requiring the execution platform to feature an online judge component.

2.4.2 Course Platforms: openHPI and openSAP

openHPI is a German MOOC platform developed by the Hasso Plattner Institute (HPI) and hosted at the institute itself. Established in 2012, it focuses on IT related topics and offers courses in German as well as English. At the time of writing this thesis, it has over 200,000 registered users with over 600,000 course-enrollments in total. The typical user interface for a student is shown in Figure 2.4.



Term	Description
Declare a variable	Set an identifier for the variable (and allocate memory for it)
Assign a value	Set the value of the variable
Initialize a variable	First assignment of a value to a variable
Define a variable	Initalization and declaration of a variable in one step

Figure 2.4: Typical impression of an openHPI course from the students' perspective.

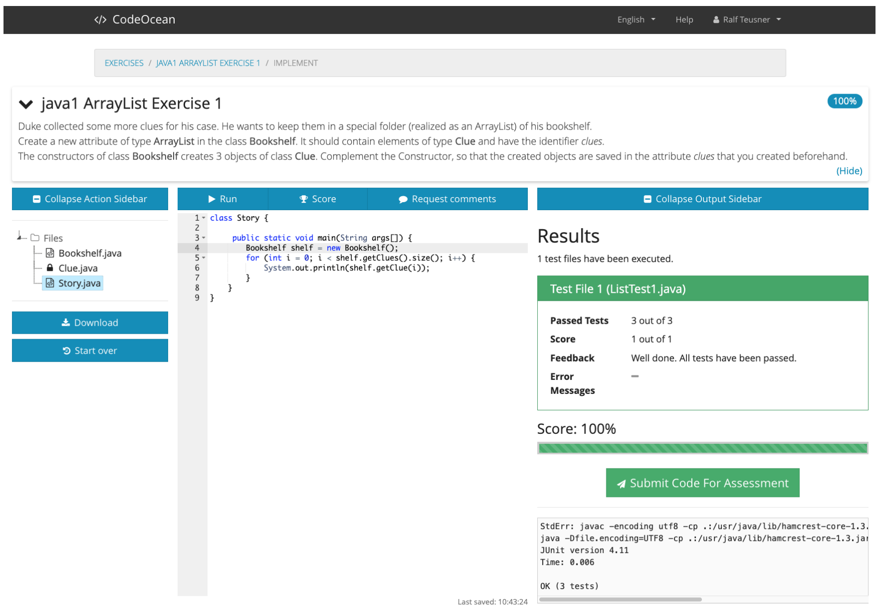
Students follow a course schedule usually divided into weeks and progress through videos, self-test quizzes, graded assignments, peer-reviewed team-tasks, and additional learning material, such as custom reading material, book chapters, or even practical exercises and games. Over the course runtime, students gather points by completing graded assignments. If they access at least 50% of the offered learning material, they receive a "Confirmation of Participation" at the course end, and if they managed to score more than 50% of the maximum possible score, they are additionally awarded a graded "Record of Achievement". The course forum is the central place for interaction between students, with additional sub-forums available reflecting the weekly course structures.

Additional MOOC platforms, such as openSAP, openWHO, and mooc.house are also running on the same software, adding further huge amounts of users served by the system (e.g. for openSAP over 800,000 students with over 3.4 million course enrollments).

Starting from a heavily customized Canvas system[21], the software was replaced by a custom-built microservice architecture mostly written in Ruby, to allow for a better customization towards the needs of a MOOC platform. The distinguished situation to provide content on the one hand, and control the backend of the platform on the other hand, provides the HPI with the opportunity to adapt both vital parts of MOOCs to support comprehensive research fueled by full access to anonymized data.

This content and data integration further allowed to easily connect other software prototypes to enhance learning success.

One of these additions, which evolved into a stable platform itself, is CodeOcean, our code execution and grading platform, further described in more technical detail in Section 4.1. As can be seen in Figure 2.5, students are presented with a working environment composed of an exercise description at the top and core parts of an IDE below. Most importantly, students can run and score exercises with numerous files, with the option to receive automated feedback instantly. Additionally, they can also download their current progress or reset it, in case they went into wrong directions and want to start anew. In case they require help, they further can request help from their fellow students. The concepts behind these collaborative measures will be explained in the next chapter.



java1 ArrayList Exercise 1

Duke collected some more clues for his case. He wants to keep them in a special folder (realized as an ArrayList) of his bookshelf.
Create a new attribute of type **ArrayList** in the class **Bookshelf**. It should contain elements of type **Clue** and have the identifier *clues*.
The constructors of class **Bookshelf** creates 3 objects of class **Clue**. Complement the Constructor, so that the created objects are saved in the attribute *clues* that you created beforehand.

```java
class Story {

    public static void main(String args[]) {
        Bookshelf shelf = new Bookshelf();
        for (int i = 0; i < shelf.getClues().size(); i++) {
            System.out.println(shelf.getClue(i));
        }
    }
}
```

Results

1 test files have been executed.

Test File 1 (ListTest1.java)

Passed Tests	3 out of 3
Score	1 out of 1
Feedback	Well done. All tests have been passed.
Error Messages	–

Score: 100%

```
StdErr: javac -encoding utf8 -cp .:/usr/java/lib/hamcrest-core-1.3.
java -Dfile.encoding=UTF8 -cp .:/usr/java/lib/hamcrest-core-1.3.jar
JUnit version 4.11
Time: 0.006

OK (3 tests)
```

Figure 2.5: Students' view of CodeOcean on a typical Java exercise.

[21] Open source LMS available at `https://github.com/instructure/canvas-lms`.

3

Approaches to Foster Collaboration in MOOCs

Within this chapter, we motivate and explain the specific concepts we implemented to bridge the gap between students and instructors as well as to foster collaboration in MOOCs. Section 3.1 describes the general approach in order to improve learning outcomes. In Subsection 3.1 we classify our implemented measures and also shortly describe approaches that lie outside the main focus of this thesis. The following sections present the developed approaches to improve learning results by fostering collaboration in MOOCs and improving content quality. Based on the structure introduced with Figure 1.1, approaches to better understand struggle (Section 3.2), to intervene on struggling students (Section 3.3), and to adapt weak course material (Section 3.4) are described in detail. The respective subsections explain the concepts of video tutoring, automatic anomaly detection of content, just-in-time interventions, code commenting, and tailored bonus exercises in greater detail. Especially the three concepts mentioned last will receive increased attention, as they will provide the majority of the data used in our evaluation. Parts of Subchapter 3.2.1 have been published in [159], respectively of Subchapters 3.2.4, 3.3.3, and 3.3.4 in [160], and Subchapter 3.3.1 in [163].

3.1 Overall Concept

Improving learning outcomes and bridging the social gap between students and instructors in MOOCs requires a broad and holistic understanding of the domain of e-learning. Even though individual proposed changes and interventions are often situational and specific in terms of their applicability, each measure has to be carefully embedded into its surroundings in order to yield the desired results. Clearly outlining all factors that influence the processes to be improved and describing the steps taken within our approaches in this chapter ensures confirmability and credibility of the effects measured later on. Therefore, we will recapitulate the main learnings so far and then connect this information with approaches to improve the status quo.

To shortly summarize our motivation and background, we are aiming to teach programming to a large and diverse audience. Hands-on programming exercises help to improve the quality of ICT education, as they are well suited to convey practical experience and deepen understanding. However, programming exercises also bear potential for numerous misunderstandings, extended struggle, and demotivation.

Extended struggle is a threat to learning, therefore we present potential options to limit those risks. Non-constructive struggle occurs usually when overburdening students by giving them tasks of much too high difficulty. The issue of assessing prior skill levels and choosing optimal exercises must therefore be addressed. Necessary information to track students' progress and measure the effects of our interventions is subsequently outlined. The proposed and implemented interventions interfere with students' learning experience and potentially affect their course results. Upcoming ethical questions were therefore carefully considered and addressed on the basis of learnings from pre-studies.

The remainder of this chapter presents the ideas and conceptual designs of our implemented approaches in detail. Video tutoring connects two or more individuals directly to provide instructors a virtual "glance over the shoulder" of students and granting the students an experience that is closest to traditional in-class tutoring.

Given our MOOC setting, it is nonetheless obvious that individual tutoring by instructors is not feasible with the large number of students in the courses. We therefore also propose an asynchronous mechanism to provide struggling students with individual feedback based on crowd-sourcing, which allows to scale organically with the audience. This measure called "Request for Commentss" requires students to actively take action and ask for help, which is known to be a hindrance to acceptance. Humans, consciously or unconsciously, hesitate to admit a lack of knowledge and approach strangers even when remaining anonymously if desired. To counter this issue, we also introduce the concept of just-in-time interventions, which encourage students to take action in the moment we assume them to be struggling.

A concept facing students that does not require any additional student interaction is the concept of so-called "tailored bonus exercises". This concept includes a mechanism that grants students the option to receive additional training exercises specific to their weaknesses. The tailored exercises are selected based on information which we extract from a knowledge model build on their previous progress.

Lastly, we describe our concept of automatic anomaly detection to pinpoint weak material. This approach grants instructors the option to quickly improve weak material still within the course runtime. It thus reduces unnecessary demotivation of students who have not yet encountered the previously weak content.

Overview of Approaches

Considering the background of online education, we outlined in Section 2.1.2 that Kizilcec and Halawa found mainly four areas inducing student dropout [77]. We argued that the two areas "course difficulty" and "format and content" offer potential for improvement, while the other two areas "time issues" and "goals and expectations" are mostly out of the reach of instructors' influence.

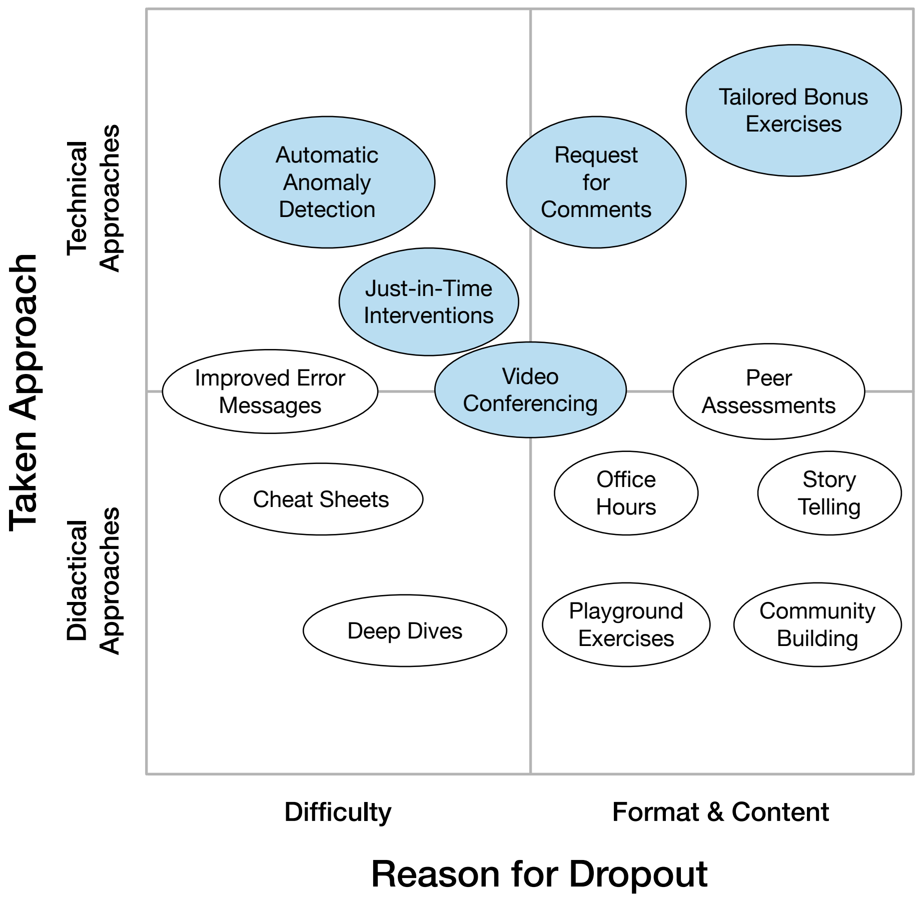


Figure 3.1: Classification of implemented approaches. Approaches showing a blue background are explained in greater detail in subsequent chapters.

In Figure 3.1, we classify our approaches and measures based on the reason for the dropout. We therefore depict the two areas inducing dropouts that are within our reach, "difficulty" as well as "format and content", and place each of our approaches with regard to the reason for dropout we want to counter.

Additionally, we also explained that the social gap found within online education has different dimensions. Albeit being a distance between humans and therefore already implying to opt for a solution on the social dimension, we outlined that the gap is essentially induced by current technical shortcomings.

The solutions space therefore spans over the two areas of technical, as well as didactical approaches.

A clear distinction between those dimensions is not always possible (nor is it favorable), as the borders are blurry and most approaches have a technical as well as didactical share, potentially affecting course difficulty as well as the course format. Nonetheless, the matrix provides a high-level overview and classifies the implemented measures on the basis of their predominant aspects concerning the taken approach as well as the tackled dropout reasons.

Within the offered MOOCs, all of the depicted approaches were applied. In the following, we will first explain the approaches that we will not cover in further detail in this thesis, and then give an overview of the approaches that build the foundation for the detailed experiments discussed in this thesis.

In the lower-left quadrant of Figure 3.1, deep dives and cheat sheets can be found. Offering concise summaries of the presented course material potentially lowers the difficulty, especially for students without prior knowledge. Within the field of programming, such a cheat sheet can prevent struggles caused by unknown syntax and misspelled keywords. Further providing additional video lectures, that cover difficult topics in greater detail and aim to convey additional background knowledge, is likely to lower the difficulty for more advanced students. The supplemental background information as well as additional examples prevent misconceptions and help in distinguishing the connections and links between the presented concepts. Deep dives, as well as cheat sheets, did not require any additional functionality to be implemented in the MOOC platform, they were simply delivered just as normal course content videos, or downloadable documents respectively. We placed the cheat sheets closer to the technical approaches area than the deep dives, because they were focused on technical syntax details regarding their content and the delivery as downloadable documents separated them further from the normal course workflow centered on videos. Deep dives, on the other hand, slightly differed from the normal content presentation style that was followed in the course videos. Facts were presented in a more direct way, and the examples usually covered more conceptual depth, instead of explanatory width. With so-called office hours, frequent questions and often occurring problems were addressed during the course runtime. The video recordings differed in so far, as they were captured with an ordinary webcam, instead of employing the recording studio, resulting in a more relaxed and approachable impression. The playground exercises are in the same area. Coming without any achievable points and unit tests, they invite students to try out concepts for themselves. Given template code and suggestions for implementation presented as source code comments thus set a different stage compared to the graded exercises in the main course corpus. Additional didactical approaches mainly differing on the format and content side were the introduction of a story within the courses and additional community building in the forum. The story was mainly told via optional videos embedded in the main course flow, and additional hints to be found in programming exercises as well as forum threads. Students were further given the opportunity to influence the progress of the story to a limited extent, which was implemented via weekly polls. One of the main aspects of introducing a story was to add another motivating factor to stay in the course in order to follow the small story arc revolving around a detective solving cases. The other main aspect was to give the students additional, non-technical topics to discuss about in order to foster community building. Other followed approaches to help establishing a community were introductory threads inviting participants to introduce themselves and share some of their background, showcasing of exceptional solutions and other contributed learning material, as well as actively promoting well-written forum answers of students. By posing controversial and tricky questions within ungraded self-tests, it was possible to further spark discussions in the course forum. All these approaches did not require changes on the MOOC platform, as they either completely reused the available options (community building, playground exercises), or just required slight alterations (storytelling decisions via polls implemented as self-tests, or office hour videos created within a different production pipeline) of existing processes.

Approaches that required technical changes are for example improved error messages that are displayed within the programming exercises. Additional parsing and translation steps of occurring errors had to be introduced in order to make them technically possible. Because enhanced error messages do not affect any student-workflows in learning, and primarily aim to lower the difficulty by improving the understandability of the feedback, we still placed it on the border between technical and didactical approach on the difficulty side of the figure. Peer assessments, although heavily influencing students' workflows, were also placed on the border between the technical approaches and the didactical approaches. They require many technical capabilities to be applicable, including an initial assignment of work to be carried out, formation of groups for so-called group-peer assessments, or numerous paths for exception handling to cope with all sorts of anomalies that occur within a group of thousands of students. However, peer assessments also heavily influence the didactical dimension. Submissions are no longer graded by an automatic test, but are judged by fellow students on the basis of pre-defined rubrics and open-ended feedback. Aspects such as documentation and understandability of supplied solutions massively rise in impact compared to the previous, unrelenting focus on correct syntax due to technical reasons. Therefore, given both the high technical requirements and impact, as well as the great didactical change, peer assessment was still placed between the two vertical areas. Concerning the horizontal distinction on the basis of the potential dropout reason ranging between content difficulty on the one hand and format and content on the other hand, peer assessments are clearly located on the format and content side. Through peer assessment, difficulty is not lowered at all. Creating, assessing and grading submissions as part of the peer assessments is of higher difficulty than most other exercises and requires initial training. With regard to the format of the exercise, peer assessments come with plenty of other quality criteria, thereby training important aspects otherwise neglected due to technical shortcomings. More details concerning the complex of peer assessments can be found in the work of Staubitz et al. [147, 148, 149].

The remaining four approaches, video conferencing, just-in-time interventions, Request for Commentss, and tailored bonus exercises will be covered in larger detail throughout this thesis, as they contribute to a large part of our experiments. While explaining the conceptual thoughts behind the approaches, their implementation and also the caused effects later, we will still classify their usage and the resulting placement in the matrix for reasons of completeness here.

Video Conferencing is placed exactly in the middle. Tutoring done via video conferencing affects exercise difficulties as reasoned in compliance with Vygotsky's theory [169]. Having a live video session also offers a completely different format with fully individual content. The tutoring session is furthermore a didactical approach, which however can only be carried out relying on numerous technical services ensuring a lag-less connection and synchrony of content between participants.

Request for Commentss are a technical approach, aiming to offer individual help for struggling students. The concept builds on top of the existing exercises, but offers help in a different format, which leads to the decision to place it in the top-right quadrant. Whether or not the difficulty is lowered or adapted depends on the answers a student receives, therefore the placement still covers the difficulty area, while the aspect representing the different format conceptually prevails.

Just-in-time interventions are a technical approach introduced to further motivate students to reach out for help or interrupt them if they are stuck. Aiming to especially affect struggling students, this approach mostly tackles the difficulty area.

Last but not least, tailored bonus exercises are placed as a highly technical approach focusing on offering additional content. This additional training option does not lower the difficulty of existing content or come with any additional explanation. It is intended to pinpoint students' specific weaknesses by using gathered progress data and retrieves suitable exercises based on a technical recommendation algorithm.

Having classified all approaches, we will next explain how we aim to detect struggle and provide additional detail on factors influencing content difficulty, before describing the design decisions of the technically implemented approaches. The structure of the subsequent sections follows the actions already visualized within Figure 1.1, resembling the order of the iterative process we took to develop and evolve our approaches.

3.2 Understanding Struggle

In order to purposeful intervene on students, instructors or an autonomous system first needs to detect or infer that a student is currently struggling with the problem at hand. The most prominent cause for extended struggle is a lack of knowledge, which may either show as the absence of prior knowledge or the manifestation of misconceptions impeding students' progress. In the following subsections, we outline approaches to understand and detect struggle within programming MOOCs.

3.2.1 Assessing Prior Knowledge

In general, exercises being considered suitable for the advancement of a student should either teach a new concept or deepen the understanding of a previously covered one. As already shortly motivated, the suitability of an exercise for a student therefore depends on the specific (sub-)topics dealt with and on the individual perceived difficulty, composed of the:

- Difficulty of the actual steps to solve the exercise
- Prior knowledge of the student
- Expressiveness of the exercise description
- Offered templates and hints
- Impact of additionally offered help

The perceived difficulty is most tightly correlated with the students' knowledge. In order to propose suitable exercises, it is therefore of vital interest to assess prior knowledge of students. In the following, we present three different approaches to assess the prior knowledge, with a focus on knowledge in the field of programming and testing.

On the most abstract level, we follow three approaches to approximate the actual skill:

1. Ask the students directly how they would classify their prior knowledge.
2. Ask (multiple-choice) questions of differing difficulty to determine their knowledge on an abstract level.
3. Incorporate metrics of the ongoing course. As we focus on programming exercises, we can analyze events specific to programming, such as unit tests solved or the number of errors produced.

The first approach, simply asking, is the most trivial and might seem superficial at first. However, pedagogical and psychological research has shown, that it is reliable at least in in-class settings [13, 133].

The accuracy of the second approach highly depends on the questions asked. The result should allow to distinguish between students that have no knowledge and basic knowledge as well as advanced or even expert knowledge. Finding a minimal set of such questions is non-trivial itself, as course instructors need to guess the actual difficulty of their survey questions with regards to the expected audience. Albeit testing these questions within a dry-run with a group of colleagues, friends, and students in different stages, anticipating the distribution of students enrolling into a course is hard.

The third approach requires more effort in data acquisition and interpretation. Furthermore, the metrics need to be based on a sufficient number of exercises solved before being treated as reliable, as outliers are especially probable during first tries and would strongly distort conclusions due to the sparse data foundation. This postpones the availability of such evaluations on metric data to later course stages.

In general, all three approaches should lead to similar results or even support each other. The accuracy of the approaches however differs and is threatened by different factors. For example, students most likely do not voice that the content was too difficult when following the first approach - they will just leave the course. However, the resulting dropouts or stopouts[22] will be reflected in the metrics of the third approach.

With regard to the reliability of the approaches, we assume that approaches one and two will be rather similar and have to be treated with caution. Not because they do not work in general, but because the self-assessment might be skewed much stronger than in a normal class setting due to a lack of fellows to compare oneself with, and because the self-test questions might have been too easy or too hard for the audience. As approach three reflects part of the actual progress, it should be regarded as most reliable and in doubt be trusted in favor of the other approaches.

In order to judge the suitability of exercises already conducted in retrospective, we consider the following data to be of most interest: First, the unique accesses of students to an exercise will reflect whether a particular exercise being much too difficult caused students to leave the course. Second, the reached scores will show whether an exercise was too hard to be completed in general. Third, the required timespan to solve the exercise will give further detail on the difficulty of an exercise even if the majority of students successfully completed the exercise.

[22] Stopouts are temporary dropouts with students coming back after some time.

3.2.2 Tracking Progress

In general, MOOCs can have any structure, from almost no structure at all, to fully structured (x)MOOCs that are relatively closely resembling traditional frontal classes. The grade of structure often also depends on the topics dealt with. Areas that are suited for so-called “Rhizomatic Learning” as coined by Dave Cormier, an approach aiming at maximal individual flexibility and letting the audience decide the content orientation [28], often lie in the sociological or languages field, whereas technical courses tend to have more structure in general. As courses on openHPI deal with IT related topics, the platform is mostly designed for sequential progress through learning units being released weekly. For general computer science courses, like “In-Memory Data Management”, “Internet Security” or “Semantic Web”, learning progress is usually checked with optional self-tests that are intertwined with the video lectures. This leads students to alternate between consuming videos to rather passively acquire new content, and actively repeating or scrutinizing the presented facts via multiple-choice quizzes. For teaching teams and platform operators, this also allows to track learning progress and detect potential weaknesses in students’ knowledge.

Programming courses with practical exercises offer additional metrics. Given the technical nature of the exercises and the necessity to evaluate students’ products, textual source code, against a central grading tool, thus offers further proxies specific to programming, such as errors or comment ratios.

The essence of learning, a gain in understanding, the ability to put it into practice, and building on top of acquired knowledge can not be measured directly. Therefore, it has to be proxied with metrics such as scores or other numeric values derived from students’ progress and learning products.

3.2.3 Measuring Effects

When conducting experiments and applying interventions, the way of measuring the outcomes of the induced changes has a high impact towards the evaluation. Findings gathered from qualitative user studies and interviews might bring up different aspects than the results of a user survey, which in turn can bring up different impressions compared to those learnings drawn from learning analytics data. The ultimate goal for any research is to uncover correlations and achieve findings that are reproducible and prove to be robust concerning the underlying experiment setup, i.e., in our case the participant group resembling from our audience of MOOC students. Such generalizable results are likely to achieve a larger impact and provoke further experiments of fellow researchers. They are thus capable of providing a lasting positive effect to the field, in addition to advancing the status quo in the respective research area. For the experiments carried out in the context of this thesis, feedback and results were gathered through all three aforementioned approaches. Interviews on the basis of low-fidelity prototypes granted first insights into the respective research area and provided feedback with regard to the general applicability of the taken approach. This initial knowledge was then broadened and validated via qualitative user studies in controlled environments and subsequent tests in smaller courses via surveys. Given positive outcomes and the prospect of a generalizable learning, the experiment was repeated under more restrictive settings, following a rigid experiment design aiming to answer previously postulated hypotheses. The results presented in this thesis were mostly taken from this third phase of experimentation, additional information on previous tests can be found in our publications belonging to the respective approaches.

Interview feedback and even survey results are affected by a positive bias of the participants. Users of a prototype often show a tendency to actively find and mention positive aspects of the prototype, fueled by the urge to appreciate the effort and to help the creator positively finishing the research. This tendency will be further outlined in the evaluation chapter. Another bias which influences the data collection within MOOCs is the fact that surveys are usually conducted at the end of the course runtime. With weaker students having dropped out in earlier stages, stronger students have more impact on the results of course-end surveys, leading to potentially skewed data. For this reason, the highest reliability and quality is given for the factual data that is continuously collected by the underlying system. This factual data lies outside of students' direct influence. Therefore it is unaffected of their conscious as well as subconscious desire to help or comply and describes the actual effects in the most objective way. The only residing bias is that the share of data coming from students who perform above average will be overrepresented for systemic reasons. Students' that quit the course do no longer take exercises and consequently also do no longer contribute data. Having that issue in mind however allows to partly mitigate this problem, e.g., by restricting certain evaluations to exercises presented early in a course if necessary.

Measured Metrics

Reducing non-constructive struggle requires detecting struggle at first. Different metrics can be employed to gain information about the progress students are making and potential hardships they might endure. In general, metrics can be divided into two main categories: those that represent finite results, and those that contain additional information that further describes the process of leading to these results. Into the first category fall metrics such as course scores, results of unit tests, the program length and its comment ratio. This data are particularly useful to reconstruct the situation a student was in. While important information such as external factors affecting the situation and mood of a student (e.g. time limits due to an upcoming appointment, potentially adding stress) cannot be gathered, the present factors are nonetheless an incomplete, but representative collection of facts that frame the setting of the students' work on the exercises. The metrics of the second category are observations and measurements that allow for deeper insights into the students' progress. Information about how long a student needed in total, how many sessions were necessary, and what additional information was copied in from other websites helps to understand and retrace the journey a student took in order to achieve the results subsumed by the metrics of the first category.

The gathered metrics are presented in the following overview, along with additional details what information can be derived from them.

Course Scores

Achieved scores are the most obvious metric to measure and reflect learning success. Individual exercise scores indicate mastery of the respective topics dealt with, the overall course score (represented as the relative percentage of the maximum score reachable if suitable) allows to draw conclusions of overall content mastery. Overall scores also mirror students' diligence to some extent, as full score is only reachable if consistent attendance until course end is combined with the willingness to overcome individual hurdles and misunderstandings.

Unit Tests

Successfully passing individual unit tests shows whether the respective issue was solved, e.g., a variable was initialized and used in the correct program context. As the unit tests are manually designed and implemented by the instructors, the results from these unit tests therefore allow to gather fine-grained insights about individual students' problems with regard to the course topics.

Working Time (including Time of Day)

The amount of time a student spent working on a specific exercise reflects and comprises three other metrics, which cannot be measured directly: the diligence of students, encountered problems and struggles, as well as prior knowledge and overall mastery of a topic. If students already have prior knowledge about a topic, they will finish the exercise faster than the overall mean. Students encountering issues will need significantly longer. If students need longer but succeed with full score in the end, this reflects their diligence. Students quitting the exercise without full score usually gave up on this exercise, only a fraction of them was interrupted by real-life duties and just forgot to commence with the exercise later on. The localized time of day when students are studying allows to infer on the learning habits the students developed.

Sessions Lengths

Average session lengths allow to draw conclusions on students' learning styles, whether they prefer multiple shorter sessions (e.g. studying of a unit each day in their lunch break) or less frequent, but longer sessions (e.g. the entire weekly content on one day during the weekend).

Program Runs and Detection of "Bursts" (until high score is reached)

Specific to programming exercises, the absolute number of program runs reflects the number of tries necessary to reach a working solution. Within first runs, syntax issues may occur, whereas later runs usually are used to fine-tune the program output towards the expected results, both with regard to formatting as well as semantic correctness. Frequent requests of program runs within a short time frame (bursts), perhaps even without any changes on the source code, serve as an indicator for frustration, because this behavior usually results from students' disbelieve that the encountered error is rooted in their contribution.

Errors (until high score is reached)
The number of occurred errors, both runtime errors as well as compile-time errors, is another indicator for potential frustration. Compile-time errors usually stem from wrong syntax and are comparably easy to fix for advanced programmers. Considering the potentially cryptic compiler output, this however is still challenging for beginners. Runtime errors are often harder to fix, e.g., null pointer exceptions or array index out of bounds exceptions in Java require more steps in order to pinpoint the origin of the error. In general, all errors hinder students' progress and usually mark a larger threat than just an increase in program runs to, e.g., format the program output.

Gaps
Extended gaps in the development process show that a student was not able to complete an exercise in one go. Urgent tasks or distractions like an incoming telephone call may of course also cause gaps not related to the exercise at hand, however, these effects are normally distributed and therefore do not impair the suitability of the metric for indicating struggles of individual students or issues within specific exercises.

Copy and Paste Events
Copying identifiers or code fragments is a common behavior, also for experienced programmers. Copy and paste occurrences however also can arise from a lack of information, for example, if complete structures like a loop header, or a sequence of statements, e.g., to swap variable contents or find the smallest element of an array, are copied. Within an exercise, having students pasting content that was not copied from the exercise page itself, therefore indicates missing information and might hint towards experienced struggle.

Characters per Time
The typing speed of students can yield information about their skill level. Higher speeds, especially for sequences typically used in programming such as keywords or loop headers, express higher familiarity or even usage of the so-called muscle memory[23].

Program Metrics: Program Runtime, Length, and Comment Ratio
The program runtime reflects the efficiency of created programs. Despite being of interest in general, given the introductory programming exercises we deal with, this measure only adds limited value for our research. The length of a program can hint towards more experience if a student submits a working solution that is shorter than others. However, programs being too short often lack expressivity, leaving this metric to be treated with caution. The comment ratio, excluding given comments from the supplied template usually either reflects a better structure or particularly hard program parts. In general, a higher comment ratio indicates a better designed and documented program, resulting in better program quality.

[23] Effect that repeated movements form procedural memory, allowing, e.g., sequences to be typed at greater speed with reduced conscious effort.

Derived Metrics: Understandability, Error Proneness
A measure for understandability can be constructed as a combination of the program length being within an expected range (neither too short, hinting at difficult code, nor too long, hinting at spaghetti code) and a decent comment ratio for the amount of code. Error proneness can, for example, be approximated by averaging the errors per successful run, potentially improving the metric by excluding consecutive correct runs showing only minor source code changes. Derived metrics have the downside that a change in one of the influencing factors might cause an extreme effect on the resulting value and that the added layer when combining factors increases complexity, thus potentially lowering comprehensibility and expressiveness.

External Metrics: Quiz Scores, Forum Usage
Further metrics originating from the course platform, such as course progress, quiz scores, the forum usage, or video speed settings can be employed to gain further insight about students' success and behavior. However, the larger the conceptual distance of the metric to the task we want to focus on, the less expressive the metrics are and the more side effects might distort the numbers. Of the many metrics potentially available on the platform side, we argue that the most suitable ones for struggle detection are the initial quiz scores of self-tests and the forum usage. Eventually, we abstained from using these data, as the correlation of the self-test scores and experienced struggle was lower than expected, and the mapping of forum posts asking for help to the respective exercises was too error-prone.

3.2.4 Identifying Struggling Students

To help struggling students we first need a method for identifying them. The most obvious metrics for that are the number of program executions and the average working times. As the working times correlate strongly (Pearson coefficient of 0.9) with the amount of code executions and probably better reflect frustration levels, we decided on using working times. We defined the 75th percentile of students to be regarded as considerable slower than their peers, therefore being potentially struggling or stuck and issued interventions on them. This decision was backed by an analysis of the working times of already existing exercises from the Java course offered in 2015 on openHPI.

In Figure 3.2, we see students' working times for an exercise of average difficulty from the Java course run in 2018 with outliers removed. The distribution is skewed to the left, meaning that most students finish the exercises quicker than the average duration (dotted vertical line), most likely without encountering any problems. The 75th percentile (solid vertical line) of working time was chosen as a cutoff to keep a relatively large treatment group in order to gather data, with the accepted risk of an increased ratio of false positives, interrupting students who are not yet struggling. Other potential cutoff metrics which we considered but discarded were fixed times, averages, and lower percentiles (like the median). Fixed times do not qualify as a suitable cutoff metric, as the exercises are of different difficulty and complexity, thus the time needed to solve them varies and can hardly be predicted beforehand. Taking the average as a metric would classify too many students as struggling students and further has the problem that outliers have a too strong impact and need to be removed. The median (50th percentile) is not suited, because we only want to indicate students who are significantly slower in solving the exercises. Other metrics, such as taking a closer look on the code structure or detecting patterns on submissions, for example bursts on trying to run erroneous code as an indicator for frustration, propose worthwhile future work. This limited approach, though probably not yet reaching the full potential, however, allowed us to practically address a series of problems described in the next sections.

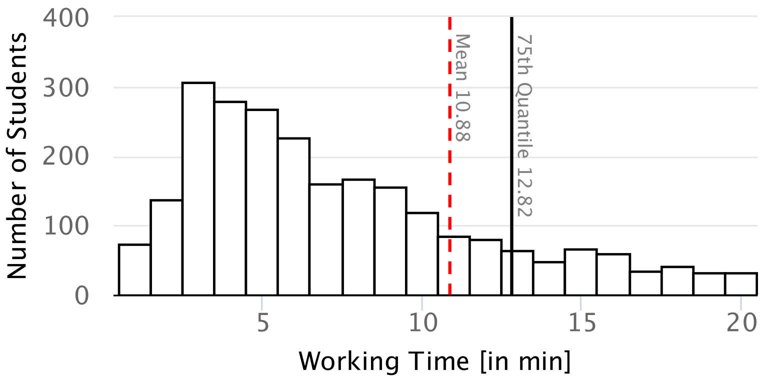


Figure 3.2: Working time distributions of students for a typical exercise.

3.3 Intervening on Students

In the following subsections, our concepts of video tutoring, code commenting, situationally intervening on students with popup windows, providing tailored bonus exercises, and automatically detecting weak material will be described. All approaches have been tested and evaluated within several MOOCs. They are presented here in similar detail, though the experiments on the concepts of tailored bonus exercises and code commenting, combined with situational interventions, will contribute the largest share of research outcomes and therefore dominate the evaluation section later on.

3.3.1 Video Tutoring

Motivation

In Section 1 we already brought up the issue of social distance between students and course instructors. This distance can be bridged by different means: course announcements for information from the instructors, surveys as a return channel for student feedback, and forum communication for bidirectional exchange. These means, however, have several shortcomings: they are textual and they are asynchronous. Textual feedback only covers the visual sense by reading and makes it difficult to express conceptual misunderstandings or provide intuitive explanations. The asynchronous nature furthermore makes it more difficult to engage in a vivid conversation, as further inquiry and back and forth discussions are often stretched over a large timespan. While studies indicate that having only textual learning material does not lower learning outcomes significantly compared to mixed material (including images, audio, or video) [106], student engagement is affected by the presentation medium and style [56]. Production value has only little impact on video watching and learning engagement. In contrast, the actual content and the appropriate means of delivery are decisive [56].

Offering other approaches for delivery of content and to establish additional connections between students therefore might yield yet untapped benefits.

Given the significant role collaboration plays in practical computer science education on campus, it becomes evident that nowadays online course platforms mostly lack the necessary collaborative capabilities. In the following, we propose and conceptualize a solution to support collaborative programming through video conferencing for practical exercises employed in MOOC contexts. Substantial parts of the motivation for this prototype as well as our results and learnings from the experiments can also be found in our publication "Video Conferencing as a Peephole to MOOC Participants" [163].

Upon designing an actual implementation, we evaluated the potential of such a solution, as well as doubts and objections students voiced. With a survey placed within a MOOC in 2014, we learned that the survey participants valued the possibilities that video conferencing can provide. Existing and remaining concerns were mostly centered around privacy issues. Based on these learnings, we designed an initial prototype and set the surrounding conditions as well as boundaries for the first tests.

Surrounding Conditions for Collaborative Work in MOOCs

Running a MOOC is an intensive and busy time for the involved, often rather small teaching team. Supervising and nurturing discussions, fixing glitches in the course material, and keeping all technical dependencies running is enough to fill each workday during the runtime of a course. Particularly courses with experimental or interactive parts require additional efforts to fix and enhance the tooling used. Therefore, the majority of content is produced before the course runtime. During the course runtime, course conductors mostly moderate the forums, record additional "office hours" videos and supervise the help desk to interact with the students. The comparison of campus centered teaching activities and distance education shows several differences concerning the engagement and communication behavior.

While the core principles of teaching remain the same, the surrounding conditions in a MOOC are different. Pea describes that collaborative efforts and the sharing of different perspectives are required to acquire knowledge [117]. This has been missing until the recent trend to integrate collaborative concepts into MOOCs [85, 150, 151]. Group-based experiences are supposed to improve "satisfaction, persistence and intellectual and social development" [23]. They are therefore relevant not only to on-campus courses, but to students taking part in online courses as well. Chen et al. measured in 2008 that, compared to on-campus students, remote participants taking the classes online were at least as engaged when it came to asking questions or contributing to the class discussion. Yet they were significantly less involved in working with other students to prepare class assignments or projects [23]. Since then, multiple approaches to improve collaboration among MOOC students, such as openHPI's Collab Spaces [150] or Stanford's Talkabout [85], have emerged.

To improve engagement, it is beneficial to focus offered support on actually occurred problems and recent issues. When using specific issues, engagement will be high for individual students who faced that issue, but given the range of possible problems, the number of reached students remains comparably low. With Talkabout or Collab Spaces, general technical means offering possibilities to discuss with interested subgroups are given. However, following this direction shifts the effort of instructors from lecturing into the direction of so-called apprenticeship, having one higher skilled "master" of a topic guiding a small group of lower-skilled "apprentices". Even though such apprenticeship approaches are beneficial to the students' learning outcome, it is not feasible for instructors to mentor and support the thousands of students who participate in online courses individually [107, 167, 169]. An option to further scale the impact might be moving the discussion back to general, public support forums. Experiments in former courses showed, that even if dedicated support forums exist, those are often too impersonal to cause an effect. First, forum usage suffers from the aforementioned drawbacks, and second, given the specific use case of spontaneous help-seeking while programming, the act of asking a question sharply interrupts the students' workflow. After one has left the program editor to post on the forum, one has to check regularly whether an answer was posted, thus further disrupting the learning efforts. In summary, forum-based solutions and solutions that require instructors to interact with many groups individually, will not provide an impact due to missing acceptance or scalability.

Still, the benefits of collaboratively working on actual problems in pairs or small teams should be employed for learning. Especially when working on practical tasks requiring creative or otherwise mental demanding solutions, teamwork has proven to be beneficial. In our case of programming, writing code collaboratively has been promoted in the form of pair programming to help programmers share learnings and improve their code's quality [105, 175].

For this reason, we further pursue the approach of video conferencing with an implementation specifically designed for our use case. A deep integration into students' workflow is aimed to mitigate acceptance and usability issues. Further options to match student pairs on open questions, will help to mitigate the scalability issues when given a widespread acceptance of the feature.

Design Decisions for Video Tutoring in Programming MOOCs

We propose a video conferencing solution integrated into our web-based execution environment to specifically support remote tutoring sessions. Leveraging students' knowledge by enabling them to mentor their peers and by encouraging them to share their recorded discussions also fosters the scalability of MOOCs.

With regard to the conditions given in programming MOOCs, our system should meet the following requirements:

1. **Pair programming support:** Lag-less synchronization of the source code and program output is crucial to enable a natural development flow.
2. **Apprentice becomes master:** By encouraging students to help each other, we offer advanced students additional options to grow their knowledge. This further reduces the workload on the teaching team and tutors.
3. **Reproducibility and rehearsability:** The ability to re-watch tutoring videos and use them as additional content will foster the effectiveness of tutoring for wider audiences.
4. **No additional plugins, no additional accounts:** Installing software or registering for 3rd party services will discourage usage and thus hinder adoption.
5. **Pairing of Participants:** Students asking for help and tutors should be automatically matched for the best potential outcome.

Pair programming allows us to put the student who is writing code, called the driver, to be set up in the previously introduced "zone of proximal development" [169]. In pair programming, the driver and the observer, the person tasked with guiding, reflecting and commenting on the code, usually switch roles after a certain amount of time. For our main use case tutoring, we will work with fixed roles: the tutor being the observer and the student being the driver. Having a tutor guiding and helping on problems, allows students to progress from "tasks they can do alone" to "tasks they can solve with external help". There will remain tasks that are still too hard, but usually exercises that are demanding and require the participant to leave his comfort zone yield the best learning results [15, 91]. Also the eXtreme Apprenticeship (XA) model suggests using scaffolding and mentors to help students. In the context of children's education, scaffolding has been described as "the way the adult guides the child's learning via focused questions and positive interactions" [8]. For this reason, we decided to limit the code synchronization to one site, forcing the tutor to explain all necessary source code changes instead of directly implementing them. Another benefit of this "guided programming", where students are lead by the tutor but need to solve all tasks themselves, is that it effectively circumvents students falling into pitfalls that would leave beginners stuck. Especially debugging sometimes feels cumbersome and demotivates students [175]. Experienced tutors can explain rather cryptic compiler errors and stack traces and help during debugging, allowing students to focus on their program design and algorithms. By limiting our approach to pairs of one participant and one tutor, we implicitly prevent the "free ride" problem mentioned by McKinsey [105], describing situations in MOOCs where students just copy existing solutions instead of developing a solution individually.

The second requirement, apprentices becoming masters, provides for the specific MOOC setting. While tutoring from teaching assistant to student works on university course scale, the proportions in MOOCs require a much larger group of tutors, preferably available over all time zones. Recruiting motivated and skilled students to take over the role of tutors is therefore necessary. In addition, students might feel more comfortable with receiving help from a fellow student, instead of the teaching staff [47]. Naumann et al. found that students are willing to contribute to forums and often solve issues without the intervention of teaching staff, e.g., by sharing pre-existing knowledge or providing links to external information [110]. The findings of Staubitz et al. [151] also support this conclusion. The fact that the ability to teach fellow students can be built up is further demonstrated by Coursera's[24] usage of Community Teaching Assistants – successful participants of former courses who volunteer to teach future classes [116]. In order to gain scalability, we propose to build up a potential pool of tutors through the tutoring itself. The main reason not to give everyone the option to answer open requests is that a synchronous audio and video connection implies potential threats with regards to privacy. As the tool supplier, we therefore want to ensure control over at least one side of the tutoring sessions. However, given the workload on the teaching team, we also admit that additional persons are required, as well. Starting from the group of course conductors and platform owners, we aim to unlock the tutoring backend for participants that we had a positive session with and who seem to be qualified with regards to knowledge as well as attitude. We are further confident that tutors do not need to be on expert level content-wise, since oftentimes it is already enough to just give a subtle hint, if necessary at all. Next to sorting out potential legal issues and carefully selecting potential participants that act as additional teaching assistants, the implementation has to offer an environment in which students feel confident in asking as well as answering questions. Good reproducibility and rehearsability allows also students to profit whose skill levels are above the average and thus simply do not encounter further issues, as well as students who lack the technical requirements or the extroversion to ask questions publicly. Synchronizing a recording of the video conference with the editor content of the student allows other students to track applied code changes and the discussion that led to these changes. Also adding a further channel to convey information improves the media richness. Apart from the primary use case within programming assignments, the software can additionally offer general availability of video conferencing to be used within arbitrary group tasks.

[24] see `https://www.coursera.org`

3.3.2 Code Commenting: Request for Comments

Motivation and Surrounding Conditions

We already reasoned that the strict syntax and structure of program code bears an increased potential for demotivation, especially affecting beginners. In the first programming MOOCs on openHPI, students were encouraged to paste their source code into the course forum in order to receive help from instructors and fellow students. The advice was accepted by some participants, while realistically the large majority of the audience was not reached due to several reasons:

1. Limited forum usage in general
2. Psychological barrier to post wrong code / admit lack of knowledge
3. Manual labor to transfer the content between platforms
4. Formatting issues
5. Loss of context

A substantial fraction of the audience does not use the forum at all. Just regarding the group of students interacting with the forum, the share of students actually posting content is only about one-third of the persons who are reading in the forum. Posting incorrect code additionally requires to overcome the psychological hurdle to admit the own lack of knowledge and also share the current state of work in its broken form. Given that most participants are in the age between 30 and 50, they already have a position in company hierarchies and might abstain from posting or at least hesitate to do it, as the post will stay available potentially forever. Additionally, other hindrances such as the manual task of copying and pasting the source code between the exercise platform and the course platform, formatting it in the forum, and adding a suitable introduction to the problem and an appropriate closing text to their post, further reduce the acceptance of the forum for this use case. On top of all that, exercise submissions consisting of two or more classes will lose their context if only one class (or only the potentially faulty function or statements) is copied over. Also the exercise description, results of unit tests, or program output, was almost never supplied alongside the pasted source code, leading to additional inquiries.

When other students replied with their advice or potential solutions, even more shortcomings became visible. Referencing a certain line number is only possible through manual counting, inserting code between two statements or within a certain line requires to copy the entire line, and discussions concerning different parts of the code will be intertwined in the forum depending on the posting order.

In summary, the naive, quick and dirty solution of using the forum turned out to be tedious for students as well as instructors and achieved only limited effects.

Design Decisions for Request for Comments

In order to improve on that situation, we decided to conceptually integrate this functionality directly into the code execution platform. This allows to tightly integrate all necessary and available information without further redundancy, preserving the context, and lowering the mentioned psychological barriers.

The following design decisions were made:

Commenting on Line Basis
Students can comment on each line separately, thereby implicitly preserving context on the surrounding constructs and forming suitable threads with respect to the issues dealt with.

Preservation of Context
Each Request for Comment is linked to its respective exercise, allowing to show the exercise description and also all contributing files, that might carry helpful information, such as read-only files, templates, or interfaces to be programmed against.

Enhancement with Program Output and Test Results
Upon requesting comments, the platform will issue a normal program run producing the standard output, as well as a scoring run resulting in the test results of the unit tests. This additional information gives advanced programmers helpful information to pinpoint existing issues and better support the student asking for help.

The approach was named *Requests for Comments (RFCs)* and implemented in our program execution platform CodeOcean.

Initial tests were promising and showed the potential of a well-crafted solution. Despite the limited technical complexity to implement the necessary program logic, user permissions and frontend views, feedback from students' was overwhelmingly positive and encouraging. Usage showed that the challenges of this approach did not lie in the technical realization, but in the appropriate and seamless integration of the feature into the respective workflows. The first prototype allowed students to ask for help by pressing a button above the source code editor. Before composing the request, the system additionally asked the students to optionally provide a specific question they have. Acceptance was given, resulting in a continuous influx of requests, initially not met with enough students answering to the raised questions.

The process to comment on others' requests initially required help offering students to go to an overview page of all open questions and pick the ones they preferred. The ordering of presented questions was based on the order of submission, placing the most recent questions first. The quality of asked questions differed, with the most prominent issues being "empty" requests without any question and no changes compared to the exercise template, potentially hoping for a solution. Another issue was that some students spammed requests to improve the likelihood of receiving an answer. Two quick fixes for these most common issues of misusing the basic prototype thus were introduced. The first one was setting a minimum time limit before allowing requests to encourage students to try on their own before longing for help. The second one was the introduction of a total limit of three open requests per student.

Whenever someone answers to an open request, the student who created the request and asked the question receives an email with the supplied comment as well as a link to the respective exercise allowing to commence working on it.

Being an improvement over the previous status quo, the prototype simplified, encouraged, and structured the help-seeking behavior of students, but only marginally improved the situation for those writing answers. Instructors and some very motivated individuals answered all reasonable requests on a daily basis. Only sustainable for the runtime of the course, and additionally binding time of instructors on a regular basis, it became clear that given the foundational acceptance of the feature, we needed to provide a better integration of students who want to potentially help their fellow peers. Liyanagunawardena et al. showed that a community is able to mentor itself, therefore it was the obvious next step to better use this potential [96].

In order to achieve a solution that provides organic scaling with the community, Request for Comments were thus integrated into the main workflow of students commencing through programming courses. Whenever students finish an exercise with full score, we know that they have passed all unit tests. While this does not necessarily mean that they fully understood all potential pitfalls, it ensures that they should, in general, know how to solve the respective exercise. Therefore, they should also be able to help a struggling fellow having a specific question concerning this exercise. So in order to let our approach scale, we forward a share of the students who have solved an exercise to an open question that belongs to the very same exercise they just solved. They are presented with a message that their score was successfully transmitted and congratulates them once more on their progress. Afterwards, the message politely asks them to help out a fellow student that is struggling with the very same exercise they just solved and that their input is appreciated.

In order to keep the motivation of answering students high, when selecting the request to forward to, requests with a student supplied question are preferred over those without a question. To further achieve a certain fairness amongst open requests and better load balancing in general, we also pick open requests not yet having received any or only a few comments over those which already received three or more comments. We decided against presenting students a choice of open questions to pick from. This was done in order to reduce unnecessary cognitive load on the one hand and to potentially have more fine-grained control where to send students on the other hand. Of course, answering a student's request remains fully optional for any one of the involved students. However, we abstained from expressing this too offensively in order to achieve a higher commenting ratio. The number of times a request for comment is featured, meaning it is being displayed to another student, is counted in order to be able to balance out the visibility of requests.

With the aim of enabling a content-centric discussion, all commenting students can see all other comments being written. Furthermore, they can indicate that they wish to receive a notification if the author of the request asks a follow-up question on their comment.

If all issues are solved and all misconceptions have been corrected, hopefully, the help requesting student finally solves the exercise with full score. In this case, the system removes the request from the pool of open questions with the goal to focus available capacities on other submissions. Additionally, after successful completion, the students are forwarded to their own requests, encouraging them to manually mark them as solved and allowing them to send a thank you note to the peers that contributed to the request. The manual marking of the request as solved yields information that the comments indeed helped in solving the issue. The option to say thank you was added only later on and demanded by students. From our perspective, with a working solution, the issue was solved. Much to our delight, our students however highlighted that learning, especially in this case, indeed is a social and collaborative effort and thus wanted to express their gratitude. Feedback especially on this tiny addition on the feature was very positive and led to several happy students voicing their joy over the friendly community in the forum.

Now having improved on the advise “giving” side, we noticed that the requesting side still showed a tendency to be caused by a rather small user base, indicating that students in need either have not noticed the functionality or were faced with remaining doubts to use the feature. In order to tackle this issue, we introduced so-called just-in-time interventions.

3.3.3 Just-in-Time Interventions

Motivation for Situational Interventions

With just-in-time interventions we aim to improve learning success, improve satisfaction and lower overstrain induced dropout of students of online programming courses. Just-in-time interventions in problem-solving motivates students to ask for help and feedback when they face an extended struggling period with an exercise.

To prevent dropouts and increase student satisfaction, we want to help struggling students while they are working on their exercises. As shown in previous research [159], students often struggled before they dropped out. From forum comments and other feedback we know that spending too much time is a cause for students’ dissatisfaction with a course. Therefore, helping students at the moment when they are stuck is beneficial, allowing to solve their problems before they lose interest and drop out.

We issue just-in-time interventions to students when we think that they are struggling or even stuck with an exercise. The purpose of these interventions is to interrupt the students and motivate them to rethink their approach, review video lectures, ask for help or do other exercises first.

Design Decisions for Intervening on Struggling Students

Interruptions can cause eye-opening moments by inducing a metaphorical step back, helping to overcome mental barriers caused by focusing too much on a certain detail. At the same time, interrupting students in their workflow, even when enacted with positive intentions, bears the risk of impairing the learning experience. They can also cause annoyance when interrupting a productive line of thought, or even lead to anger if they are too coercing. The actually caused effect does not just depend on the timing, but also on the tone, vehemence, and the frequency of the interruptions. Accordingly, when designing an efficient and helpful intervention mechanism, the following problems should be addressed:

1. **False Positives**: If exercises are short, we might bother students even though they are not struggling.

2. **Cold Start Problem**: If only few students have worked on the exercise, data is still unreliable to determine intervention timings.

3. **Different Focus**: If students review lecture videos while they work on an exercise, they should not receive interventions.

4. **Commencing an Exercise**: If a student closes the exercise and comes back at a later point in time, when do we issue the intervention?

5. **Annoyance**: How to keep the annoyance of interventions low and still cause an effect and gather data?

To solve problems 1 and 2, we set a minimum time limit of 10 minutes. If the 75th percentile lies below 10 minutes, we trigger the intervention at the 10-minute mark, granting students enough time to solve their problems on their own first.

With regards to problem 3, we just consider and count working time while students are actively working on the exercise to prevent unnecessary or away from keyboard interventions. We stop the timer if the exercise view in the web browser lost focus and continue if the student comes back to the exercise.

When students close the exercise and come back later, they need some time to understand the task and their previous code again. For returning students, we therefore set the intervention interval to be the maximum of the remaining timespan to the 75th percentile and 10 minutes.

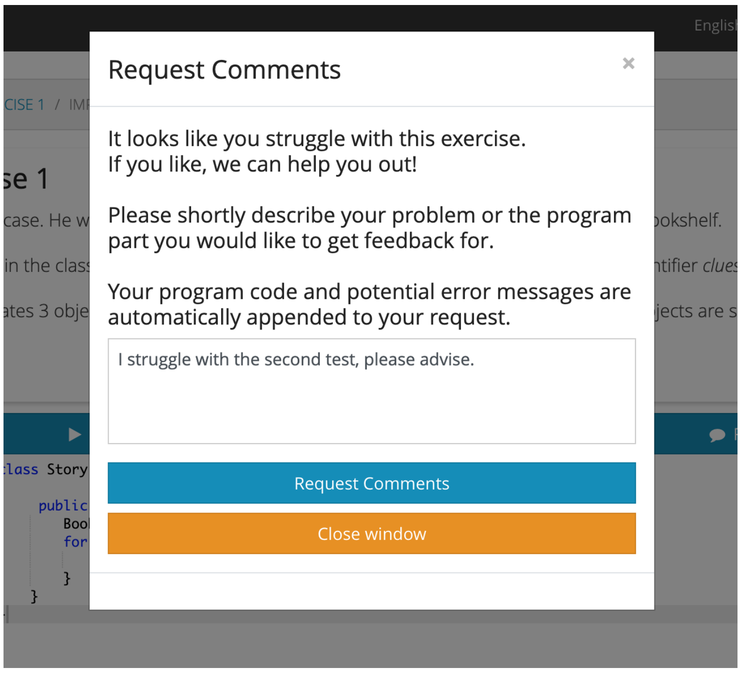
As we are aware that the interventions can annoy students, especially if they are not stuck or just like to fiddle with their code (problem 5), we set a daily limit of interventions to three interventions per day and an additional limit of two interventions per exercise for each student. This ensures that we intervene often enough to gather insightful data on the one hand, and do not annoy the students too much on the other hand. Whether or not a student was actually struggling cannot be detected or derived automatically and thus has to be examined within a survey.

Aiming to analyze the effects of the issued interventions later on, we further decided to create two different types of just-in-time interventions. Both types will be presented via popups and follow the same guidelines concerning the frequency and point in time of being issued. The essential difference between the interventions is the students' reaction we are aiming to induce. In the course of our research, we employed *Request for Comment Interventions* and *Break Interventions*, shortly explained in the following paragraphs.

Request for Comment Interventions

Requests for Comments (RFCs) are a useful feature for students to get help from other students who already solved the exercises.

Unfortunately, we have seen little use of this feature in our initial experiments when introduced in previous courses[25]. Although the feature was deemed helpful by the students using it, they seemed to need a little push to actually reach out for help, reinforcing the findings of Aleven and Koedinger [2]. We achieve this additional nudge by directly showing them the integrated Request for Comments dialog (see Figure 3.3).


Request Comments

It looks like you struggle with this exercise.
If you like, we can help you out!

Please shortly describe your problem or the program part you would like to get feedback for.

Your program code and potential error messages are automatically appended to your request.

I struggle with the second test, please advise.

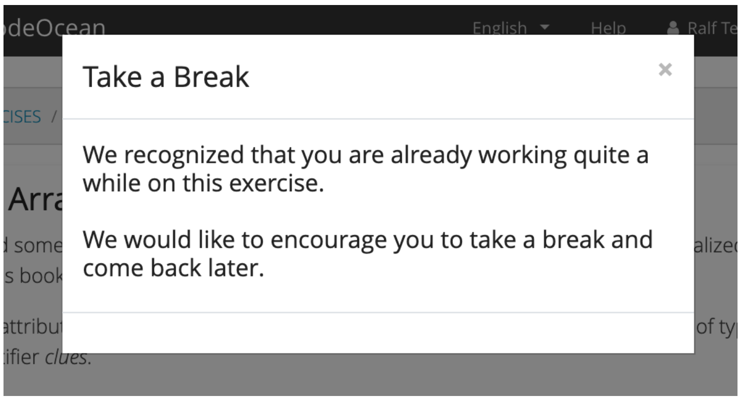
Figure 3.3: Screenshot of the Request for Comment intervention dialog.

Instead of just motivating students to press the button to issue a Request for Comment, we decided to directly integrate the request dialog into the popup in order to cut down on unnecessary clicks. While being encouraged by us, writing an RFC stays optional: the dialog is closable and might be reopened later, allowing students to complete the present thought first.

[25] Feature was available in the courses *javaeinstieg2015* and *javawork2016*.

Break Interventions

Break interventions encourage students to take a break and come back to the exercise with new ideas. The break interventions are also issued when we assume that the student is struggling with the exercise. Similarly, we show them a closable dialog, but in this case only presenting a text to remind them to do a break (see Figure 3.4). Doing a break and giving the brain some rest can be beneficial to overcome side effects of concentrated working: fatigue and distraction, which results in errors and eventual frustration. Neuroscience researchers recommend taking a short break after periods of concentrated work [6]. Besides taking a break, we assume that struggling students also regard the break intervention as a motivation to review the course material.



Take a Break

We recognized that you are already working quite a while on this exercise.

We would like to encourage you to take a break and come back later.

Figure 3.4: Screenshot of the break intervention dialog.

Our interventions furthermore affect the state of being stuck, which is another threat to learning. Being stuck is actively disrupting potential flow and causing frustration. The so-called “flow” state, as defined in psychology and coined initially by Csikszentmihalyi [29], describes a state in which someone is fully immersed in an activity, receiving positive feedback and emotions from the process of pursuing this activity. Flow correlates with one’s performance in learning [31], while it is still debated whether so-called “overlearning”, meaning reaching knowledge after initial mastery, is necessary to achieve a flow state [30]. Nakamura and Csikszentmihalyi further state that “the optimal level of challenge stretches existing skills” and thus helps to maintain the flow state [142], which is also in compliance with Vygotsky’s aforementioned zone theory [169].

Both just-in-time interventions have the positive potential to disrupt unproductive struggle when being stuck. The break interventions directly aim at this effect, while the RFC interventions potentially cause this effect as a byproduct. We consider the probability to cause a negative outcome by disrupting a flow state to be low. If a student is learning successfully or even exceptionally well, as likely if having reached a flow state, we expect a comparably fast completion of the presented exercises and thus will not issue any intervention. From the conceptional design, our approach of just-in-time interventions should thus support positive outcomes without deteriorating learners’ experience.

3.3.4 Tailored Bonus Exercises

Motivation and Surrounding Conditions

As a second type of interventions, we propose tailored bonus exercises. They offer each student additional training exercises which are suited to tackle the individual weaknesses of the student.

From previous courses we learned that many students want additional exercises for practicing. As some of our students told us that they had to leave the course because of time constraints, we cannot simply increase the amount of exercises for all students. Therefore we decided to add optional exercises to the course. Since we know how well a student performs in the course, we are able to recommend bonus exercises tailored to the strengths and weaknesses of the individual student accessing them.

Adding bonus exercises as optional material has the benefit that students who want to solve more exercises get more training and those who already spend enough time with the course are not penalized in terms of grading.

Learning material and exercises in MOOCs are often incrementally designed, which means that materials of ongoing weeks assume that students understood everything from the previous weeks. For a given exercise, the core problem however also might not be the underlying concept itself, but the specific exercise description causing issues. Thus, a differently designed exercise or repetition of the content might help them to understand the concept better.

Design Decisions for Tailored Bonus Exercises

Based on the aforementioned conditions, we posed a set of requirements for our bonus exercises, of which the most important are presented in the following. The bonus exercises should be:

optional, meaning that students are not obligated to solve them and do not miss any graded points if they skip them.

solvable for the student, meaning they should only cover topics already encountered.

tailored, meaning they should deal with the concept the student had the greatest problem with.

focused on the current course progress, so each course week will get its own pool of bonus exercises.

non-repetitive, meaning that contrary to Michlík et al. [108], we do not want to present an exercise to a student that was already encountered another time, as we believe that the learning effect is low because students would either skip the exercise or copy the solution from the previous attempt.

uncapped, meaning that a student can request as many exercises as desired.

Typically, recommender systems present a list of most relevant items, i.e., exercises in our case. We provide bonus exercises on a weekly basis. This means that each week has its own pool of potential bonus exercises in order to not confuse students with older material of previous weeks and to target specific deficits of the current week. To find the most relevant bonus exercise, a content-based recommender system is used in favor of a collaboration based approach (see Section 2.1.5). Furthermore, our approach follows the Item Response Theory [90], as we base our recommendations on individual student-topic gradings instead of their achieved total test scores.

As a prerequisite, we need to identify how well a student understood a topic. For this, we manually annotated all exercises with the topics they cover and a difficulty level. Having information about the student and the individual exercise submissions, we create a vector-based student knowledge profile, which builds the basis of the employed content-based filtering. In the following, we will shortly outline the domain model comprising all additional information necessary for our recommendations. On the basis of this model, we then explain the student model representing individual students' knowledge as well as the employed recommendation algorithm in greater detail.

Domain Model

As a first step for our content-based approach, a domain model has to be defined. It is important to note, that domain-models have to be created for each course independently. For example, Java's class hierarchy is different from the one used in Python. To create the domain model, course instructors have to provide the topics students will learn in the course and annotate them to the exercises with which these topics are practiced. In Figure 3.5 we see that each exercise has one or more topics associated. We recommend not to use too many topics per exercise, i.e., less than three and to focus on the most important topics. Weights are used to describe the importance of the topic in the exercise. Exercises are also annotated with their difficulty level. Furthermore, we suggest to keep the topics as broad as possible and on the same abstraction level. Michlík et al. [108] recommend to connect topics among themselves with directed relationships. To keep the additional effort for the teaching team small and not to integrate too many variables into our first experiment, we purposely chose to leave this task for future research.

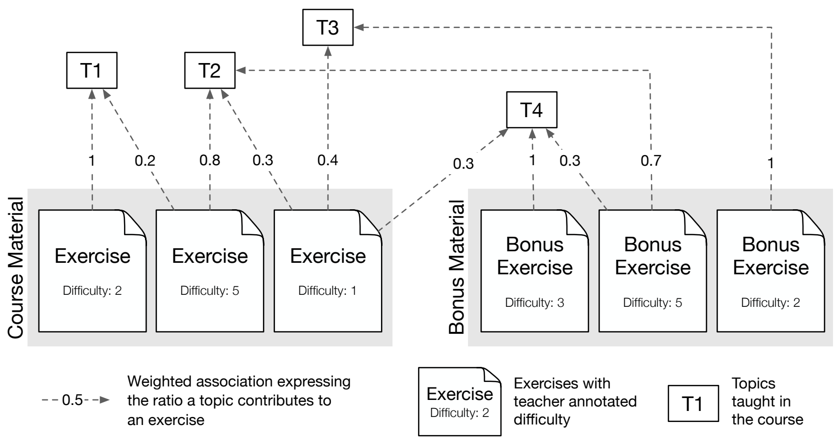


Figure 3.5: Example of our domain model for programming exercises.

Student Knowledge Model

As a basis for recommendations, we create a profile in the form of a vector-space model for each student that reflects how well a student understood the topics taught in the course with a value between 0 (not understood) to 1 (fully understood).

Similar to the model used in ALEF (Adaptive LEarning Framework) [108, 168], we are dealing with programming exercises. However, we do not rely on self-evaluation, but base our data on automated grading and testing. We assume that understanding of a concept and being able to apply it in exercises are correlated. Therefore the understanding is directly correlated to the students' exercise performance.

Concerning the classification described in Section 2.1.5, our approach is deemed as a content-based approach, as we recommend exercises based on information about previous exercises and students' achieved performance, opposed to the students' likeliness with other students.

Our knowledge model should reflect the following criteria:

1. Students who required more time to solve exercises than their peers did not understand the concepts behind the exercise as well as their peers.
2. Not reaching full points means that a student had problems with an exercise. However, students who got full points but needed a very long time to solve the exercise were able to show their ability to use the concept in the end. Therefore, the knowledge level of topics solved 100% correctly should always be higher compared to fast but incompletely solved topics.
3. Difficult exercises need a deeper understanding of the topics. Consequently, difficult exercises have a higher potential learning effect and should be weighted stronger in the model.
4. The resulting exercise proposals should be comprehensible.

We developed a knowledge score function (see Equation 3.1 and Table 3.1) reflecting those criteria. We calculate the knowledge score $\Theta(s,t)$ for each student s and each topic t of the course.

$$\mathit{\Theta}(s,t) = \frac{\sum_{e \in \mathcal{E}_s} \sigma(s,e) \cdot \delta(e) \cdot \rho(t,e) \cdot \varphi(e,\mathcal{E}_s)}{\sum_{e \in \mathcal{E}_s} \delta(e) \cdot \rho(t,e) \cdot \varphi(e,\mathcal{E}_s)} \tag{3.1}$$

S	Number of students	$\mathcal{E}_s \subseteq \mathcal{E}$	User accessed exercise $e \in \mathcal{E}_s$
T	Number of topics	$\iota(e,\mathcal{E}_s)$	Returns position of e in $\mathcal{E}_s$
E	Number of exercises	$\rho(t,e)$	Ratio of topic t in exercise e
$s \in \mathcal{S}$	Student s in $\mathcal{S} = \{1,\ldots,S\}$	$\sigma(s,e)$	Scoring of e for student s
$t \in \mathcal{T}$	Topic t in $\mathcal{T} = \{1,\ldots,T\}$	$\delta(e)$	Difficulty level of exercise e
$e \in \mathcal{E}$	Exercise e in $\mathcal{E} = \{1,\ldots,E\}$	$\varphi(e,\mathcal{E}_s)$	Diminishing function
$\Theta(s,t)$	Knowledge score function		

Table 3.1: Overview of variables in our knowledge model.

The formula consists of the following parts:

Scoring Function

The scoring function $\sigma(s, e)$ (see Table 3.2) calculates how well we think the student s solved the exercise $e \in \mathcal{E}_s$, based on the score and the working time of the student. We take the test score the student reached in the exercise (row) and compare the student's working time to the working time of his peers (column). The achieved test scores and working times are rounded down to the next specified value. For example, if a student reaches 95% of the possible points, we round down to 80%. If a student reaches full test score (100% of points), the resulting score will never be below 0.7, regardless of the required working time. We give a lower score (0.6 or less) if a student did not finish the exercise with 100% score to strongly separate the scores of solved exercises from unfinished ones.

		Working Time Percentile			
		$< 40\%$	$< 60\%$	$< 80\%$	$\geq 80\%$
Score	$< 40\%$	0	0	0	0
	$\geq 40\%$	0.2	0.2	0.2	0.1
	$\geq 60\%$	0.5	0.4	0.4	0.3
	$\geq 80\%$	0.6	0.5	0.5	0.4
	100%	1	0.9	0.8	0.7

Table 3.2: Definition of the values of the scoring function $\sigma(s, e) \in [0, 1]$. The achieved exercise score in exercise e between (0% and 100%) together with the working time percentile of student s within exercise e determine the resulting value of the scoring function as shown in the table.

Weights

We rank the scores $\sigma(s, e)$ based on the share of the topic t on the exercise e with $\rho(t, e)$ and based on the instructor supplied difficulty of the exercise $\delta(e)$.

Diminishing Function

Recent exercises better reflect the actual knowledge status of a student. Initial misunderstandings might have been clarified in the progress of the course exercises. In order to accommodate for this, we add the diminishing function (3.2):

$$\varphi(e, \mathcal{E}_s) = \frac{1}{1+e^{\frac{-3}{0.5\cdot|\mathcal{E}_s|}\cdot(\iota(e,\mathcal{E}_s)-0.5\cdot|\mathcal{E}_s|)}} \tag{3.2}$$

$\varphi(e, \mathcal{E}_s)$ is an adapted sigmoid function that ranks exercises based on the order $\iota(e, \mathcal{E}_s)$ in which they have been solved. Since we have different amounts of exercises for different topics, we adapt to the number of exercises $|\mathcal{E}_s|$.

Averaging and Normalizing

To compute the final score, we calculate the average of all factors and normalize it between 0 and 1.

Recommendation Algorithm

Our recommendation algorithm first ensures that students are capable of solving the presented bonus exercise so they will not be overburdened right away. From the pool of potential bonus exercises, we remove all exercises that are either too difficult for the student (*difficulty appropriateness*) or contain topics the student has not used yet (*concept appropriateness*). The potential benefit for each bonus exercise is assessed by re-calculating all affected topic scores under the assumption that the student fully solves the bonus exercise in optimal time. The sum of the resulting deltas of the topic scores is the potential benefit that we use to rank the bonus exercises and recommend them to the students.

For recommendation, we serve only the highest-ranked exercise to the students instead of providing them the ranked list. Many students want to solve all offered exercises with 100% score. Providing them with a long list of exercises thus may have negative effects. Giving students a list of exercises may also counteract our intention to improve their biggest weakness, as they may choose the exercises they estimate to be the easiest for them. If the list of ranked exercises remains empty, which happens if the student did not attempt any exercise yet or all potential exercises are too difficult at this point, we return the easiest exercise of the pool. Since the knowledge model is updated for each exercise a student attempts, the system can be asked to recommend more exercises if required.

3.4 Adapting Course Material

Educational material within MOOCs mostly consists of videos and self-test quizzes. Occasionally, additional reading material in the form of links to book chapters, external blog posts, or downloadable documents is provided. In order to expose weak learning material, course instructors require feedback from the students interacting with it. This is in general possible for videos and self-test quizzes, as the interaction of the students with the material happens directly on the platform and thus can be recorded. For the self-tests, the chosen answers and the required time provide meaningful insights into possible issues. For the videos, pause and resume events, as well as rewind actions and the chosen playback speeds, yield helpful evidence for ambiguous content. The quality of supplied reading material or additional resources (e.g. cheat cheats) can only be proxied via self-tests, as these documents lie outside of the MOOC-platform and thus do not offer a direct feedback-channel.

Given this general assessment of the suitability for automated feedback of material, it becomes apparent that the surrounding conditions for the field of programming education in MOOCs are nearly optimal. Students interact with the exercises extensively and perform all required steps to solve the exercises online within the supplied execution environment. In our research, we thus focused on the automatic detection of anomalies in the interaction of students with the programming exercises.

Automatic Anomaly Detection for Exercises

Programming exercises in MOOCs may serve many different needs. Their primary focus may be exploration, repetition, application, or a deep dive. Therefore, it is nearly impossible to set specific circumstances or metrics to be given or to be optimal for all cases. However, in a typical course context, the average exercise should not be out of the norm with regards to difficulty, pitfalls and required working time. As an increased overall difficulty or undesired pitfalls result in extended working times, all of the effects can be identified by a disproportionally high working time. Work on well-reasoned exercises thus stays within a given timeframe and requires a certain amount of execution runs. Trial and error is a given pattern in novice development and often necessary to establish a solid understanding of concepts. Thus, the required working time for training exercises is estimated to be higher than the working time needed by an advanced programmer to solve it. Nonetheless, good exercises should still be conceptualized to be completed in one session by beginners. Based on anomaly detection, we can thus outline unsuited or especially frustrating exercises. For a given course with rather homogeneous exercises, we expect the upper limit of desired working times to be within twice the average of all exercises of that course. With this course based approach, we prevent the exercises of courses aiming for advanced students, like our course on unit testing, from all being flagged as too hard on the one hand. On the other hand, we also prevent the more complex exercises of this course from erroneously increasing the overall mean working time of exercises intended for beginners. Founding analyses on course level, therefore, ensures a balanced base difficulty and thus improves the overall detection quality.

Analog to the reasoning with regards to our intervention concept (see Section 3.3.3), we want to prevent issues caused by a shallow data pool (cold start problem) or extreme data points, e.g., extremely low effort exercises within a course lowering the average working time to an unserviceable low value. We therefore set a lower bound for the automatic triggers to be at least 10 minutes.

If the average working time of an individual exercise thus lies over 10 minutes as well as over twice the average working times over all exercises, we take action: the author of the exercise receives an email that the exercise showed an anomaly in its results and is thus informed that students' learning experience possibly was not as intended. At the same time, the system sends emails to representative student populations asking them to share possible explanations and improvement suggestions. In order to receive a balanced feedback, we email three students who finished the exercise but struggled particularly hard with it, three students who solved the exercise on average, and three students who aced the exercise. Additionally, it is helpful to also incorporate the feedback of students who have not (yet) fully solved the exercise. We register the fact that the exercise showed an anomaly, as well as who got emails from the system in order to even out the effort of improving the content over all participants.

Contrary to our initial idea of simply asking the students for feedback directly via email and setting the "reply to" address to the exercise author, we opted to supply a feedback form on the platform, in order to keep the feedback centralized and augment it with the progress data of the specific student on the exercise being optimized.

4 Implementation

In this chapter, we describe the main workflows and data models of the developed software we use for our experimentation. To the largest part, this concerns the code execution environment CodeOcean and its extensions for experimentation. Additionally, we briefly explain the main workflow of the video conferencing solution CodePilot, which is used to enable our approach of remote tutoring. Parts of Subchapter 4.1.3 have been published in [160], respectively of Subchapter 4.2 in [163].

4.1 CodeOcean

openHPI and its related platforms openSAP and mooc.house offer courses that teach programming skills. For these programming courses to be effective, we offer practical coding exercises that let students run individual source code on our machines.

The employed system has been initially developed as part of a master's thesis by Hauke Klement in 2014. A general overview of the fundamental design decisions and their motivations can be found in his thesis [60] and a publication based on it [146]. Since CodeOcean's initial release, several substantial changes have been made to improve scalability, elasticity, functionality and user acceptance. For this reason, this section summarizes the current general architecture as well as specifics of important subsystems of CodeOcean that formed the basis for the experiments and research presented in this thesis.

4.1.1 Main Workflow for Student Interaction via Learning Tools Interoperability (LTI)

Students do not register or login on CodeOcean themselves but are forwarded to it from their learning platform. This saves the user from creating and maintaining an additional account, including remembering an additional username as well as password. This however also implies, that CodeOcean and the respective learning platforms must interact to share potentially sensitive data. In order to offer general compatibility with a wide range of learning tools on the one hand, and a proven as well as reliable communication infrastructure on the other hand, CodeOcean complies with the LTI standard[26] from IMS Global. Data is transferred via an OAuth encrypted connection that is build up between the LTI service provider (CodeOcean) and the and service consumer (the respective learning platform, e.g., openHPI) via a so-called OAuth key and an OAuth secret. The core data transferred consists of the following attributes:

- exercise id
- external id
- display name (optional)
- email address (optional)

The only required information is, therefore, the desired exercise and an artificial external id. Along with the information which LTI consumer registered the user, it is possible to uniquely identify each user via this external id. The display name is used to name students in collaborative tasks (e.g. requesting or writing comments), the email address is used to inform students on new comments on their requests or to ask them for additional feedback if necessary. For technical simplicity, CodeOcean additionally assigns a unique internal id.

The minimal requirement of a technical external id allows CodeOcean to be integrated into privacy-sensitive platforms such as K-12 education systems (e.g. the HPI Schul-Cloud) without ever knowing students names, therefore providing full pseudonymity.

[26] see `https://www.imsglobal.org/activity/learning-tools-interoperability`

4.1.2 System Architecture

The overall architecture of CodeOcean, depicted in Figure 4.1, has two main building blocks: The CodeOcean web application, and a docker instance providing containers as execution environments. The web application itself consists of a Javascript client responsible for the user frontend, a Ruby on Rails server backend, and a Postgres database providing persistency.

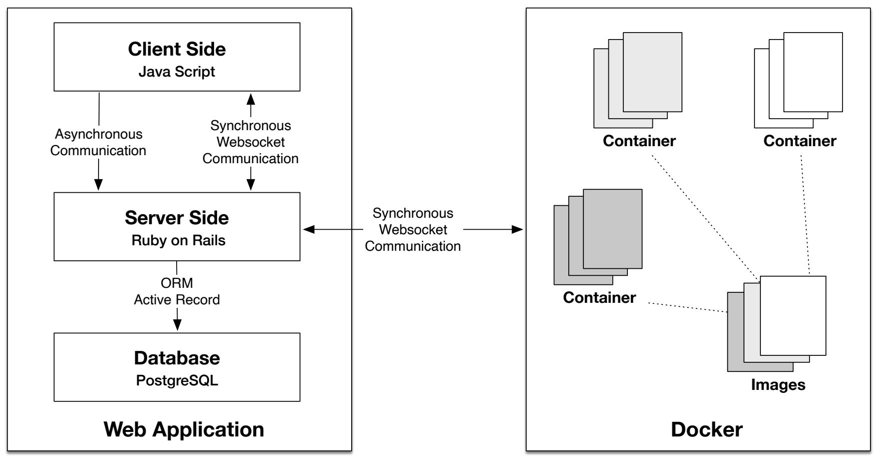


Figure 4.1: General architecture of CodeOcean: A three-tier client server web application interacting with pools of Docker containers encapsulating the different runtime environments for program execution.

The user client communicates with the server backend either via an asynchronous POST request to issue a scoring attempt or via a synchronous WebSocket connection to start an interactive execution run. The synchronous WebSocket connection allows for an exchange of input and output data during code execution, thus enabling small interactive games such as hangman, playback of drawing commands, or transmission of pictures as program results. The asynchronous scoring prevents any additional user interaction during program assessment and thus circumvents additional threats for cheating. Either way, the connection between the server backend is realized via a synchronous WebSocket connection, enabling both involved parts, the server process as well as the program execution process within the container, to exchange control commands during program execution, e.g., to terminate the program run earlier or on timeout. The WebSocket connection into the individual containers acting as execution environments is established via the HTTP Remote API of Docker, enabling a complete separation of execution environments and web application in case of a necessary scale-out.

4.1.3 Data Model

To further understand the internal structure of CodeOcean, a solid understanding of the data model is necessary. It currently consists of more than 25 tables, the overall presentation and explanation is therefore split into meaningful partitions of the complete data model. Analog to the research areas of this thesis, we distinguish between the following parts of the CodeOcean data model:

- the core data model, containing data to support the core functionality
- the interventions data model, reflecting data used for student interventions
- the feedback data model, holding interaction events and survey feedback
- the error data model, storing traces and errors encountered by students during their learning sessions

For the sake of clarity, technical values present in every entity (i.e. id, created_at, and updated_at timestamps) as well as indexes or additional join-tables are omitted in the overviews.

The core data model comprises central entities such as users on the platform, the exercises they work on and the submissions they create:

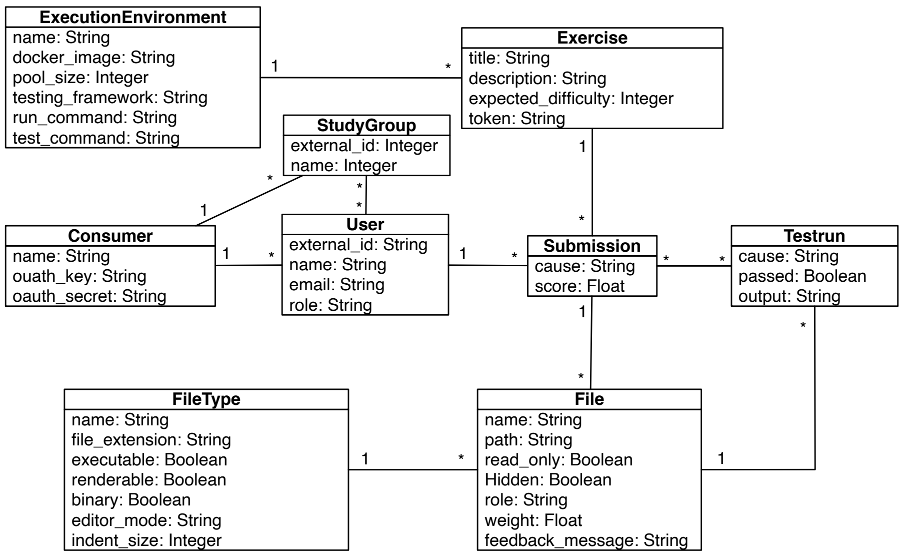


Figure 4.2: CodeOcean core data model.

The consumer table identifies LTI consumers, such as openHPI, openSAP or the Schul-Cloud. Data kept is minimal, we just store the name of the platform and its credentials. The main task of these consumers is to identify their users. Users can't exist without a consumer that "authenticates" them. Additionally to the necessary external_id which has to be unique for the consumer, we optionally store a name to be displayed and an email address to be used for notifications and feedback. Users can be grouped together as a study group, which allows to represent for example school classes in CodeOcean and thus give the assigned teachers access to group-specific data. Apart from the users, the exercises are another central element of CodeOcean. Basically, the exercises consist of a title, a description that holds instructions on how to solve them, optionally an expected difficulty rating set by the exercise's creator, and a generated token to embed and reference the exercise in LTI consumers. Each exercise belongs to one execution environment. As an example, an exercise that deals with foreach loops belongs to an execution environment called "java8". Execution environments reflect a complete environment suitable to execute the exercises belonging to them. To serve this purpose, additional to their name, they reference a docker image that usually comes with all necessary software build in, a pool size that tells CodeOcean how many of these environments should be kept available, the specification of a testing framework (e.g. JUnit for Java), and distinct run and test commands. The run and test commands are simply command calls which will be issued on the bash shell of the docker container for that execution environment. They are used to invoke program runs or respectively program tests for the associated exercises. These commands can include some variables to be supplied by CodeOcean on the moment of issuing the call, e.g., for testing Java programs, we call a make file with specific parameters supplying the class- as well as file-name: `make test CLASS_NAME="%class_name" FILENAME="%filename"`. Submissions are the central element, representing most user interaction with CodeOcean. Whenever a user runs or tests his written code, CodeOcean creates a submission. A submission is also created if the current progress is auto-saved, the user asks for help or transmits the score of the final solution back to the learning platform. Submissions save the cause for them being created as well as the achieved score if the cause was a test. Furthermore, they contain references to the user that created the submission, the exercise in which's context the submission was created and all necessary files for the execution. The referenced files can be mainly of three different kinds: files that users edited, files that are static and supplied by CodeOcean to enable execution, and files necessary to run tests. Files that are editable by users are created individually for each submission and reflect the current status of users' progress. Static files necessary for execution, e.g., make files or additional libraries, are not created on every submission but simply referenced. Also, static test files are referenced this way, additionally they are referenced in testruns. Testruns simply save the generated output when running a specific test file, whether or not the file could be run without errors and the cause for the test run (e.g. scoring or request for help). All files generally have a name, a (relative) path in the working folder, a role (e.g. whether they are the main file, a test file, a sample solution), binary options (whether they are read-only or hidden). Files used for tests optionally also have weight defining the relative influence they have on the overall score and a feedback message that is shown if the test fails. Each file also references a file type, which abstracts the file extension and several settings belonging to it, such as binary information whether the file is executable, renderable or binary. Also, we store technical information with regards to presentation in the editor here, the indentation size for tabs as well as the editor mode which defines syntax coloring in the frontend.

With this core data model, all basic operations can be carried out. As this thesis deals with approaches to improve users' learning experience, corresponding data also has to be processed. The entities in the following excerpt of the data model deals with interventions (see Figure 4.3). We kept the background color white for all entities belonging to the core data model and applied a color coding for new entities to make them better distinguishable. Entities dealing with just-in-time interventions are colored blue, entities supporting our approach via bonus exercises are depicted in yellow.

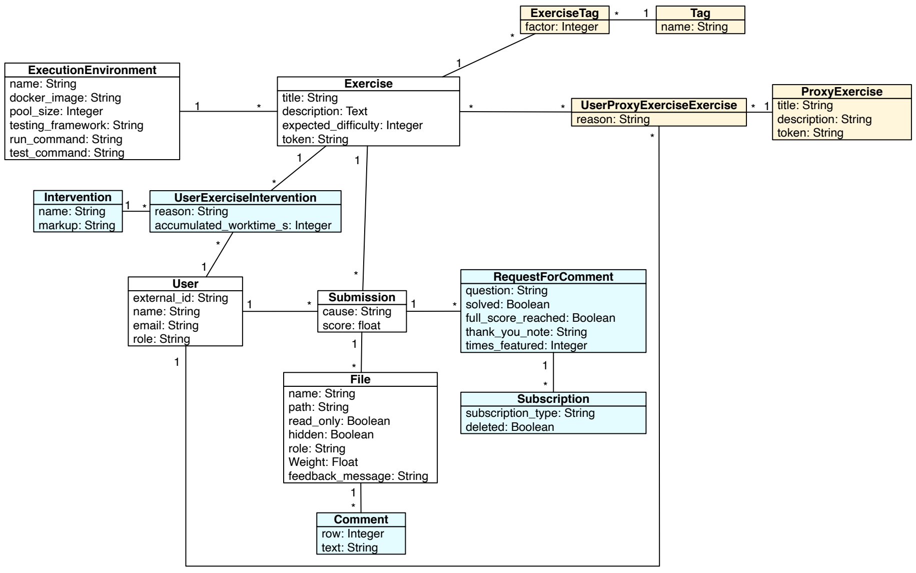


Figure 4.3: CodeOcean data model for interventions. Entities for just-in-time interventions are tinted blue, entities for tailored bonus exercises are tinted yellow.

The central element that purposely disrupt students' work-phases are the interventions. The system shows an intervention to a user if data suggest that the user is likely to be struggling. The intervention itself consists of a name, for example "break_intervention" or "rfc_intervention", and a markup string that will be rendered into the popup shown to users. To persist each intervention the system has made, we store an entry in UserExerciseIntervention. It connects all necessary entities, namely the user being affected, the exercise the user was working on, and the type of intervention, and adds a reason for the intervention (currently always "longWorktime"), as well as the accumulated working time the user spent on the exercise prior to the intervention. If the user acts on the intervention, most likely a request for comment is created shortly after. Such an entity stores the question the user typed into the popup, information whether the request has been marked as solved by its creator and whether full score was reached afterwards, as well as a potential thank you note the creator sent to his commenters. For analytical reasons, we also measure how often we redirected users to each specific request. The creator and all interested commenters can subscribe to a request for comment to be informed when new comments are given. Comments itself are stored in a small entity just holding the actual comment text, and the row of the source file it belongs to. Each comment is directly linked to the respective file of a submission, for which a user requested comments.

Our second type of interventions, the tailored bonus exercises, build upon the exercises of the core data model. The central element for bonus exercises are the proxy exercises. Attribute wise, proxy exercises only consist of a title, an internal description, and a token. The most important information of proxy exercises is kept via their connections to exercises. These reflect, which exercises the proxy exercises can redirect to.

The calculation which specific exercise to choose from a given set of exercises depends on the tags. Each exercise used in this context is annotated with a set of named tags, for example "loops" and "arrays". These annotations are weighted and allow a calculation of the potential knowledge gain a user is expected to gain in the best case when solving the respective exercise. After the algorithm has chosen a specific exercise to redirect from a proxy exercise based on the user's prior scores and the tags, this decision is saved in the entity "UserProxyExerciseExercise". The reason of the decision, for example the algorithm used and the specific knowledge values leading to the decision, can be saved alongside the connection.

Figure 4.4 shows the data model of processes we use to store explicit and implicit user feedback with regards to the exercises. We currently redirect 10% of random users that solved an exercise to a feedback form and encourage them to share their opinion as well as their perceived difficulty and working time. From the system side, we also store the actual calculated working time of the user.

Exercises can further be grouped into internal collections. If the collection is under anomaly detection, CodeOcean monitors the exercises within the collection for their average working times. If an exercise violates the dynamic boundaries of the collection, the system sends out emails asking for additional feedback, which is stored as explained above. Apart from that, CodeOcean stores events that students might emit, for example, copy and paste events originating from the source code editor. These copy and paste events that usually hint at missing information in course videos or the exercise description.

Implementation of Interventions

As described before, we issue interventions after a calculated amount of time at which we assume that students struggle. In order to keep the user interface as responsive as possible, we retrieve the needed information, the 75th percentile and working time of the student on the requested exercises, asynchronically in the frontend using Javascript. Once the necessary data is collected by the frontend, a timer is started. If the timer reaches zero, an intervention is shown. As we do not want interventions to show up when the student is not really working on the exercise. Therefore, the timer is automatically stopped if the browser focus of our coding platform is lost, e.g., if the student is distracted to check emails or to perform other actions. No interventions are shown after the student solved the exercise completely.

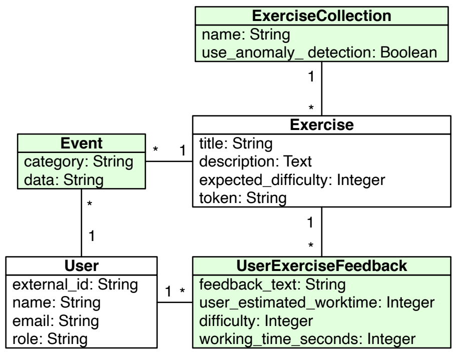


Figure 4.4: CodeOcean data model for user feedback. Entities for user feedback are tinted green.

The backbone of the automated learner feedback is the data that are collected from programming errors that students make. The sub-model for this data is visualized in Figure 4.5. Whenever a programming error or exception occurs, we save it as a structured error, that belongs to a specific submission. These structured errors comprise a number of attributes, that comply to predefined attribute templates. An example of such an error template attribute is a template that extracts the line number of a Java exception. The error template attribute uses a regular expression for that. Furthermore, CodeOcean uses so-called error templates to outline more specific comprehensive errors, such as “division by zero” exception. These comprehensive error templates then allow the system to show a descriptive, human-readable hint to the user, and further supply the details from the connected structured error attributes. Error templates are specific to execution environments, as the regular expressions are dependent on the used programming language and the employed testing framework.

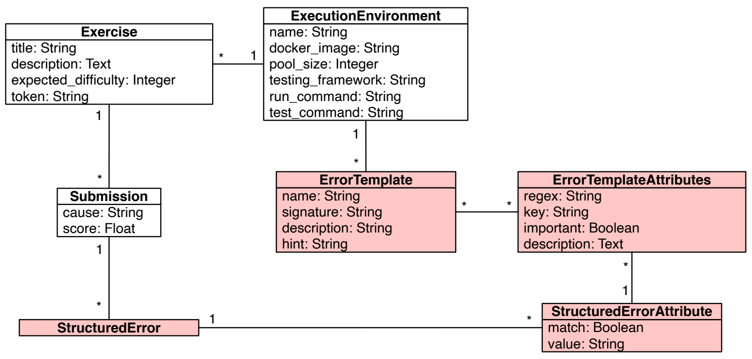


Figure 4.5: CodeOcean error data model (error entities tinted red).

4.1.4 Code Execution Workflow

When a user presses the run button in the browser, numerous steps on at least three systems have to be executed in order to present the expected program result. In this general overview, we deliberately abstract from specific implementation details, e.g., details on file copying and subsequent cleanup, in order to establish a comprehensible but solid understanding of the main code execution workflow between the client browser on the user's system, the CodeOcean rails backend on our server and the docker containers, currently running on the same as the backend, shown in Figure 4.6.

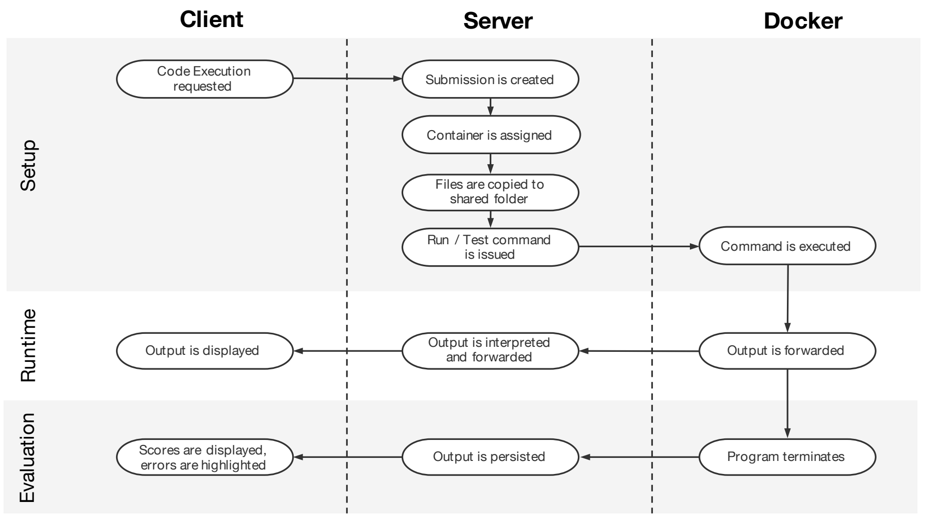


Figure 4.6: CodeOcean workflow to execute students' submissions.

Users express their intent to run or test the current status of their source code by pressing the respective run or score button. If the user wants to run the code, we establish a WebSocket connection between the client browser and the CodeOcean server as the first step to enable synchronous bidirectional communication. If the user instead wants to score the submission, we skip creating a WebSocket between the browser and the server, as there is no need for synchronous communication. CodeOcean then creates a submission to have a stable data set also usable for subsequent analysis annotation. Afterwards, a container is retrieved from the pool of available containers for the execution environment associated with the exercise that the user is working on. CodeOcean tests the availability of the container and copies all necessary files for program execution to a shared folder between the host file system and the container afterwards. As a next step, either the run or test command needs to be executed within the container. For this, CodeOcean establishes a WebSocket between itself and the docker container. The WebSocket is attached to the shell of the container and thus gives CodeOcean full control over the container within the limits of this shell. This step is performed regardless of whether the execution is a normal or a scoring run. Upon having established an acknowledged socket connection, CodeOcean sends the respective run or test command with the appropriate parameters set over the socket, followed by a return. This effectively causes the container to run the desired command and will return all program output to CodeOcean unfiltered and synchronously. In case special treatment of the program output is necessary, such as the transformation of turtle-drawing commands for our python courses, the rendering of a base64 encoded picture or terminating the program run, the program output is scanned on CodeOcean for control statements encoded as JSON. After potential transformations, the result is either buffered or directly forwarded to the client, depending on the requirements (scoring, or respectively just running a submission). Once the program terminates (either by successfully exiting or by exceeding the allowed program runtime), the beginning of the program output, as well as potential test results, are persisted for reference. In the case of a scoring run, the resulting scores are extracted and sent to the client browser together with the program output. In the case of a normal program run, this step is omitted. Lastly, the output is scanned for program errors to be extracted and highlighted in the users' browser. If the program run exited successfully, the used docker container is cleaned and returned to the pool of available containers. If the program run crashed or exceeded the time limits, the container is destroyed and CodeOcean issues a new container to be booted into the pool.

Despite the shift from Server-Sent Events (SSE) to WebSockets and some confined internal changes, this overall workflow remained stable over multiple years and is expected to endure over the timeframe of this thesis.

4.1.5 Feature Configuration

In order to support different scenarios and use cases, the offered feature set and the visual presentation of CodeOcean can be externally configured. This allows for example to enforce a read-online mode, to embed programs that can just be run, but not edited, in interactive worksheets for school usage. Similarly, also interventions or request for comments can be disabled if desired for didactical reasons. The desired options are sent as custom parameters during the LTI launch when starting an exercise. The full set of boolean options is presented in Table 4.1.

Custom Parameter	**Effect**
disable_run	Prevents running the program, e.g., for exam situations
disable_score	Prevents automated assessment, e.g., for exam situations
disable_interventions	Disables just-in-time interventions
disable_rfc	Disables RFC functionality, e.g., to focus discussions in class
disable_hints	Hides additional hints, e.g., for exam situations
disable_download	Removes download capabilities
hide_exercise_description	Hides the exercise description, e.g., to allow teachers to provide a different one
hide_navbar	Hides the title bar, including any navigation to support seamless integration
hide_sidebar	Hides the navigation sidebar, e.g., to prevent file switching
hide_test_results	Hides tests results, e.g., for exam situations
read_only	Removes editing capabilities, e.g., for demonstration purposes

Table 4.1: Supported custom LTI parameters to configure CodeOcean.

4.2 CodePilot

The prototype of our proposed tutoring solution is implemented as a separate Ruby on Rails application that integrates into CodeOcean via an iFrame. For the actual video conferencing part, we rely on the open-source project Jitsi Meet. With respect to the workflow, our prototype supports the actions shown in Figure 4.7. In order to realize the flows, we modeled the required data as shown in Figure 4.8.

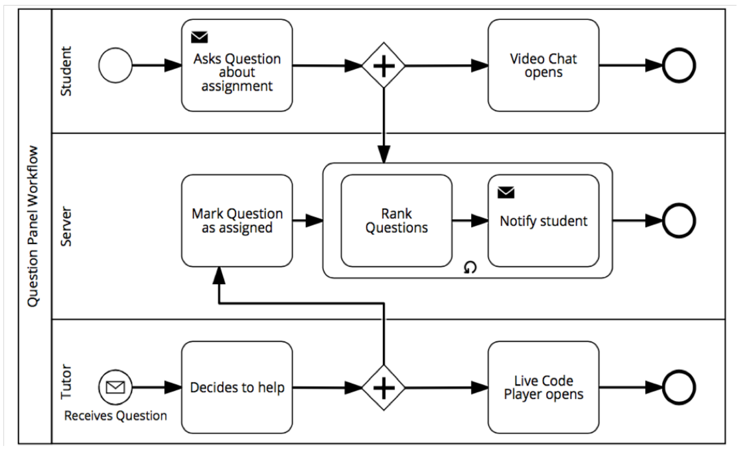


Figure 4.7: Main workflow for tutoring sessions in BPMN [163].

The core element within this data model is the question, which usually has attached two participations of students via their user IDs. Technically, to also support other use cases like public demonstration sessions or group discussions, CodePilot allows an unlimited number participations.

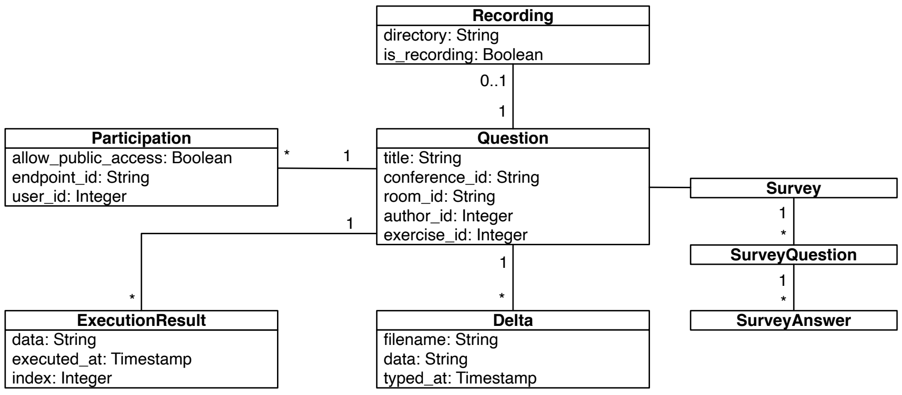
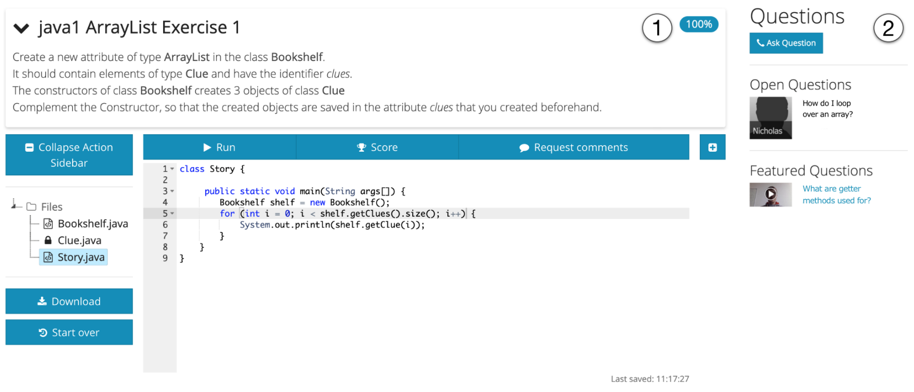


Figure 4.8: Data model of CodePilot.

Whenever a student starts a program run during an ongoing tutoring session, we save the execution result to allow for subsequent playback. All changes to the source code are saved as deltas, to support a seamless playback and synchronization with the tutor. If a session ends, a recording entry is stored and all participants are surveyed for their opinions (rating of the helpfulness of the tutor, information whether the problem was solved, the allowance to use the recordings, and additional free text).


java1 ArrayList Exercise 1

Create a new attribute of type **ArrayList** in the class **Bookshelf**.
It should contain elements of type **Clue** and have the identifier *clues*.
The constructors of class **Bookshelf** creates 3 objects of class **Clue**
Complement the Constructor, so that the created objects are saved in the attribute *clues* that you created beforehand.

```
class Story {

    public static void main(String args[]) {
        Bookshelf shelf = new Bookshelf();
        for (int i = 0; i < shelf.getClues().size(); i++) {
            System.out.println(shelf.getClue(i));
        }
    }
}
```

Last saved: 11:17:27

Open Questions
Nicholas
How do I loop over an array?

Featured Questions
What are getter methods used for?

Figure 4.9: User interface of CodePilot.

For students, the user interface of our prototype presents itself as shown in Figure 4.9. On the left side (1), students see the usual controls of our execution platform. The only difference is that changes made in the editor during an active conference session are recorded and transferred to the tutor via a synchronous webRTC connection. We currently restrict the synchronization direction to be unidirectional towards the tutor. In order to potentially enable full bidirectional synchronization for full pair programming, additional methods, like operational transform (OT), are required [42].

On the right side (2), the main compartment of our prototype is embedded. We show open questions and existing recordings here. If the student clicks “Ask question”, an input box to phrase the question opens up. Afterwards, the student is forwarded to an empty meeting room that opens up in the iFrame (2). The coding environment (1) stays as it is; the progress on the exercise is not interrupted in any way. If an existing recording is chosen, which is available under “Featured Questions” the current progress is persisted and the recording is played back in fullscreen, to make room for the recorded code to be presented. With regard to usability, we kept the controls and new elements as minimal as possible.

5 Evaluation

MOOC research has undergone a natural shift from technical proof-of-concept demonstrations, over first insights about learners' motivation and verdicts, to comparisons of general course metrics employed to reflect content effectiveness and learning success [126]. However, as Justin Reich argues, many of these initial findings do not have the setup to further advance the field of learning research. Boiled down to the statement "It does not require trillions of event logs to demonstrate that effort is correlated with achievement." [126], it becomes evident that research aimed to advance the current status quo has to be based on mindfully proposed hypotheses and carefully crafted experiments.

Therefore, in this chapter, we will first formulate our hypotheses, and then describe the general setup of our research, as well as the specific details for the individual experiments, before further explaining the employed methods. Following on that, we present the gathered results, as well as the respective discussions of the likely conclusions of the results towards the hypotheses. Several smaller experiments have been run alongside our main research areas. They brought up additional insights but did not directly contribute to the main argumentation. Subsequently, they are presented with slightly less detail.

Lastly, we show several interesting findings that surfaced from our data that are of general interest but come without a further connection to the hypotheses we followed. We refrain from giving a detailed discussion for these findings, as the proper and qualified evaluation of the results would require additional experiments, surpassing the scope of this thesis.

5.1 Research Hypotheses

Advancing from the universal truth in MOOCs, that effort correlates with achievement, in this case reflected as higher course scores, we will further specify influences that improve learning success. We analyze the following high-level hypotheses by answering several sub-hypotheses. The sub-hypotheses further specify, for example, which metrics are suitable to reflect learning success, or which further criteria are helpful to distinguish subgroups in our audience in order to pinpoint our results and answer further detail questions.

1. **Request for Comments improve students' learning success**
   The possibility to ask for help increases overall scores. Receiving help reduces the required time to successfully finish an exercise.
2. **Just-in-time interventions have an effect on students' behavior**
   The shown popup windows increase the number of requests for help, respectively the breaks taken. They furthermore cause students to reflect, improving their performance.
3. **Tailored exercises help students to overcome their weaknesses**
   Students require more time to solve an exercise which targets their specific weaknesses than students who were assigned to the exercise at random. Students further agree on the recommended topic as their weakness.

Before we present the results, we first describe our experiment setup, the respective audience, the employed methods as well as the used metrics.

5.2 Setup

The experiments resulting in the main findings presented in this thesis often required a large number of participants, allowing to detect possibly small effects only on some individuals of a group. Running an experiment in a MOOC requires careful preparation of the study, as errors or lapses cannot be fixed while the experiment is ongoing and require an additional, time-consuming iteration or even the preparation of an entirely new MOOC. Therefore, the complete progress of our main experiments measuring the effects of just-in-time intervention and request for comments spans over several courses, with the first courses acting as a proving ground to validate the assumptions to fuel a full-fledged analysis in a controlled randomized study later on. In the following, for each experiment, we will focus on the course and the findings contributing the most important results concerning the underlying research questions. Therefore, most results will be drawn from our latest course Java 2018, which represents our most elaborated setup and resulted in the most comprehensive dataset. For this reason, the following descriptions, for example of the instructional design of our courses, will be exemplified with the Java 2018 course, while the same principles and approaches have also been applied during the conceptualization of all other courses. If a particularly interesting finding was replicated from a former experiment or revalidated in another dataset, we will mention this in the respective description of the results.

5.2.1 Courses

The majority of experimenting was done on the data of the German course "Objektorientierte Programmierung in Java" run over six weeks on openHPI in 2017, and its redesigned English counterpart "Object-Oriented Programming in Java" run on openSAP in 2018. Some side experiments were conducted on the two-week-long workshop courses, "Java Workshop: Einführung in eine Java-Programmierumgebung (IDE)" (2017) and "Java Workshop - Einführung in die Testgetriebene Entwicklung mit JUnit"(2016) run on openHPI in German language.

	Java 2017	**Java 2018**
Start Date	March 27th (2017)	June 13th (2018)
Duration in Weeks	6	6
Course Platform	openHPI	openSAP
Course Language	German	English
Enrolled Students	8,781	18,856
Attendees	6,610	9,504
Active Students	6,008	5,581
Granted Certificates	2,124 (35.4%)	2,317 (41.5%)
Granted CoPs	3,638 (60.6%)	3,377 (60.5%)
Forum Posts	7,673	6,222
Request for Comments	5,381	3,109

Table 5.1: Key metrics for analyzed Java courses.

Table 5.1 visualizes the most important key metrics for the main courses analyzed in this thesis. The organizational course details can be found at the top of the table, while the resulting performance indicators are to be found in the lower part.

The two courses are relatively similar, both offering six weeks of content over a comparable course runtime. They were conducted in two consecutive years, by similar-sized teaching teams. The major differences are the course language, having switched from German in the 2017 version to English in the 2018 version. In order to ensure an audience that was unaffected by the previous iteration, also the course platform was changed from openHPI to openSAP.

Within the key metrics, two major differences are directly visible: the number of enrolled students and the number of Request for Comments differs immensely for 2017 and 2018. Considering the number of enrolled students, if one regards the number of attendees measured after three weeks, which only reflects students that logged in on the course since the beginning, the previous gap closes to a large part. "Active students" describes students who accessed at least one exercise on CodeOcean and scored at least one point. Using this metric as a basis for further comparison, it becomes apparent that both courses finished with similar results, resulting in about 35-40% of active students reaching the Record of Achievement, a certificate that is granted if a student scored at least 50% of the possible graded points. About 60% of each course's active student audience also got a Confirmation of Participation (CoP), which is issued if at least 50% of the offered learning material was accessed. The number of forum posts was about 20% higher in the German course, which can still be regarded as normal variation.

Thus the only remaining significant difference is the number of Request for Comments issued. Within the German course, 5,381 requests were issued, while in the English course only 3,109 were made. This results in a difference of 73% percent more Request for Comments in the German course compared to the English one. To set this discrepancy into context, it is important to know that in the Java 2017 course, all students were actively encouraged to use this functionality via a course video in the first week. In the Java 2018 course, some students were not able to use the feature at all, and for the reason to not endanger the A/B test conducted in the course, the course team also did not actively promote the Request for Comments in any way.

5.2.2 Course Setup

In the following, we will explain the content used as the foundation of the courses in more detail.

The difficulty of the courses was on the same level, also the covered topics have been the same with regard to the tested and graded content. Some differing, optional topics were offered at course end, however, they neither directly nor indirectly influenced course results, as the introduced areas there were independent of the graded content (e.g. usage of the distributed versioning system git). Of course, with a completely re-created course in another language, differences concerning the understandability of certain units will occur, however, we are confident that the average understandability stayed on the same, high level. For the subsequent explanations, we will focus on the English version of the course, while the same principles and argumentations also apply for the german version.

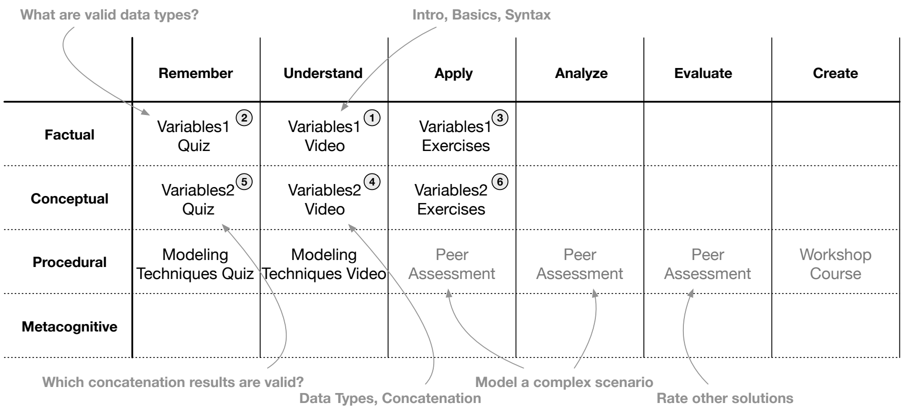
Instructional Design: Course Content

The content and skills conveyed in the programming courses, as well as the activities to be performed by the students, were carefully considered and sketched out, given the restrictions on time as well as didactical measures available. Within each course, the content was structured on a weekly basis, being mostly self-contained and aimed to be balanced in terms of required time as well as complexity of the presented concepts.

As being described in Subsection 2.1.1, Krathwohl's Taxonomy is well suited to classify and evaluate learning content. An example of such a classification is given in Figure 5.1 with the first three units of the first week of the Java course run in 2018, dealing with variables.

Upon the creation of the course, the instructors agreed on learning objectives and sketched out content areas. The course design was an iterative process based on multiple models and theories, fitting in content into weeks and defining learning goals to be achieved within the six course-weeks. The influences and processes for the decisions will not be explained in detail here, however, we will shortly exemplify the use of blooms revised taxonomy.

New content and concepts are first introduced via videos in our online course. For our example, we focus on the introduction of the concept of variables first. The explanation in the video (1) covers a short introduction into the topic, an outline of potential benefits and also several examples for using them, thereby also showcasing the syntax. Overall, the information presented is mostly factual, establishing a basic understanding to build upon later on. Subsequently, the initial understanding is checked within an ungraded multiple-choice quiz (2), which we call a self-test. The aim here is to fortify the knowledge creation by encouraging the student to remember presented facts. To further follow this goal of solid fortification, students then have to solve practical programming exercises (3) in the execution platform CodeOcean, thereby applying the presented knowledge on their own. The sequence of "video, followed by a quiz, followed by several practical exercises" is applied for each presented concept or subconcept within the programming courses.


What are valid data types?
Intro, Basics, Syntax
Which concatenation results are valid?
Data Types, Concatenation
Model a complex scenario
Rate other solutions

	Remember	Understand	Apply	Analyze	Evaluate	Create
Factual	Variables1 ② Quiz	Variables1 ① Video	Variables1 ③ Exercises			
Conceptual	Variables2 ⑤ Quiz	Variables2 ④ Video	Variables2 ⑥ Exercises			
Procedural	Modeling Techniques Quiz	Modeling Techniques Video	Peer Assessment	Peer Assessment	Peer Assessment	Workshop Course
Metacognitive						

Figure 5.1: Learning items classified into Krathwohl's taxonomy.

Variables, being a fundamental as well as an important part of programming, are a concept that requires students' to grasp the concept of abstraction. In order to facilitate comprehension, the concept is split into several of the aforementioned "video, quiz, exercises" sequences, each focusing on consecutive subconcepts building on top of each other. These different focus points or subconcepts, usually resemble and belong to the knowledge dimensions of Krathwohl's Taxonomy. The second video on variables (4) introduces data types and the concept of concatenating different variables. Apparently, the video conveys factual as well as conceptual knowledge. This serves as an example that an exact classification is often not possible, as the actual learning content may cover different areas in the knowledge as well as in the cognitive process domain. Krathwohl's taxonomy aims to uncover shortcomings of available material and outline focus areas, to allow for discussion and content optimization. Whenever facing unclear or ambiguous classification decisions, one can either place the content in all affected areas or just place it in the most important area, which usually is the position representing the instructors' goals. For the aim to exemplify the model and focus areas of our programming course, we decided on the latter. The aim of the video is to build upon the factual knowledge presented in the variables1 video and present concepts to gain flexibility in programming. As new content on the factual level, data types are introduced out of the necessity to represent different types of data in our programs. Similar to the first sequence, a self-test quiz (5) and some exercises (6) were placed after the video, this time covering mainly conceptual knowledge instead of factual knowledge.

When classifying the majority of learning items in the matrix, it becomes apparent that the course mainly addresses the areas factual and conceptual on the knowledge dimension, and the areas remember, understand and apply on the cognitive process dimension. The state on the knowledge dimension appears valid, because an introduction course is likely to focus on the fundamental areas. The procedural area within the cognitive process dimension, dealing with the"knowledge of criteria for determining when to use appropriate procedures" [84] amongst others, is reached within the fourth course-week. In this week, the course introduces modeling techniques to map real-life scenarios to object-oriented scenarios and further reason about the representation of relationships. This increased abstraction on the one hand, and the existing body of factual and conceptual knowledge on the other hand, allows to cover the procedural area. The metacognitive area, covering strategic "[...] knowledge of cognition in general as well as awareness [...]" [84] is not aimed for in the course. Subsequent weeks focus on technical details and give an overview of related areas within computer science, thereby offering additional conceptual and factual knowledge.

With regard to the cognitive process, the levels that are easily reachable differ between face-to-face education and online education. Despite Krathwohl stating "[...] objectives from Understand through Create are usually considered the most important outcomes of education, [...]" [84], MOOCs should also emphasize on the remember category, as fortification of knowledge is especially important when exposing the material towards students the first time. Reaching the "higher" levels (on the right side), is hard within MOOCs, as they usually require far more time from students to be spent on the one side, and very flexible and task-specific feedback on the other side. While the majority of content within traditional MOOCs thus is found in the cognitive process areas of understand and remember, the regarded programming courses add additional value by covering the apply area. With additional offerings, such as peer assessments and the Request for Comments explained later, the respective courses further cover the additional areas of analyze and evaluate. Within an independent workshop offered optionally after the course, the instructors rounded up their educational approach and gave participants the opportunity to develop an "original product" of their own and thus also covered the create area.

5.2.3 Used Data Sources

We combined datasets from the course platforms openSAP, respectively openHPI, with datasets from our programing platform CodeOcean, in order to, e.g., correlate survey answers with experiment groups. Datasets used from the course platforms are a general course export, as well as the course start and course end survey. The general course export contains information reflecting aggregated course progress and achieved scores, as well as demographic data that was optionally supplied. The surveys supplied further information present prior to the course (for example, details about prior knowledge and students' expectations) and opinions after the course (e.g. subsequent appraisal collected in the course end survey). From the program execution platform, we extracted detailed information regarding our interventions as well as surrounding details and scoring metrics. The main scoring metrics used are required working times as well as achieved scores. Concerning our interventions, we gathered amount and timestamps of issued request for comments, resulting comments, and issued just-in-time interventions. Each of these data points was captured in the context of its creation, i.e., the exercise it belongs to and the associated student. On the basis of the combined data, ratios of future interest, such as "RFCs per Student" or "Comments per RFC" were deviated. All analysis of data was conducted in compliance with the data privacy statements of openHPI/openSAP.

5.2.4 Audience of Java 2018

During the course runtime, 7,186 students accessed at least one exercise on CodeOcean, of which 5,581 are considered active for reaching at least one point. The gender distribution of our audience is visualized in Figure 5.2. 56% of all students did not share their information, 37% identified as male, 7% as female. The shares between the students who accessed at least one exercise on CodeOcean and only those who reached at least one point are similar. Regarding only those students who shared the gender information, we thus have a ratio of 16.6% females to 83.4% males. This ratio can be generalized under the assumption that the ratio of males and females not stating their gender is relatively equal.

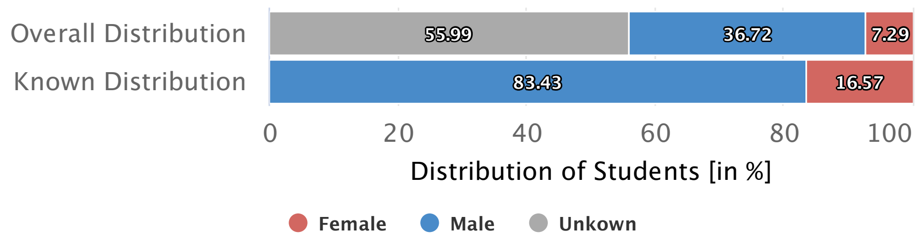
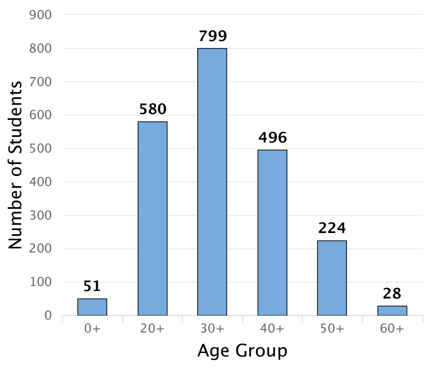
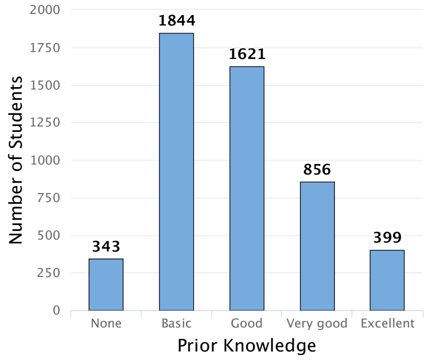


Figure 5.2: Overall gender distribution of active students. About 44% of our students shared information about their gender, the distribution within this known population is approximately 83 males to 17 females.

The age distribution of our active participants is as shown in Figure 5.3a. 61% of the active students did not share their age information with us. Apart from this, the resulting numbers resemble a normal distribution with the highest shares between 20 and 50.



(a) Age distribution



(b) Skill distribution

Figure 5.3: General metrics for active students.

For future analyses, the prior knowledge of students will play an important role. The self-assessed prior knowledge categories “none”, “basic”, “good”, “very good”, and “excellent” are mapped to interval skill levels between zero and five when calculating averages. 4,191 of 5,581 active students shared their prior skill levels (75%). The prior skill distribution of our audience can be found in Figure 5.3b.

Finishing up the demographic overview of our audience, Figure 5.4 visualizes the distribution of our learners over the world. The three most prominently represented countries are Germany, the United States of America, and India. Apart from the high share of students from these three countries, learners all over the world were reached.

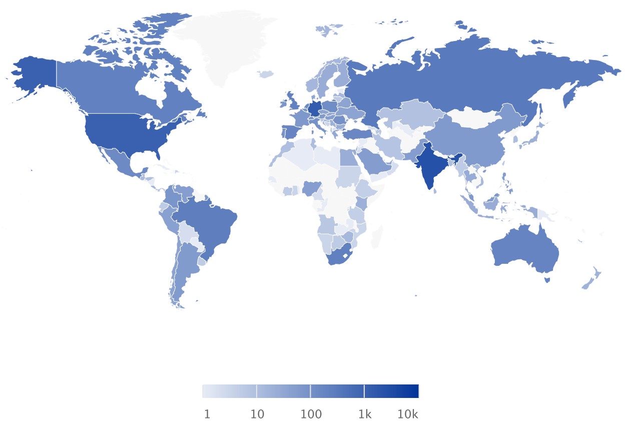


Figure 5.4: Number of students per country (on logarithmic scale).
WorldMap: Highcharts.com©Natural Earth

5.3 Methods and Procedures

In general, we analyze data gathered via surveys, acquired from the course platform, and acquired from the code execution platform to gain insights concerning our experiments and answer our research hypotheses. To assess prior knowledge, we collected self-stated skill levels and conducted a short, ungraded multiple-choice test about OOP and programming concepts. In Subsection 3.2.3 we presented the metrics potentially suitable to describe and explain effects of our experiments. Of those metrics, especially the course scores and working times showed the highest relevance. Other metrics are reflected in the aforementioned metrics, thus allowing us to simplify analysis by substituting them in most cases. If we encountered significant deviations within one of these general metrics, we further analyzed the additional, more detailed metrics. Considering the substitutions, results from unit tests as well as quiz scores are reflected in the overall course scores, as this is the sum of all unit test and graded quiz scores. Working times further correlate with runs, program assessments as well as the number of gaps, thus allowing us to substitute in this dimension. The impact of the prior skill level towards other key metrics is going to be described in Subsection 5.4.1.

After the course runtime, we surveyed our students on their perception of our course and the experiments, including their valuation of Request for Comments, the just-in-time interventions, and the tailored bonus exercises.

The created Request for Comments were manually labeled after the course, in order to structure and analyze their content and intent, e.g., their style, friendliness, and expressiveness.

Further details of the used methods slightly differ for the experiments that have been carried out, e.g., the composition of the experiment groups. We will, therefore, describe them separately per experiment in the following. All in all, two major experiments have been conducted, which partly covered multiple hypotheses and research areas. The first experiment covered Request for Comments as well as just-in-time interventions, as the interventions aim to increase the amount of RFCs issued. Within the second experiment, we analyzed the effects of the supplied tailored bonus exercises.

5.3.1 First Experiment: A/B Testing of Just-in-Time Interventions and Request for Comments

In order to determine the effects separately, we built seven A/B-testing groups in total. The assignment of students to the experiment groups was done based on the user id on CodeOcean, which is an independent and artificial value. This ensures a randomized distribution. The individual setup of the testing groups is reflected in Table 5.2.

Group	Interventions	Request for Comments	Student Share	Abbr.
1	none	disabled	10%	ND
2	none	hidden	10%	NH
3	break	hidden	10%	BH
4	RFC	hidden	10%	RH
5	none	shown	10%	NS
6	break	shown	10%	BS
7	RFC	shown	40%	RS

Table 5.2: Experiment groups for just-in-time interventions and Request for Comments.

Group 1 (abbreviated as ND for **n**one-**d**isabled) serves as our control group. Students within this group did not have the possibility to request comments at all. Likewise, they did not receive any interventions. This was technically implemented by disabling the commenting feature system-wide, removing the respective button from the user interface and removing all intervention triggers for them. Students being assigned to groups two to four were able to request comments, however, they never received any feedback from peers. Their requests were excluded from the selection process when forwarding a suitable student to open requests. This "hiding" of requests allows us to determine the effects of actually receiving comments. Groups two to four differ with regard to the just-in-time interventions being performed on them. While students within group two received no interventions, students of group three received break interventions suggesting to take a break, and students of group four received interventions suggesting to ask for help. Experiment groups five to seven were able to use request for comments without any impediments. Their requests were included in the selection process and thus they received feedback. Again, the groups differ concerning the interventions being issued, with group five receiving no interventions, group six getting break interventions and group seven being served RFC interventions. The experiment groups one to six each make up 10% of our MOOC's student population, while the remaining 40% of students are placed within group seven. This allocation was done in order to aim for an optimal treatment group (receiving RFC interventions and having full access to the RFC feature) to be sized as large as possible while retaining all other groups representing the possible combinations of conditions to be individually represented in sufficient numbers.

Regarding this setup from the different viewpoints of analysis, we end up with 60% of our audience being able to request help and receive comments, 30% solely being able to request help (but without receiving any answers) and 10% missing the feature completely. Concerning interventions, 30% of the students did not receive interventions at all, 20% of the students received break interventions, and 50% received RFC interventions. These resulting sums can also be found in Table 5.3.

		Interventions			
		none	break	RFC	Sum
RFCs	disabled	ND (10%)			10%
	hidden	NH (10%)	BH (10%)	RH (10%)	30%
	shown	NS (10%)	BS (10%)	RS (40%)	60%
	Sum	30%	20%	50%	

Table 5.3: Combination of experiment groups and resulting shares of students.

5.3.2 Second Experiment: A/B Testing of Tailored Bonus Exercises

Similar to our first experiment, we split the students into control and treatment groups based on an independent, artificial value. In this case, we used the microseconds of the timestamp of their initial registration on CodeOcean, taking place implicitly when they start their first exercise from any course platform. This random value is furthermore independent of the grouping value used for the A/B test of our interventions and request for comments (the user id), allowing to conduct both experiments in parallel without resulting in potentially skewed experiment groups. Despite the threat that the experiments could potentially interfere with each other, in such a way that solving tailored bonus exercises massively improves course scores (albeit we assume that the overall effect is too small to be registered), the grouping based on independent variables ensures that effects from one experiment would affect all groups of the other experiments evenly, given the large number of students present in our courses.

For the experiment on tailored bonus exercises, we assigned just three groups. The specifics of these groups can be found in Table 5.4.

Group	**Bonus Exercises**	**Student Share**
1	dummy	20%
2	random	20%
3	tailored	60%

Table 5.4: Experiment groups for tailored bonus exercises.

The first group, once more acting as our control group, always received dummy exercises. These exercises were intentionally simple with the aim to inhibit specific learning effects while offering at least some repetition in order not to waste the respective student's time. The second group randomly received one of the crafted bonus exercises. The random assignment allows us to measure the effects of the actual "tailoring" being performed with our algorithm and thus serves as another kind of control group. The third group receives a crafted bonus exercise based on the recorded weaknesses resulting from the student's prior performance. The test groups comprised 20% (group 1), 20% (group 2) and 60% (group 3) of our audience respectively.

5.4 Results

The basis for all of the following examinations will be the Java 2018 course unless otherwise stated. We refer to previously published material if necessary when comparing the recent results with previous ones. Also unless otherwise stated, we always use the group of active students as our foundation. Students who never showed up or never achieved at least one point were not affected by our treatments. Removing those inactive students from the data set thus improves overall data quality for the given hypotheses.

The first metric to evaluate the outcomes of a MOOC usually is the achieved course score per student. The overall distribution of course scores of all 5,581 active participants is depicted in Figure 5.5.

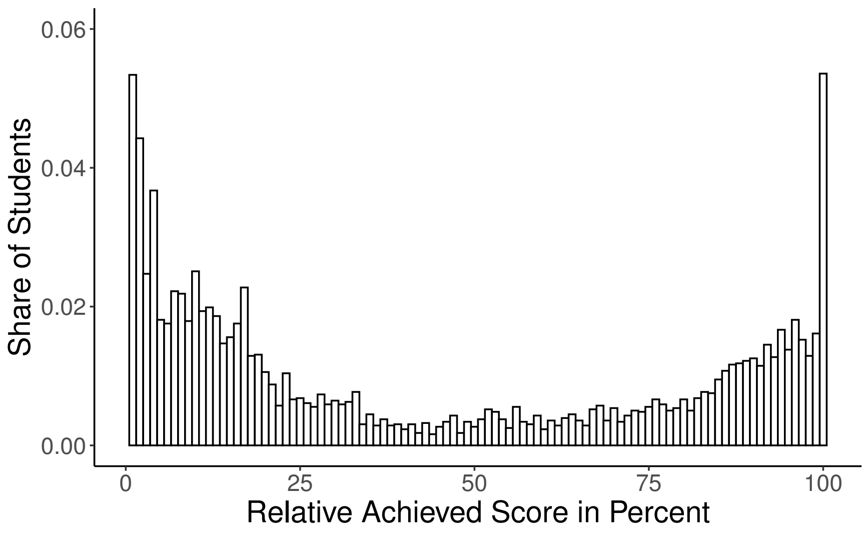


Figure 5.5: Score distribution of all active students.

We see a typical U-shaped distribution, with a large fraction of students residing in the lower score areas up to 20% of the available score and another accumulation of results at the 100% score mark. This distribution is typical for analyses of graded work or exams, reflecting a share of students having given up on the lower end, and determined top-scorers at the upper end. The sharp spike present at the 100% mark in this graph is partly caused by the possibility to reach a score of more than 100% through bonus activities such as the peer-assessment. Scores above 100% were subsequently cut to 100%, causing the visible spike.

5.4.1 Request for Comments and Just-in-Time Interventions

The results on the effects of request for comments and the impact of just-in-time interventions make up the largest part of the evaluation. To ease getting an overview of the numerous details presented, the following subsections will each be headed by the most relevant finding as the headline.

Request for Comments Increase Course Scores

When aggregating data per intervention group, the key metrics shown in Table 5.5 emerge for the active students.

Group	CoP	RoA	Avg. Score	Avg. Score Finisher
ND	47.53%	36.20%	38.84%	82.83%
NH	51.82%	41.24%	43.33%	83.85%
BH	49.28%	39.75%	42.41%	84.47%
RH	50.54%	42.32%	43.35%	84.95%
NS	52.71%	41.86%	44.09%	85.73%
BS	52.32%	43.75%	44.72%	85.01%
RS	53.33%	42.52%	44.17%	84.05%

Table 5.5: Key metrics for experiment groups. Enabling RFCs and issuing interventions improves all key metrics.

Our control group ND shows the weakest performance with regards to all presented key metrics. Despite the absolute changes being relatively small, the trends are coherent. The share of students having reached the Confirmation of Participation (CoP) by accessing at least half of the offered course content is 47.53%. The Record of Achievement (RoA) was granted to 36.2% of our control group, on average they achieved 38.84% of the maximum possible score. Students of the ND group who finished the course (meaning that they received an RoA) scored 82.83% of all possible points on average. The table further shows common behavior for the three intervention groups sharing the “hidden”, respectively “shown” property for the RFCs. The share of students having achieved the CoP ranges around 49% to 52% for the “hidden” groups, while the “shown” groups reached 52% to 53%. For the share of students having earned a RoA, “hidden” groups range between 40% and 42%, while the “shown” groups range between 42% and 44%. Similar tendencies are also visible for the average achieved scores (between 42% and 43% for “hidden” vs. around 44% for “shown”), as well as the average scores of finishers (84% to 85% vs. 84% to 86%).

Group	CoP	RoA	Overall Score	Score RoA
ND	0%	0%	0%	0%
NH	9.03%	13.93%	11.56%	1.23%
BH	3.68%	9.81%	9.20%	1.98%
RH	6.32%	16.91%	11.64%	2.56%
NS	10.90%	15.63%	13.52%	3.49%
BS	10.08%	20.86%	15.15%	2.63%
RS	12.19%	17.46%	13.73%	2.01%

Table 5.6: Relative differences in key metrics for experiment groups.

Compared to the results of the group with no interventions and a disabled RFC feature (group ND), the following relative changes displayed in Table 5.6 can be observed for all students. Taking the group ND as a baseline, for example, 9.03% more students achieved the Confirmation of Participation in group NH.

Aiming to improve the learning success of our students, reflected by their achieved scores, it is worthwhile to compare the score distributions separated by the experiment groups. Histograms of the respective distributions, here depicted with a bin size each comprising 5% of the total achievable score, are shown in Figure 5.6.

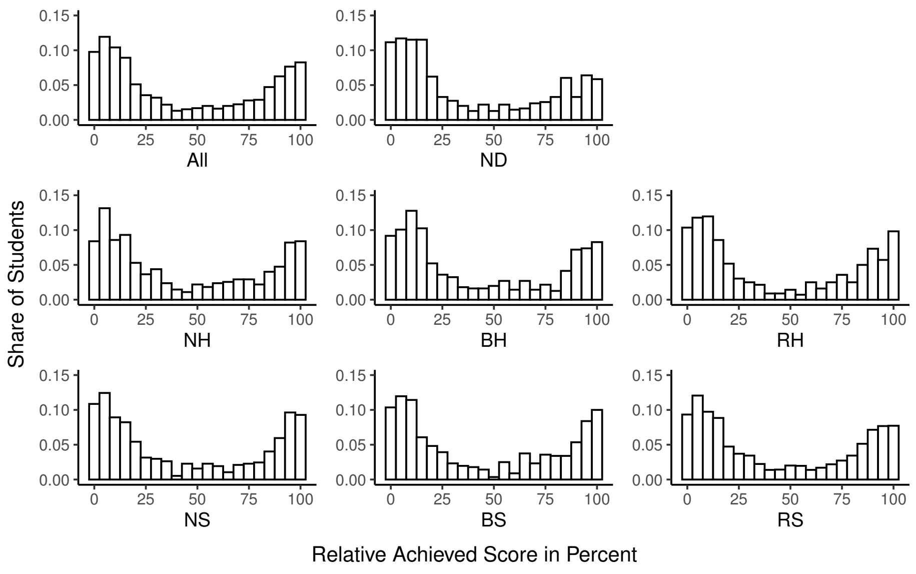


Figure 5.6: Score distribution per experiment group.

Drawing insights from these histograms was not easily possible, as they do not share a clear trend.

In order to unveil further insights, we alternated the visualization to a combination of density plots and boxplots, thus giving an overall impression of the distributions, while at the same time clearly showing the respective quantiles.

The density plots and boxplots shown in Figure 5.7 better reflect what was already indicated by the aggregated results shown in Figure 5.5. Group ND performs worst in general, with the median as well as the 75$^{\text{th}}$ quantile mark being located at lower scores than all other groups. The differences between the “hidden” and “shown” groups are only partly visible in the graphs, with the medians of the “shown” groups placed slightly higher than the ones of the “hidden” groups. The 75$^{\text{th}}$ quantile marks differ only marginally, thus not providing any insight. We additionally visualized the average achieved score per experiment group as a dotted vertical line.

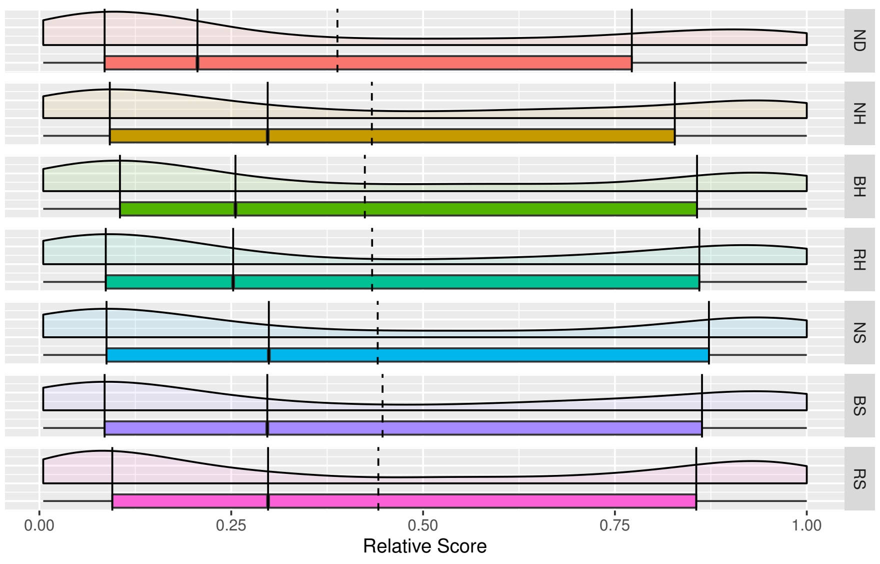


Figure 5.7: Combined density- and box-plots of the scores achieved by students of our experiment groups. Means, depicted by dotted lines, increase significantly between ND and the other groups.

The differences between the average achieved scores are statistically significant, as indicated by statistical hypothesis tests. We ran Student's t-test as well as a Welch test. On the comparison between the groups NH and RS, the tests statistics are $p = 0.053$ for the t-test with Bonferroni correction and $p = 0.001$ for the Welch test.

The reason for running both tests is that the valid application of Student's t-test might be questioned on the basis of the assumptions for the test. Usually, three assumptions have to be satisfied in order for Student's t-test to be applicable. First, the test groups have to be independently sampled. Second, the means of the dependent variable have to follow a normal distribution. And third, the populations being compared have to have equal variances. In our case, the requirements are only partially met. While our groups were independently sampled, the scores are clearly not normal distributed (see Figure 5.7). Furthermore, the variances between the groups differ (as indicated by Levene's test, $p = 0.001$ on groups NH and RS, $p = 0.016$ for all groups).

According to the central limit theorem [64], means samples being large enough are usually well-approximated by a normal distribution, even if the data are not normally distributed. Further adding that Slutsky's theorem implies that the distribution of the sample variance only has little effect on the distribution of the test statistic for large sample sizes, allows us to apply Student's t-test for large groups even in the absence of a normal distribution and equal variances. For our case, all groups have sample sizes >500 and can therefore safely be considered as "large", with the lower bound for "large" often being set between 30 and 100. For these reasons, we conducted Student's t-tests.

We additionally conducted pairwise Welch tests, because Student's t-test is prone to outliers. Welch's test further does not assume equal variances and has higher statistical power in this case [123].

Following this argumentation, we consider the difference in means between groups NH and RS to be significant on the basis of the Welch test with $p < 0.05$.

When focusing on particularly hard exercises, e.g., the three exercises requiring most working time, the distribution depicted in Figure 5.8 shows up.

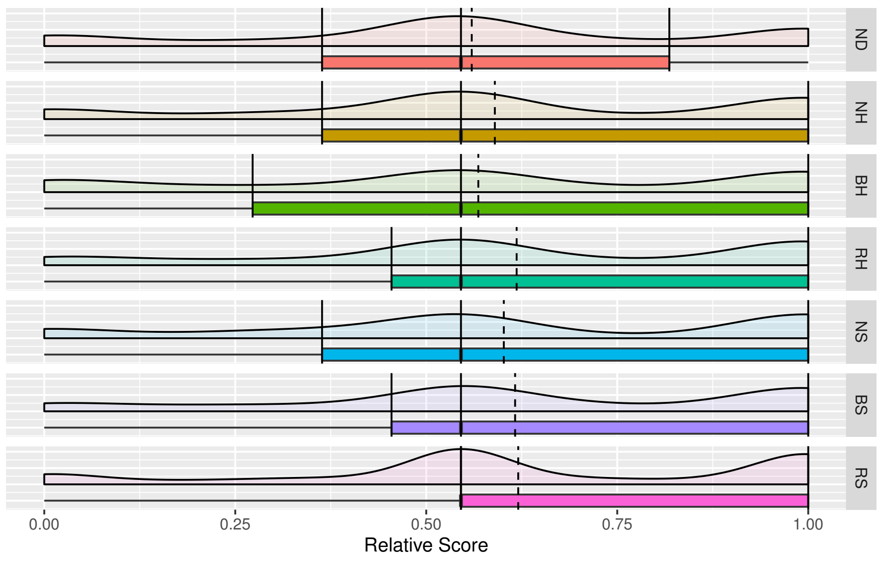
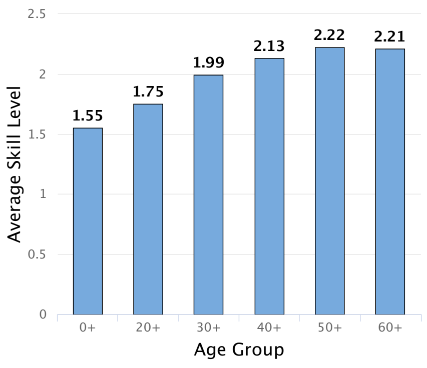


Figure 5.8: Combined density- and box-plots of the scores achieved by our experiment groups in the three exercises requiring most working time. The first quartile (first vertical bar) is affected by interventions.

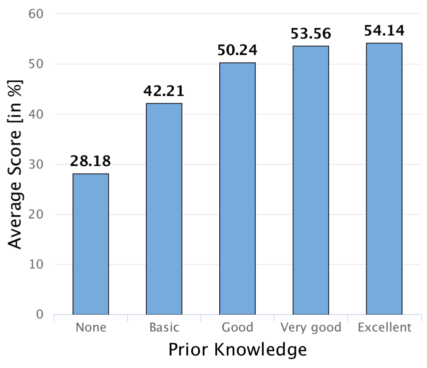
For the most difficult exercises, which also cause significantly more RFCs to be issued than simpler ones, also an effect on the weaker students becomes visible. The difference in means between groups ND and RH is significant (Welch test, $p = 0.001$). An additional visualization of another exercise subset can be found in the appendix.

Confounding Factor: Prior Skill Level

On the basis of the age and skill distributions presented in Subsection 5.2.4, we measured the underlying correlation between age and prior skill level. The resulting distribution is depicted in Figure 5.9a.



(a) Metrics: age and skill level.



(b) Metrics: skill level and achieved score.

Figure 5.9: Correlations between core metrics: age, skill level, and score.

The means of skill levels and the age groups show a strong Pearson correlation of $\rho = 0.95$. On an individual level per student, the age groups show only a very weak correlation with the skill levels ($\rho = 0.16$).

As can be seen in Figure 5.9b, also the prior skill level and the mean achieved course score show a high correlation ($\rho = 0.92$). Once more, the correlation on individual student level is very weak ($\rho = 0.16$).

We recorded student's prior skill levels via voluntary surveys as motivated in Section 3.2.1. The correlation of key metrics with these prior skill levels can be taken from Table 5.7.

Skill Level	**Average Score All Users**	**Average Score RFC Creators**	**Average Score Exam**	**Average Score RFC Creators Exam**
1	28.18	30.89	74.41	80.71
2	42.21	50.07	80.64	83.16
3	50.24	64.33	84.05	87.13
4	53.56	69.01	87.06	88.70
5	54.14	61.92	84.49	87.75
Total Avg.	46.99	56.23	83.23	83.58

Table 5.7: Achieved scores per skill level.

We see a constant trend of rising average scores per skill level up to skill level 4 (very good prior knowledge). Skill level 5 (expert) shows similar to slightly lower averages than level 5. These overall observations hold for all subgroups examined, including just students who requested help at least once (RFC Creators), just students who took the final exam (Exam), and students who requested help at least once and took the exam (RFC Creators Exam). The largest differences in average scores are visible between skill levels 1 and 2 in all subgroups. On average, RFC Creators reach higher scores than all other active users. Naturally, the scores of students who have taken the exam are considerably higher in general. But once more, also under this condition, the scores of students asking for help (RFC Creators Exam) turned out to be slightly higher than to the respective comparison group (Exam).

To consider and rule out potential biases caused by different skill levels in our experiment groups, we analyzed the skill distributions per experiment groups. The respective shares can be found in Table 5.8.

Skill Level	**ND**	**NH**	**BH**	**RH**	**NS**	**BS**	**RS**
1	6.60%	5.88%	5.26%	6.40%	5.91%	5.75%	5.56%
2	33.76%	34.56%	40.43%	36.26%	35.93%	33.75%	35.92%
3	33.76%	33.82%	30.38%	34.60%	29.79%	32.25%	32.16%
4	18.78%	18.14%	15.55%	16.82%	18.44%	20.50%	17.79%
5	7.11%	7.60%	8.37%	5.92%	9.93%	7.75%	8.57%
Avg.	2.86	2.87	2.81	2.80	2.91	2.91	2.88

Table 5.8: Average skill level and distribution of skill levels per experiment group. All experiment groups show a similar distribution, with nearly equal averages.

The distributions are nearly similar across all groups, also the average skills level per group do only vary at most ±2% around the overall average of 2.867. The results of our experiments thus were not systematically skewed or distorted by differing skill levels.

Request for Comments Accumulate on Hard Exercises

Figure 5.10 shows the distribution of Request for Comments on exercises covering different topics and learning targets. High request rates are visible for the topics “Concatenation”, “Classes, Methods, Attributes”, “Loops”, “Access Control” and “Collections”. The general decline of students over the runtime of the course is not reflected in the rates of requested RFCs. Additionally to the expected issues on loops, especially exercises combining several concepts caused a high rate of requests.

Commenting is Widely Distributed Among the Audience

Figure 5.11 shows the density function of comments per student. This visualizes the distribution of comments written over the individual students in the audience. The plot is separated into four differently colored areas, each representing a quantile of the students. This allows to infer that 75% of all commenting students created less than 5 comments per person for example.

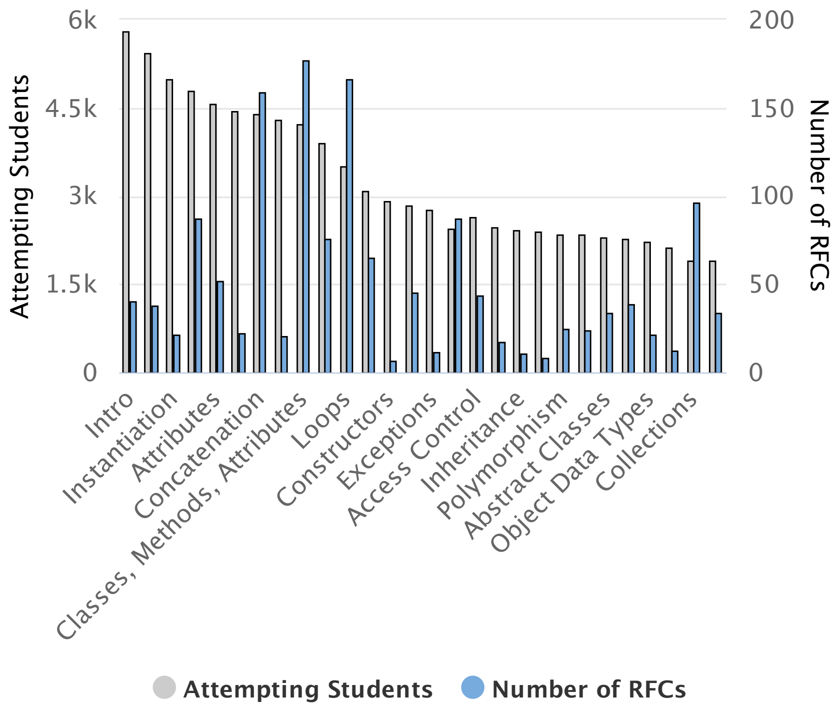


Figure 5.10: Distribution of Request for Comments on exercises of specific learning topics. The amount of RFCs on a topic does not correlate with the number of attempting students.

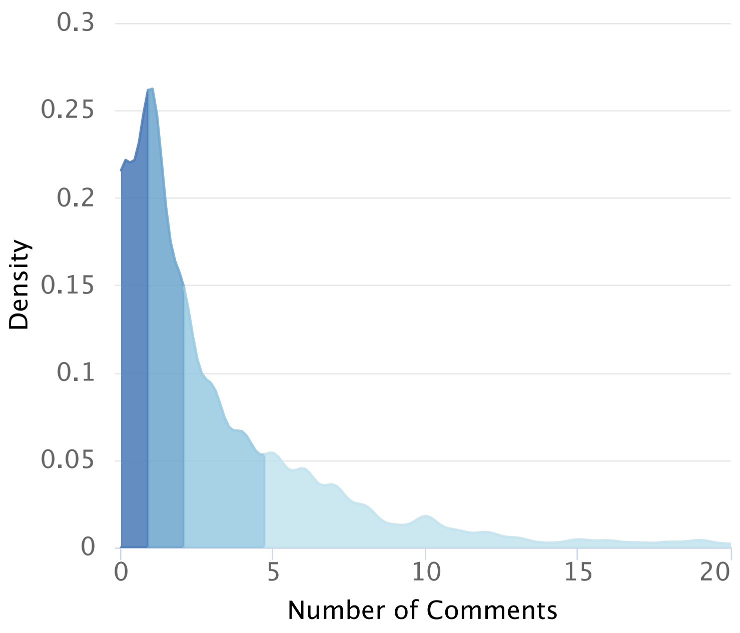


Figure 5.11: Density plot of created comments per student. Each colored section represents 25% of students. The majority of students writes less than five comments.

Hiding Request for Comments Reduces Future Outreach Attempts

We technically prevented to forward students to the Request for Comments of all "hidden" experiment groups. This effectively cuts the respective students from external feedback and affects their subsequent behavior. Table 5.9 shows the key metric which is affected by this measure, the average request for comments per requesting student. To support easier understanding of this metric, we also present the originating factors. We further added the resulting relative deltas compared to the corresponding baseline group, i.e., between groups NS and NH, between groups BS and BH, and between groups RS and RH.

Group	#RFCs	Total Students	Requesting Students	Share Req. Students	Avg. RFCs per Student	Relative Delta	Avg. RFCs per Req. Student	Relative Delta
NH	129	548	84	15.33%	0.235	-39,59%	1.54	-37.05%
BH	157	556	95	17.09%	0.282	-27,51%	1.62	-24.95%
RH	309	560	154	27.50%	0.552	-38,67%	2.00	-38.19%
NS	222	571	91	15.94%	0.389	-	2.39	-
BS	218	560	99	17.68%	0.389	-	2.15	-
RS	2016	2239	621	27.74%	0.900	-	3.18	-

Table 5.9: Effects of hiding RFCs. The average number of RFCs per student decreases by up to 40%. The share of requesting students is only affected by the issued interventions, with the RFC intervention groups resulting about 10 percentage points higher than the no intervention groups.

The share of students requesting help at least once (referred to as "requesting students" in the following) is between 15% and 18% for the groups NS, NH, BH, and BS. The two groups being targeted with RFC interventions show a share of requesting students around 27.5%. The condition whether the requests were hidden or not thus did not affect the share of requesting students. This is perfectly plausible since the effect of this limitation comes to play only after reaching out for help.

When comparing the average RFCs issued per student in the "shown" groups with their belonging "hidden" counterparts, it can be seen that the effect of hiding RFCs causes a decrease in average RFCs issued between 28% and 40%. The overall average is a decrease of 36%. Doing the same analysis on RFCs per requesting student, shows decreases between 25% and 38%, with an overall average of 35%.

These results are relatively stable, when doing the same analyses just on students who completed the course with a record of achievement (not shown in the table for the sake of simplicity). In this case, the overall average of decrease is 40% per student, respectively 43% per requesting student. These slightly higher numbers are reasonable, as the students reaching the record of achievement were longer in the course and thus the "shown" comparison groups had more opportunities to issue additional RFCs.

Just-in-Time Interventions Increase RFC Rates

When comparing the Request for Comments issued per student (see Table 5.9), the average within the "RFC intervention" groups results to 0.831 RFCs per student (weighted average from 0.552 for RH and 0.9 for RS), while the average in the "no intervention" groups is 0.314 RFCs per student (weighted average from 0.235 for NH and 0.389 for NS). This translates to an increase by 165% from no intervention to RFC intervention. Expressed otherwise, the increase factor is 2.65. This number is relatively stable, as becomes evident when comparing the ratios of just the groups for which the RFCs are shown to their peers, (i.e. NS and RS) instead of averaging over the "hidden" and "shown" groups (NH and NS, respectively RH and RS). The result when comparing just the "shown" groups is also an increase of 132%, comparing RS to NS.

The break interventions did not cause a major increase in Request for Comments, the measured increases compared to the "no intervention groups" are 7.12% for all active students (and 0% for just the respective show groups).

This finding collides with the learnings made in 2017 from the course Java 2017. Data originating from the 2017 course showed that break interventions caused an increase of RFCs from 0.6 RFCs per student without interventions to 0.8 RFCs per student receiving break interventions, which translates to an increase of 33% (*Anomaly 1*). Potential reasons for this difference and also the different base levels will be further reflected upon in the discussion section.

Receiving Answers Increases Full Score Ratio

Figure 5.12 shows the two major effects that are caused by receiving help. The average number of received comments per RFC is 3.2 for the experiment groups that did not have their requests hidden. Of all students who received comments, approximately 76% solved the exercise at hand within one hour. Within the group of students who did not receive answers, only 58% reached full score within one hour. This difference of ≈18 percentage points, caused by the comments received, represents an increase of 26% of students having solved their exercise with full score. Concerning the required working times to solve an exercise, only a small difference becomes visible in this graph.

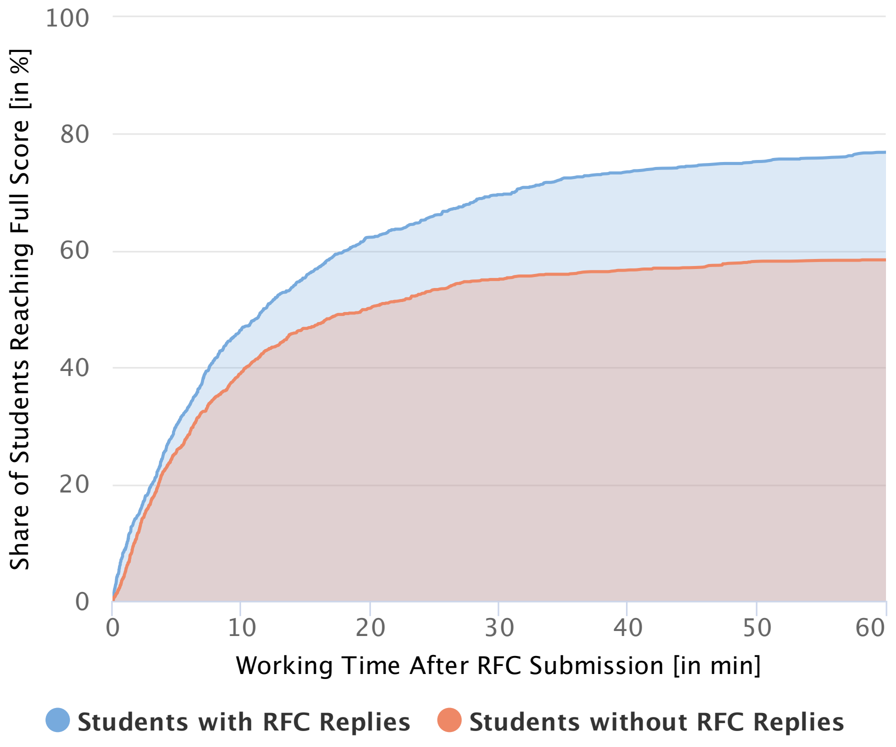


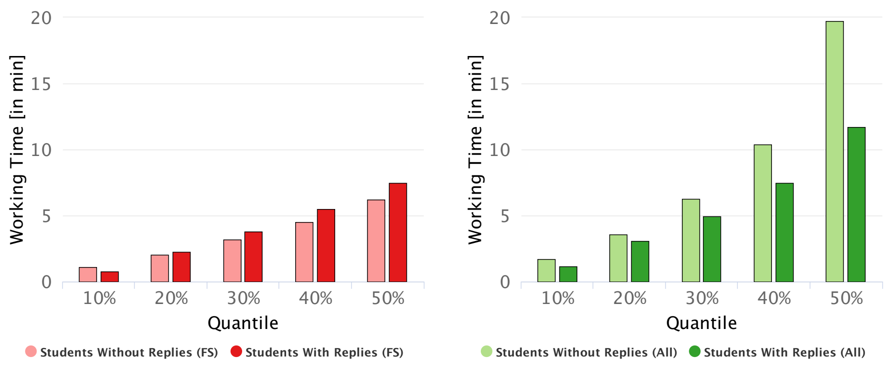
Figure 5.12: Effects of receiving help on required working times and success rates of students. The required working time is measured starting at the moment of issuing a Request for Comment.

Receiving Answers Reduces Required Time

A closer analysis of changes in the time students required to solve the exercises yields two additional results, visible in Figures 5.13a and 5.13b. A calculation of the first five 10% quantiles (10% fastest students, 20% fastest, ...), considering just students who fully solved the respective exercise, shows that students who did not receive any help were often faster (Figure 5.13a). This calculation is based on sets of data. The first data set solely contains the working times of the ≈60% of students who did not receive help and succeeded, while the compared second data set contains working times of the ≈80% of students who received help and finished their exercises. The quantiles of the respective groups thus represent a different absolute number of students having reached full scores. Therefore, a higher required working time until half of each group finished with full score is plausible for the group which includes several weaker students who only succeeded with external help and likely needed longer.

This obvious but naive approach is not able to reflect the effects towards students who improved their performance due to external help, but most likely would have also succeeded without help, just requiring more working time. In order to show this effect, we refrain from excluding students who gave up on their exercise, but keep them within their respective groups. To enable a coherent analysis without leading to skewed results, all students who did not solve their exercise are placed at the end of their group concerning the required time, with an assumed time of three hours. The assumed time does not affect the quantiles, as long as we just regard student populations faster than the assumed time. From Figure 5.12 we infer that we can safely analyze our student populations up to the 50% mark. Their required time is, in any case, lower than 60 minutes and therefore also lower than our assumed time of three hours.

Thus widening the scope of our analysis to all students that requested help, regardless of whether they fully solved the exercise at hand, the perspective changes. Students who received help were able to solve their issues faster, with an increasing delta for increasing quantiles. 50% of the students who received help were able to solve their exercise within an additional working time of 12 minutes after their request. In comparison, within the group of students who did not receive any help, nearly 20 minutes of subsequent working time were required until 50% of the group solved their exercise. This means that the group of students receiving help reached the 50% quantile more than 40% faster.



(a) Only students who achieved full score **(b)** All students

Figure 5.13: Detailed analysis of students' required working time after requesting help. Quantiles are compared in 10% steps up to 50% for students who received help and those who did not receive any help.

RFC Interventions Partially Countervail the Effects of Hiding RFCs

The combined effects of hiding RFCs and just-in-time interventions can be observed if one takes the group NS as a baseline. This group, having its RFCs shown to users and not being nudged by any just-in-time interventions, represents the default case. The average numbers of RFCs per student and per requesting student presented in Table 5.10 show how the positive and negative influences interact.

Group	**Avg. RFCs per Student**	**Relative Delta**	**Avg. RFCs per Req. Student**	**Relative Delta**
NS	0.389	Baseline	2.39	Baseline
NH	0.235	-39.45%	1.54	-37.05%
BH	0.282	-27.37%	1.62	-32.26%
RH	0.552	+41.92%	2.00	-17.75%
BS	0.389	+0.13%	2.15	-9.74%
RS	0.900	+131.59%	3.18	+33.07%

Table 5.10: Combined effects of hiding RFCs and just-in-time interventions. RFC interventions partially countervail the effects of hiding, visible for group RH.

Regarding the average number of RFCs issued per student, NH and BH show a negative impact between approximately 30% and 40%. The combination of RFC interventions and hiding the requests resulted in an increase of around 42% RFCs. Group BS behaves similar to group NS. Lastly, group RS shows a tremendous increase of approximately 132% over the baseline group NS.

Concerning RFCs per requesting student, the “hidden” groups show an 18% to 37% lower amount of RFCs compared to NS. Within the “shown” groups, the deltas range between -10% and +33%.

Request for Comments are Particularly Popular With Beginners

When analyzing the shares of students creating RFCs or commenting on other's questions per skill level, we encounter the metrics shown in Table 5.11.

Skill Level	**RFC Creating**	**Commenting**	**Avg. RFCs**	**Avg. Comments**
1	33.18%	21.20%	0.79	0.52
2	28.90%	37.30%	1.01	1.46
3	21.38%	41.77%	0.46	1.89
4	21.13%	43.13%	0.45	1.91
5	17.31%	44.55%	0.32	1.86

Table 5.11: RFC and commenting metrics per skill level. Beginners issue more RFCs, experts write more comments.

We see a consistent trend of declining chances for a student to create at least one RFC with increasing self-stated skill level. On the opposite side, increasing skill levels increase the probability that the student will comment on at least one request for comment issued by a fellow student. Correspondingly, the mean number of issued Request for Comments declines with increasing skill level, while the number of contributed comments increases with increasing skill level. The minor deviations from these trends for skill levels 1 and 5 most likely derive from the comparably low student populations within the respective groups, resulting in a higher variance of values and thus presumably not as accurate means.

To also analyze the effects of just-in-time interventions and RFC hiding separately by skill level, we aggregated the average number of RFCs issued by skill level as well as experiment group. Results are shown in Table 5.12.

Skill Level	**ND**	**NH**	**BH**	**RH**	**NS**	**BS**	**RS**
1	0.00	0.29	0.32	0.50	0.40	0.57	0.81
2	0.00	0.38	0.31	0.68	0.29	0.31	0.68
3	0.00	0.22	0.23	0.37	0.39	0.27	0.50
4	0.00	0.18	0.35	0.37	0.29	0.39	0.41
5	0.00	0.03	0.23	0.28	0.26	0.42	0.32
Avg.	0.00	0.25	0.28	0.48	0.33	0.34	0.55

Table 5.12: Average Request for Comments per student, grouped by skill level and experiment group. Outliers $\geq$10 RFCs per student have been removed. Interventions especially affect lower skill levels one and two.

The average number of RFCs is affected by the issued interventions, especially for the lower skill levels. This result becomes clearly visible when combining the "shown" and "hidden" groups to reduce complexity and then plotting the results per intervention type (see Figure 5.14). Combining the groups does not skew the results significantly, as the effect caused by hiding RFCs is nearly independent of the issued interventions as shown in Subsection 5.4.1. To improve expressivity, we removed students issuing ≥10 RFCs as outliers, the unaltered distribution can be found in the appendix.

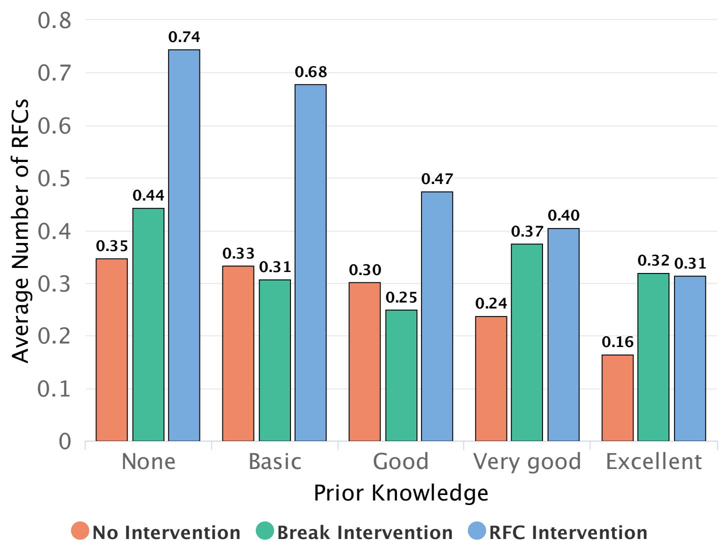


Figure 5.14: Average amount of RFCs per combined experiment group and skill level per student. Outliers with ≥10 RFCs per student removed.

Going into greater detail, we have a look at the share of students requesting help at least once and break down the numbers by experiment groups as well as students' skill levels.

	Group							
Skill Level	ND	NH	BH	RH	NS	BS	RS	**Avg.**
1	0	25.00	22.73	29.63	20.00	43.48	39.58	33.18
2	0	21.99	18.93	40.52	17.11	16.30	35.97	28.90
3	0	14.49	15.75	17.81	22.22	13.95	26.85	21.38
4	0	13.51	20.00	23.94	17.95	20.73	23.78	21.13
5	0	03.22	20.00	20.00	14.29	22.58	18.92	17.31

Table 5.13: Share of students requesting help per skill level and experiment group (values in percent). Beginners are more likely to request help than experts.

Table 5.13 shows that there is a partly consistent trend towards fewer students issuing RFCs on higher skill levels. The positive and negative deviations showing a discrepancy to the general trend (e.g. for group NS) indicate that the analyzed subpopulations are too small to draw reliable conclusions on individual group levels. For skill levels one and five, the number of students ranges between 1 and 10 for all experiment groups except the larger sized RS group.

	Group							
Skill Level	ND	NH	BH	RH	NS	BS	RS	**Avg.**
1	0	1.17	1.40	3.00	2.00	3.30	2.37	2.38
2	0	1.71	1.63	2.16	3.31	2.36	4.49	3.48
3	0	1.50	1.45	2.08	2.71	1.94	2.29	2.16
4	0	1.30	1.77	1.53	1.64	1.88	2.63	2.15
5	0	1.00	1.14	1.40	1.83	1.86	2.11	1.83

Table 5.14: RFCs per RFC requesting student, grouped by skill level and experiment group. Hiding RFCs reduces the amount of RFCs issued.

Within Table 5.14 we provide the amounts of issued RFCs per requesting student. When regarding the group NS as a baseline, it becomes apparent that all "hidden" groups suffer from reduced numbers, along all skill levels. The issued RFC interventions in group RH dampen the effect, but cannot fully mitigate it. The positive effect of RFC interventions also shows in group RS. We cannot outline an effect of our interventions on specific skill levels. Once again, the partly strong positive and negative effects showing a discrepancy to the general trends (in this case, e.g., for group BS in skill levels one to three) are caused by small subpopulations.

	Group							
Skill Level	ND	NH	BH	RH	NS	BS	RS	**Avg.**
1	26.92	25.00	13.63	14.81	20.00	21.74	23.96	21.20
2	33.83	39.00	37.28	34.64	38.16	42.22	36.29	37.30
3	35.33	43.47	41.73	39.04	50.00	45.74	39.28	41.77
4	29.72	41.89	49.23	45.07	46.15	36.58	42.67	43.13
5	21.43	38.71	48.57	44.00	52.38	45.16	42.57	44.55

Table 5.15: Share of commenting students per skill level and experiment group (values in percent).

Table 5.15 shows that the experiment group had no visible effect on the commenting behavior in general, with one exception. Students in the group ND, especially the ones with higher prior skill levels, wrote less comments than the other groups.

We additionally evaluated the results just for students who submitted the exam, without any further interesting findings.

Written Questions Increase Likelihood to Receive Help

When being asked for help, the care the asking person has put into the question might have an influence on the willingness to answer. Also, expressing one's thoughts can start new lines of thoughts which might lead to higher success rates. For these reasons, we further analyzed the potential effects a written question has towards the commenters as well as the requester. The results are shown in Table 5.16.

Visibility	**Expressivity**	**#RFCs**	**Commented**	**Commented by Others**	**Solved**	**Full Score**	
hide	without question	115	3.48%	0%	18.26%	47.83%	53.03%
	with question	496	6.05%	0%	29.03%	54.23%	
show	without question	590	75.42%	74.58%	31.02%	61.02%	66.25%
	with question	1908	83.18%	81.87%	45.55%	67.87%	

Table 5.16: Effects of written questions on commenting behavior and exercise completion.

A written question improves the likelihood of receiving an answer, reaching full score and (of course) marking it as correct. Further looking into the details, a written question leads to a higher share of students reaching full score, within both visibility categories, with an average improvement of approximately 12% (6.5 percentage points).

Abstracting from the written question, RFCs shown to other students lead to students reaching full score by 66.25% , while hidden RFCs only lead to success in 53.03% of all cases. The effect of showing or hiding RFCs causes a delta of 13.22 percentage points and is thus larger than the effect of a written question.

RFC Interventions Increase Share of RFCs Without Questions

Another interesting aspect is the effect of the just-in-time interventions towards the expressivity of the questions and the care struggling students put into them.

Interventions	**Without Question**
none	11.75%
break	14.62%
RFC	25.68%

Table 5.17: Share of Request for Comments without a written question per intervention group. Especially RFC interventions increase this share.

Table 5.17 shows that interventions cause more RFCs without a question. This is true especially for RFC interventions. Concerning the expressivity of the questions, if a written question was supplied, we did not notice any differences based on the issued interventions.

Students Value Request for Comments Feature

Student valuation of the request for comments feature was assessed via a survey at the course's end. The answer rate is thus significantly lower, leading to larger deviations. An overview of the questions and the associated answers can be found in the appendix.

The following tables show the answers of our students on the question "The possibility to request comments directly in CodeOcean when solving the practical programming exercises...", which could only be answered with one of the offered options. Students were grouped by their skill level (Table 5.18), respectively their experiment group (Table 5.19).

Skill Level	Students	Not Necessary	Lacked Visibility	Not Helpful	Mediocre	Helpful
0	28	60.71	10.71	0.00	3.57	25.00
1	268	48.88	9.33	4.85	2.24	34.70
2	252	62.30	6.35	3.57	1.98	25.79
3	188	68.09	6.91	4.26	0.53	20.21
4	74	72.97	2.70	1.35	0.00	22.97

Table 5.18: Students' valuation of RFCs per skill level, distribution in percent. Valuation is higher for low skill levels.

For skill level 0 only a very low number of students answered, therefore we abstain from drawing conclusions for this group. Valuation expressed via the option "helpful" is highest for skill level 1, with generally decreasing value for increasing skill levels. Overall, most students (50% to 70%) regarded the feature as not necessary for them, with increasing values for higher skill levels. Only a few students regarded the feature as not helpful or mediocre, independent of their skill level.

RFC Interventions Amplify Perceived Helpfulness

Concerning the question of visibility, a separation by experiment group, expressed in Table 5.19 yields further insights.

Group	Students	Not Necessary	Lacked Visibility	Not Helpful	Mediocre	Helpful
ND	73	54.79	17.81	2.74	1.37	23.29
NH	92	61.96	6.52	6.52	1.09	23.91
BH	87	51.72	14.94	6.90	2.30	24.14
RH	95	66.32	9.47	10.53	0.00	13.68
NS	93	62.37	7.53	2.15	0.00	27.96
BS	81	59.26	4.94	2.47	2.47	30.86
RS	376	58.78	3.99	2.13	2.13	32.98

Table 5.19: Valuation of RFCs per experiment group, distribution in percent. Compared to ND, helpfulness is higher for RS and lower for RH.

While RFC interventions could have decreased the “lack of visibility” for group RS compared to group NS, this conclusion cannot be drawn for the comparison between RH and NH. Also, BH shows a high value for the lack of visibility, which cannot be explained by the experiment conditions and thus hints towards reliability issues caused by the lower answer rates. A coherent difference can, however, be seen when focusing on the “helpful” column. In general, the hide groups report lower helpfulness than the show groups. When further analyzing the RFC intervention groups, the results indicate that the interventions decreased the perceived helpfulness when no answers were received (group RH) and increased the perceived helpfulness when answers were received (group RS).

Commenting Improves Knowledge

When asked about their opinion on commenting on other students' questions and code, participants' answered with the options summarized in Table 5.20.

Skill Level	**Enjoyed Helping**	**Learned Something**	**Felt Disturbed**	**Never Encountered**	**Not Able to Help**	**Want More**
1	39.13	47.83	26.09	17.39	30.43	21.74
2	24.37	45.80	27.73	27.31	35.71	15.55
3	32.69	42.79	23.08	27.40	29.33	13.46
4	30.32	41.29	26.45	27.74	30.97	14.84
5	39.39	39.39	24.24	16.67	27.27	13.64
Avg.	29.80	44.05	26.01	26.14	31.37	14.25

Table 5.20: Students' valuation of the commenting feature, distribution in percent. Predominant answers are that students learned something and enjoyed helping.

Of all students who have answered the question, ≈30% enjoyed helping others. More than 40% stated that they learned something when commenting on the code of a fellow learner. The share of students stating this decreases with increasing skill levels. About 25% felt disturbed by the requests, another 25% stated they never encountered the commenting page. About one third further stated that they were not able to help the asking student, as the question at hand was too difficult. Again, this option shows a slightly decreasing trend with increasing skill level. About 15% further expressed they would have answered more questions of fellow students if that option was offered.

Actionable Interventions are Preferred

The experiences students made with the just-in-time interventions are summarized in Table 5.21. Again, students were able to chose one of the offered options to end the statement “The interventions that encouraged me to ask for help or take a break...” to best represent their opinion. Either they stated that they were not influenced at all, that they never witnessed any interventions, that they felt bugged, or that they had the impression that the popups actually helped. Percentages add up to 100% per row.

Group	No Influence	Never Occurred	Bugged	Helped
ND	15.71	67.14	1.43	15.71
NH	24.44	45.56	1.11	28.89
BH	49.43	11.49	26.44	12.64
RH	48.91	16.30	19.57	15.22
NS	22.58	52.69	2.15	22.58
BS	34.57	11.11	39.51	14.81
RS	42.70	12.95	17.91	26.45

Table 5.21: Students' experience of just-in-time interventions, distribution in percent.

For our analysis, the main focus lies on the column "helped". As can be seen for groups BH and RH, respectively BS and RS, request for comment interventions were perceived as more helpful than break interventions. Break interventions, on the other hand, bugged students significantly more than RFC interventions. The relatively high numbers for groups ND, NH, and NS once again show that survey data has to be interpreted with caution. These groups did not receive any interventions, as is also indicated by the high shares of students stating that the popups never appeared. These results are in line with the results we made with our Java course in 2017 [160].

Basic Struggle Detection is Practical

With regard to the timings, students' responses can be found in Table 5.22.

Group	Never	Too Early	On Time	Too Late
ND	73.53	0.00	10.29	16.18
NH	60.23	0.00	18.18	21.59
BH	14.29	48.81	30.95	5.95
RH	23.33	45.56	24.44	6.67
NS	62.37	1.08	17.20	19.35
BS	7.41	58.02	24.69	9.88
RS	15.15	38.57	34.99	11.29

Table 5.22: Perceived timing of just-in-time interventions, distribution in percent. Interventions appeared too early ($\approx$45%) or on time ($\approx$30%).

Just considering students who actually received just-in-time intervention, $\approx$45% stated that the popups appeared too early, $\approx$30% had the impression the popups occurred right on time, and $\approx$10% stated they appeared either too late or never. There are no consistent differences between RFC and break interventions visible, which reflects the actual situation with no differences in the struggle detection approach.

5.4.2 Bonus Exercises

Results show five key findings with regards to bonus exercises.

First, students attempted the ungraded bonus exercises at a lower rate than the last graded exercises of the same week (week 1 shows a decrease by 23%, week 2 by 16%, and the outro-week by 54% resp. 60%). The differences are statistically significant (Welch Two Sample t-test $t = 3.1$, $p = 0.043$) . Also the completion rate was reduced ($t = 2.3$, $p = 0.025$) . The average working times were not affected ($t = 0.30$, $p = 0.76$). The presence of statistically significant differences for attempts and completion rates collides with our findings from the data collected in 2017, where no differences were detected (*Anomaly 2*).

Second, when analyzing the differences in weaknesses specific to skill groups, certain topics seem to be harder than others for students stating low skill levels. With increasing skill levels, the distribution of weaknesses becomes more balanced (Figure 5.15 shows that exemplary for course week 1).

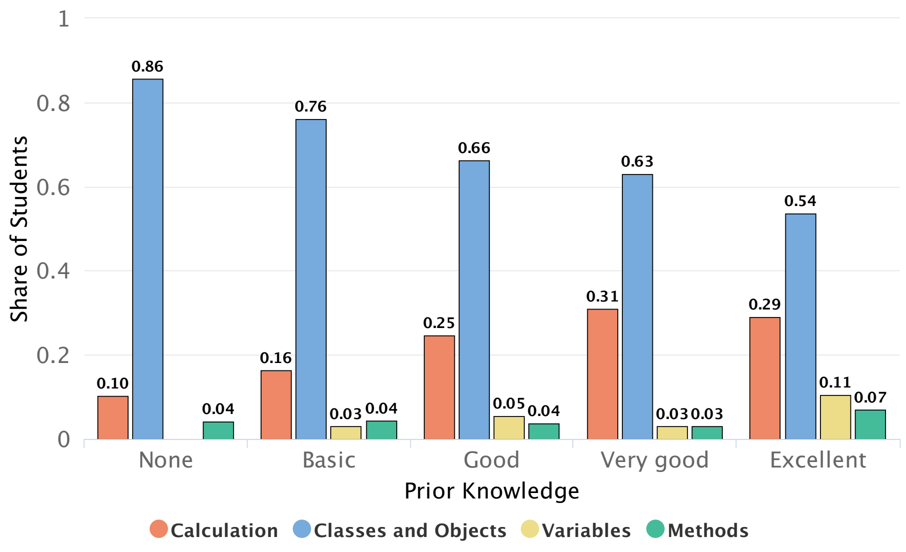


Figure 5.15: Students' weakest topics of week 1 by prior knowledge. Particularly beginners struggle with classes and objects. With increasing prior knowledge, the share of the topic calculation rises and the distribution becomes more balanced in general.

Third, students who were assigned to an exercise by our algorithm solved the respective exercise 10% faster in average than students who were assigned to the same exercise at random. This is in line with our findings from the Java 2017 course, but again without statistical significance ($t = 0.62$, $p = 0.53$).

Fourth, student perceptions, as shown in Table 5.23 ($N = 992$), did not vary significantly based on whether the bonus exercises were recommended, picked at random, or dummy ones (all $p > 0.30$).

Fifth, the bonus exercises were mostly received as helpful and fitting the specific weaknesses (54%). About 28% perceived the bonus exercises as good but not specifically helpful, 6% as too difficult, and 11% as superfluous in general. The absence of varying perceptions between dummy exercises and the other exercises conflicts with our results collected in the Java 2017 course (*Anomaly 3*, see appendix for results from the Java 2017 course).

Group	**Exercise**	**Too Difficult**	**Superfluous**	**OK**	**Helpful**
1	dummy	4.97%	11.05%	25.97%	58.01%
2	random	3.93%	11.24%	30.90%	53.93%
3	recommended	7.98%	10.38%	27.94%	53.69%

Table 5.23: Students' perception of bonus exercises per experiment group. Perceptions did not vary between experiment groups.

5.4.3 Video Tutoring

Video tutoring turned out to be not applicable on large scale, as also outlined in a previous publication [163]. The survey from our students in the 2018 course ($N = 992$) showed a similar answer distributions as encountered in previous courses. The main aspects are that about one-third of our audience state that they will decline the offering due to privacy concerns. Technical shortcomings exclude ≈15% of the participants from such an offering. Regarding the potential benefits, feedback from tutors or course instructors is the most popular option, with ≈45% of students expressing this as a desired use case. Substantial differences based on prior skill levels do not appear. A complete overview of the respective survey results can be found in the appendix.

5.4.4 Automated Anomaly Detection

The automated anomaly detection was continuously running on the exercise set of the Java 2018 course. The visualization of the mean required working times to finish each exercise presented in Figure 5.16 shows that four exercises exceeded the anomaly threshold of twice the group mean. The order of the bars reflects the order of the exercises in the course. Exercises exceeding the threshold of ≈1020 seconds are two exercises in the middle of the second course week (dealing with loops and conditional) and two exercises at course end (giving an outlook on advanced data structures like ArrayLists and HashMaps). Over the course runtime, the algorithm detected five anomalies: the four exercises just mentioned, and an exercise dealing with "Classes, Methods and Attributes Combined", showing a mean working time of 1006 seconds in the graph. Mails were sent for all five anomalies, resulting in meaningful feedback in all cases. The four exercises clearly exceeding the threshold were deemed as appropriate in their current form. The issues students were encountering originated on coping with the concepts introduced and were therefore regarded as inevitable or tolerable by the course instructors. The exercise on "Classes, Methods and Attributes Combined" however suffered from an obstacle originating from the structure of the exercise. The course instructors thus subsequently added an additional hint, helping to prevent students from struggling with undesired instantiation issues. As a result, the average required working time slightly dropped and stabilized at the 1006 seconds mark shortly below the threshold.

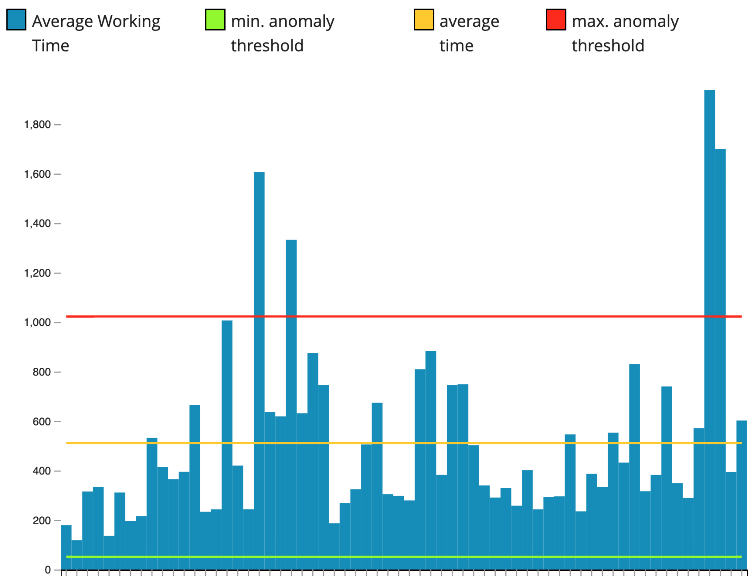


Figure 5.16: Screenshot of the anomaly detection for all Java 2018 exercises. Four average working times of exercises are exceeding the upper anomaly threshold. An exercise in the left third was improved and now lies closely below the threshold.

5.5 Discussion

The findings presented in the results section will be shortly summarized in the following and evaluated in the context of the formulated hypotheses. Regarding our first hypothesis, that request for comments improve students' learning success, data show different effects and improvements. Request for comments increase students' average scores by up to 13%, the likelihood of achieving a Confirmation of Participation by 12%, and the likelihood of achieving a Record of Achievement by 17%. Concerning the distribution of issued RFCs over all exercises, more difficult exercises showed high request rates in particular. As visible in the resulting box-plots of score distributions, especially in difficult exercises, predestinated to cause more issues and misconceptions, weaker students benefitted from expressing their problem and receiving help.

For our experiments on the effects of hiding the requests, data show that actually receiving an answer increases the share of students reaching full score on an exercise by 26%. The required working time to reach full score decreases, the mark of 50% of the students having solved their exercise is reached 40% faster on average.

Further notably and plausible findings are that request for comments especially lead to an improvement for students within lower skill levels (levels "no prior knowledge" and "basic prior knowledge"), and that students within higher skill levels showed a higher tendency to comment on open requests. Requests with a written question were more likely to be commented on and also showed a 12% higher chance to lead to a fully solved exercise later on. Further noteworthy is that the improved chance caused by a RFC to result in a fully solved exercise is independent of the fact whether the request received any comments. Our explanation for this is the effect of so-called "rubber duck debugging"[66], describing the positive influence of externalizing one's thought and thereby potentially overcoming one's hurdles on one's own.

Summing up, the answer to our first hypothesis is a distinct "yes". Request for comments improve learning success, indicated by multiple metrics.

The subquestion, whether hiding request for comments restrains the positive effects, also has to be answered positively. Hiding requests lowered the total amount of request for comments issued in the respective experiment group by 35%. The decline in total RFCs shows that student interaction and a positive experience is necessary in order to establish a recurring help-seeking behavior and thus to reach the best effect. However, already expressing one's thoughts in a question improved students' scores, thus being a positive intervention. Receiving a helpful answer adds further benefit, but a large share of improvement was also noticed for our experiment groups with "hidden" RFCs.

This explanation is further supported by the results concerning our second hypothesis, the effects caused by just-in-time interventions. Just-in-time interventions increase the rate of students requesting help up to 165%. Thus, the gains achieved via RFCs are further amplified by the RFC just-in-time interventions suggesting to reach out for help.

Additionally to the aforementioned "rubber duck effect", in this case, also the effect of interruption comes into play. Albeit we do not force students to add a description to their problem, the mere interruption and prompt to describe the issue presumably has an impact on the students and is likely to trigger reflection as an activity of self-regulation.

Similar to request for comments, the just-in-time interventions showed a stronger effect on students having a lower prior skill level than on already more advanced students.

The discrepancy between the published results on the basis of our course "Java 2017" [160] and the data presented in this thesis, labeled "*anomaly1*", can be explained with a different framing. The seemingly counterintuitive results, that just-in-time interventions that motivate students to take a break caused a significant increase in RFCs in 2017 and no increase in 2018, become comprehensible when focusing on the experiment setup. In the 2017 course iteration, request for comments were initially introduced. Aiming for as much usage and resulting data, request for comments were actively recommended to all students within a video lecture. The teaching team advocated all students, that whenever they are stuck, they should reach out for help. Contrary, in 2018, the video introducing the programming environment was recorded with the feature disabled and the possibility not mentioned at all. This explains the different behavior of students receiving break interventions. While 2017 students might have been reminded of the advocated RFC feature, 2018 students might just have dismissed the intervention.

Detailed analysis, including an examination of the combination of hiding requests while exposing students to just-in-time interventions, showed that just-in-time interventions motivating students to reach out for help can partially countervail the effects of hiding the request. While the individual student is likely still disappointed about not receiving help, the interventions nudge more students to try the functionality at least once. Also, the interventions motivated some students not having received answers to try another time, resulting in an overall higher RFC per student metric.

From the survey answers we further learned that just-in-time interventions disturbed the students. This can be seen as a positive effect to trigger reflection, however, it also caused annoyance. The group receiving RFC interventions answered to a lower share that they felt annoyed, therefore we conclude that actionable interventions are preferred.

Summing up, the second hypothesis, whether just-in-time interventions have an effect on students' behavior, is thus also answered with "yes".

For the recommendation of bonus exercises, only some of the findings we made in 2018 are in line with the results from the 2017 iteration. Our algorithm assigned relatively more beginners to the bonus exercise dealing with classes and objects than experts. Based on the argumentation, that the exercises assigned to experts should trend towards an equal distribution as experts have already mastered all topics and thus show no weaknesses, the decline of the distribution shares for the exercise on classes and objects reassures us of the applicability of the concept. Considering the working times, we see that students who struggled with a specific concept, on average solved the related bonus exercise faster than students who were randomly assigned to this exercise. This indicates a learning effect, however, the findings are not significant. We encountered two differences, anomaly 2, which showed in different behavior regarding attempts and completion, and anomaly 3, which showed in a different perception of students. Anomaly 2 can be explained with a change we made in scoring, granting no points in 2018, while we granted some in 2017. This likely reduced the number of attempting students, as well as their finishing rate. Anomaly 3, the difference in perception concerning the helpfulness and "fitting" of the exercises, cannot be plausibly explained on the basis of the data we collected.

The third hypothesis, "tailored bonus exercises help students overcome their weaknesses", therefore cannot be answered confidently on the basis of our result. Data are suggesting an effect, however, its size is too small to serve as the foundation for a reliable answer. Students' value all offered bonus exercises, even the intentionally "unhelpful" dummy exercises.

Video tutoring turned out to be not applicable to our audience. While demand for video tutoring was voiced over several years, privacy issues, as well as technical deficiency, prevail. Within ongoing attempts over several courses, the number of successful tutoring sessions remained below ten. As described in a previous publication [163], the focus was shifted from "intervening" on students towards a focus on "understanding" students, granting instructors direct feedback and the otherwise missing "look over one's shoulder".

Automatic anomaly detection has proven its usefulness within the Java 2018 course. Due to having no direct comparison with the unaltered version, we cannot further conclude on the efficiency.

5.6 Threats to Validity

The most prominent threat to the validity of our findings are skewed experiment groups. The relatively strong increase in key metrics over our control group, also for the group that just had the possibility to reach out for help, but without any interventions or actually receiving help, was our main concern. For this reason, we ruled out all obvious confounding factors, e.g., skill levels, academic background, age, or place of residence, without finding any anomalies. The sizes of our experiment groups, with the smallest group size being 547 active students, should further rule out other effects caused by individual outliers for general observations. For experiments requiring additional filtering and aggregation (e.g. by prior skill levels), we removed outliers as described.

The general setup includes seven experiment groups, each being comprised of more than 500 active students. The tendencies found within the data are coherent and plausible. We are therefore confident that the effects we uncovered are of causal nature. While the actual effect sizes might be questioned, the existence of the measured effects is to be regarded as verified.

Data originating from course surveys are likely affected by some form of response bias. The positive tendency can be explained by a persistent response bias caused by at least three different reasons. First, the audience remaining at course end is most probably higher skilled than the base audience at course start. Second, students answering the course-end survey completed the entire course, showing endurance as well as skill. Consequently, most have developed some pride in their achievements, leading to a positive attitude. Third, many students want to support the researchers and instructors, possibly also as a kind of “thank you” for offering the MOOC free of charge. The impact of students’ motivation to please the researcher was specifically outlined with the question design of the question focusing on the helpfulness of the offered bonus exercises and could thus be mitigated.

5.7 Further Findings

Given the carefully prepared data set, several findings were made that are not directly related to the previously formulated research questions. In the following, we will present the findings, backed by the corresponding data, but will refrain from drawing premature conclusions, as we can only make educated guesses without further independent experiments. The number of available datasets is comparably low for the subgroups of female students, impeding to further draw reliable conclusions. In the following, to ensure transparency concerning reliability, the data labels printed above the resulting bars show the number of datasets included instead of the reached y-value.

5.7.1 Females Create Twice as Much RFCs Than Males

When comparing the average amount of RFCs created per student, data shows a considerable difference between males (0.337) and females (0.782), depicted in Figure 5.17. On average, female students create 2.3 times as much RFCs as males.

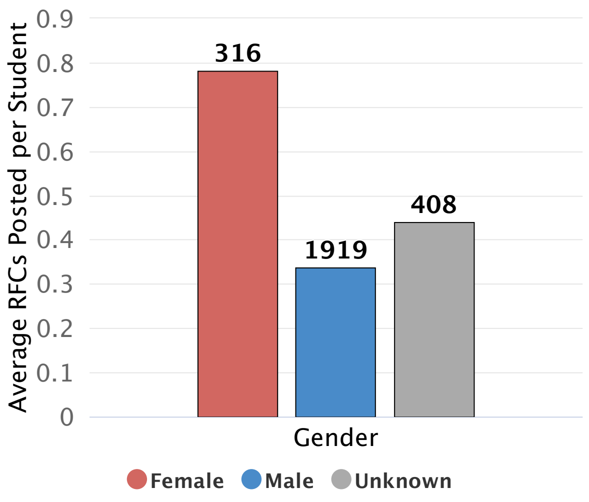


Figure 5.17: Average number of Request for Comments issued per gender.

When further grouping the data by age, no coherent effect of age becomes visible (see Figure 5.18). The difference in requested RFCs shows for all age groups for which we have sufficient data.

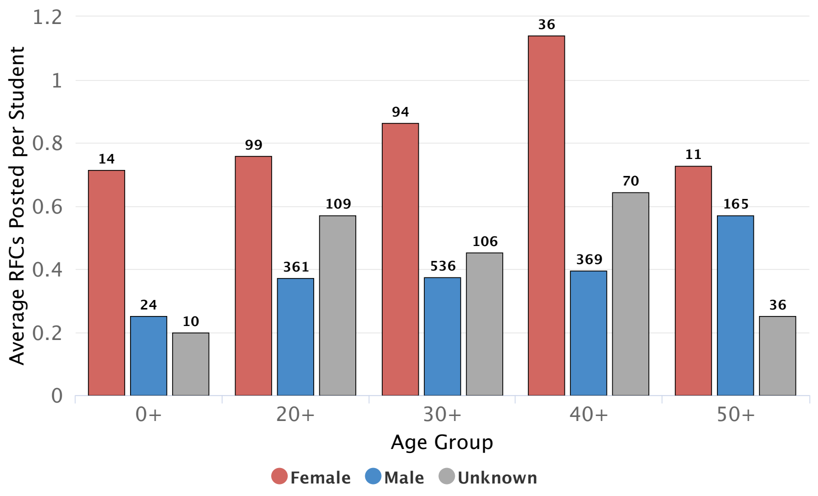


Figure 5.18: Average number of Request for Comments issued per gender and age. Differences in the amount of issued RFCs show for all age groups for which we have sufficient data.

For the age group below 20, the ratio of RFCs created by females relative to the number of RFCs created by males is 2.86, for the age group between 20 and 30 (20+) it is 2.04, for the age group 30+ it is 2.30, for age group 40+ it is 2.88, and for the age group 50+ the ratio is 1.28.

5.7.2 Average Skill Levels are 20% Lower for Females

With the difference in RFC rates at hand, we also compared skill levels for males and females. From our previous experiments (see Figure 5.14), we know that the self-assessed skill level negatively correlates with the number of RFCs created. Therefore, this analysis might be able to explain the difference. Results are shown in Figure 5.19.

What can be drawn from Figure 5.19 is that female students self-assess their average individual skill between 14.2% to 31.7% lower than male students. The averages are consistently lower among all age groups, with an overall average difference of 20%. This situation has also recently been presented in the Stack-Overflow Community Survey 2019[27].

Based on the effect sizes in our previous evaluation, the differences in RFC behavior cannot be explained by differences in self-assessed skill levels.

[27] see `https://insights.stackoverflow.com/survey/2019`

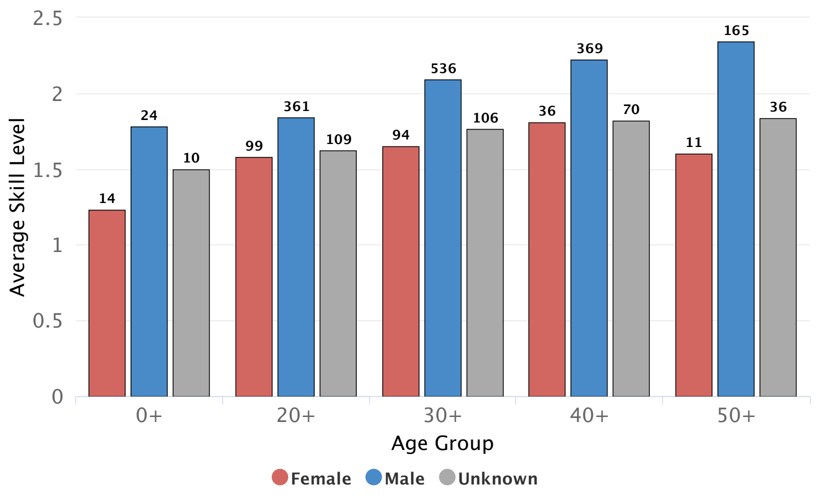
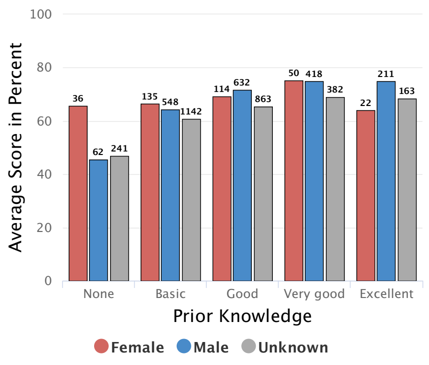


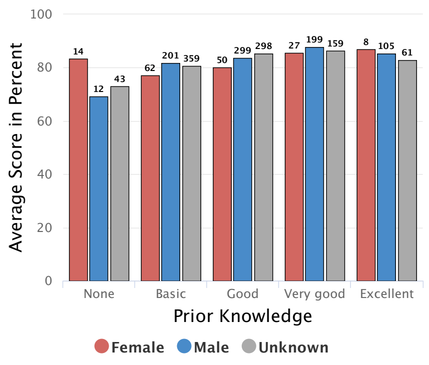
Figure 5.19: Average skill level per gender and age group. The average self-assigned skill level is lower for the female group compared to the male and unknown groups within all age ranges.

Figure 5.19 further confirms a previous finding: the average skill level correlates with students' age, independent of gender. Average skill levels increase by about 0.15 absolute points per 10 years.

Figures 5.20a and 5.20b additionally show that there is no difference in achieved scores between males and females. This holds for the overall average as well as the subgroup of students who have finished the course.



(a) All students



(b) Only finishing students

Figure 5.20: Average scores of students per gender and skill level. Average scores increase with skill prior knowledge. No significant differences are visible between genders.

6 Future Work

Alongside the numerous insights that were gathered, rise plenty of subsequent questions, research directions, and ideas for further improvement. Following the structure of this thesis, we will present these future directions separated into the topics “understanding”, “intervening”, and “adapting”.

To better understand students’ learning experience and potential causes of struggle we propose two main approaches. First, supplementing students’ voluntary self-assessment with means for comparatively inaccurate, but automatic estimation of prior skills. As discussed in our publication on optimal programming exercises [159], student performance classification is a tough research topic itself, because there are many variables to consider and assessments before the course starts are impractical. We argue that we can use our knowledge model as a first step towards this topic. Further incorporating typing patterns and writing speed of students into our knowledge model, allows to estimate a general “coding readiness” level, independent of specific programming concepts to determine their skill as discussed by Leinonen et al. [93]. While students solve the introductory exercises, the programming platform could record additional information to supplement the self-assessment without requiring additional effort from students. Specific patterns such as curly brackets or particular common expressions like “`i++`” and their typing speeds allow to deduce students’ general programming skill levels to a certain degree. Second, we propose to improve struggle detection, for example by automatically detecting bursts of errors, frequent retries with little to no source code changes, or back-and-forth editing.

The field of interventions can further be improved by mitigating the influence of instructor set variables. In our research, such instructor set variables include the chosen percentile to intervene on, the topic weights used in the student knowledge model, and exercise difficulty ratings. In order to mitigate the influence, additional experiments are necessary to first assess the impact of the specific variables. Subsequently, optimal values can be derived from the performance of student groups within an A/B-testing setup. To ensure the required student numbers to enable reasoning of statistical relevance, these experiments have to be conducted in future course iterations.

Further technical improvements, such as additional context-sensitive interventions, are a promising approach to better support struggling students on individual level. Specific hints, references to cheat sheets, and appropriately detailed feedback could be provided based on encountered errors, students' current programming context within the exercise (e.g. if they are failing to implement a loop header), and the assumed skill level. Thus offering a personalized support system could provide fine-grained step by step guidance for beginners, without disturbing advanced students. Concerning Request for Comments, a deeper analysis of the influence of gender is promising for future insights. A survey-based approach is recommended in this regard, as it offers more open feedback and does not rely on gathering a large subgroup to uncover further details and connections between previously unknown factors.

Another idea is to cluster Request for Comments in order to subsequently manually provide a foundation for problem-centric and meaningful forum discussions. In this way, struggles encountered by some students can not only serve as teaching material for the students directly involved by requesting or providing help, but the opportunity to learn would be opened up to the general audience.

In order to further improve just-in-time interventions, we propose to develop means to detect students' progress. Our current approach depending on percentiles cannot separate actually struggling students from ones just taking it slow. Distinguishing those groups based on the presence or absence of progress, e.g., by detecting working program structures as a positive sign, or by singling out back-and-forth editing as a negative one, will help to reduce the caused annoyance of our interventions.

In the area of bonus exercises and exercise recommendation, the predominant need is to supply a larger pool of exercises. Increasing potential choice, both in terms of concepts conveyed as well as content difficulty, is likely to increase students' perceived benefit. Especially providing high performing students with harder exercises deems to be promising. Challenging those students, providing potential for further growth, and keeping them engaged ensures the availability of advanced knowledge in the community as a vital component for peer feedback.

Our efforts in finding indicators for the need of adapting course content can be enhanced by taking results from the improved struggle detection into account. Especially the accumulation of certain error types can serve as a hint towards issues on exercises. While `ArrayIndexOutOfBoundsExeceptions` have to be expected in the first exercises combining loops and arrays, high counts of these exceptions in later exercises may indicate issues with the exercise description or the supplied template. The general approach we take on programming exercises further can be generalized for other content types. Such efforts exist with video event analytics and quality check alerts on openHPI, however these alerts come without automated gathering of additional feedback or proposed solutions.

General future research directions promising worthwhile results are the integration of professional tooling into online programming education, further expanding collaboration options, and enhancing automated program assessment. First prototypes were build to connect CodeOcean with the Eclipse IDE, in order to enable more advanced workflows such as debugging and refactoring without losing the online functionality for assessment and requesting help. While the CodeOcean assessment API and the Eclipse plugin are already functional, appropriate experimentation and analyses are still pending.

Additionally, integrating means to support pair programming is likely to improve suitability for blended-learning settings given in schools. Adding capabilities for static code analysis further offers to provide early feedback on code style and program design. The idea to showcase different student's solutions in order to fuel discussions about expressivity and understandability of code points in a similar direction.

Summing up, apart of the obvious directions to revalidate findings and improve reliability by reiterating the experiments under different surrounding conditions, the presented concepts have shown effectiveness and provide plenty of options for further research.

7

Related Work

This chapter gives an overview of related research and approaches concerning the improvement of online programming education. In particular, we highlight the field of large scale tutoring and the area of educational interventions. Where applicable, we further demarcate our approaches from those followed by fellow researchers. Parts of Subsection 7.1.1 have been published in [163], respectively of Sections7.2 and 7.3 in [160].

7.1 Scaling Tutoring

Helping students to overcome their misconceptions and struggles is usually addressed by individual tutoring in traditional classes. Mainly two aspects qualify tutoring as superior over other approaches such as automated answer crawling or improving expressivity of error messages: individual focus and human flexibility.

Tutoring is a social activity between two or more individuals, thus requiring an appropriate situation for collaboration. Such a situation usually involves an issue to be solved, as well as an established relation between tutor and the student requesting help. Given the absence of a shared physical location in online education, also the virtual environment largely influences the possibilities for collaboration and thus tutoring effectiveness.

Altebarmakian and Alterman present three different usage scenarios of online collaborative learning tools to improve cohesion and exemplify their proposed design heuristics [4]. They use a synchronous chat, an asynchronous blog, and a combination of both. Our approaches of video conferencing and asynchronous commenting differ concerning the presented heuristics (e.g. organization of the conversation and feeling of presence), but satisfy them in general. We thus see our and their work aligning well and complementing each other.

7.1.1 Remote and Video Tutoring

The research of Philip Guo is tightly related to our approaches. In the following, we will shortly describe different tools he created in order to improve programming education and remote tutoring. Namely, these are Python Tutor, Codeopticon, Codechella, Codemotion, and CodePilot.

As the underlying foundation for other approaches Guo implemented Python Tutor, a tool to run python programs online and visualize the steps of program execution [52]. Based on this foundation, Guo conducted research in different directions to improve online programming education.

The direction to give individual feedback usually leads to different forms of tutoring, with live remote tutoring being the most common one. Live remote tutoring means that a tutor supports one or more students synchronously while they are solving their exercise. The other direction, further enhancing and building on error messages as well as stack traces, usually leads to automated and asynchronous solutions. This means that after a program run finished unsuccessfully, different processing steps are carried out to supply further information on the basis of the occurred errors. With the processing steps usually requiring less than three seconds to be performed, these approaches are perceived as "live" or "synchronous" for the students, albeit they are technically asynchronous.

Guo followed the first direction and approached the scenario of live remote tutoring with a software called "Codeopticon" [53]. The aim is to enable instructors to monitor and support multiple students in live sessions at the same time. Within the MOOC setting, the major issues concerning tutoring are the usually missing opportunity "to look over the shoulder" and the mismatch of workload and numbers of students in comparison to the number of tutors. Fundamentally, Guo's solution offers tutors a live view onto the current state of active students' code. Tutors can also interact with students potentially requiring help via chat. Codeopticon, as a novel approach, further contributes a scaling mechanism on top of that. It shows tutors a grid of multiple tiles, each containing a single student's solution, updated in real-time. This way, tutors monitor about 15 students at once on one screen, allowing for multiple advantages: (1) downtime is minimized for tutors, who no longer have to wait for students' reactions on their hints, but can attend to other students, (2) tutors can passively monitor students potentially too shy to ask for help, and (3) they can proactively approach students they think are in need of help without intimidating them too much. Given the virtual setting, it further allows tutors to (4) set their own pace of work and (5) allows them to regulate overly demanding participants by non-attendance. The ability to monitor 15 students at once of course still does not suffice to support everyone in the MOOC scenario, therefore Codeopticon shuffles students' solutions in and out, depending on their activity. Instructors may pin code of interest and additionally are further visually supported by automatic highlighting of errors, execution positions during debugging, and ordinary code edits such as character additions or deletions. Results show that tutors can assist about three students at once, and that this kind of tutoring is accepted by students. Shortcomings outlined by Guo incorporate the limitation towards text based feedback not allowing for helpful diagrams when explaining general concepts, the danger of favoring quantity of tutoring over quality, and flaws in attention management by shortcomings in the shuffling algorithm. On top of that resides the conceptual shortcoming that this solution still does not scale automatically with the audience. Codeopticon thus serves as a helpful step to gradually balance the mismatch between students and instructors. However, the point in time when too few tutors will face too many students will still occur, just delayed at a higher number of enrollments.

Codeopticon relates to our approach of video tutoring as well as to our idea of Request for Comments promoted via just-in-time interventions. Video tutoring conceptionally shares many advantages and disadvantages with Guo's idea. On the negative side, video tutoring does not scale at all, therefore being inferior to Codeopticon in this regard. On the positive side, video tutoring establishes a more personal and feature rich relation between the tutor and the student, supporting better explanations by enabling to actually talk instead of having to type words. Furthermore, our approach also allows drawing and sharing of schematics, which was mentioned to be especially helpful when facing conceptional misunderstandings.

Guo et al. subsequently extended the offered features to also allow for synchronous tutoring with a shared view of the source code, mouse course positions, as well as the program output. The resulting tool was named Codechella [57]. Based on Mozilla's TogetherJS library[28] it attempts to resemble an in-person tutoring setting as close as possible. However, they decided for a chat-based communication rather than a video conference due to the greater simplicity and the observed easier communication behavior for coding-related content. This approach scales with the audience, but opposed to our setting, it cannot ensure the required knowledge to be available for the exercise at hand. While Python Tutor has a relatively large user-base, the synchronous nature also limits the available pool of tutors.

Widening the scope, Warner and Guo developed a prototypical IDE to allow multiple programmers to coordinate in real time while coding, debugging, managing issues, and versioning their files [171]. It shares the main motivation with our video tutoring approach. The main difference between them is that their prototype aims at a broader setting including issue tracking and versioning. Coincidentally, they also called their implementation CodePilot. In contrast to our approach, they did not introduce video conferencing, but widened their scope of applicability to more tools (e.g. the issue tracker of GitHub) and more situations (e.g. collaboration in productive settings). With a focus on the educational aspect, the conclusions they have drawn comply with ours, outlining that tutoring eased the understanding of particularly hard topics (they focused on versioning with Git).

Similar to Codeopticon by Guo et al., Glassman et al. developed Overcode [46] to support tutors in scaling by grasping the details of different student solutions through clustering of similar code. The approach aims at asynchronous situations, e.g., giving students feedback on submissions to empower them to commence later on. Building on different approaches, including canonicalization, transformations based on the abstract syntax tree, and program synthesis, OverCode builds so called "stacks" from similar solutions to programming exercises. Each stack shows tutors a "cleaned" representation of the various solution it represents. The stacks can then be scrolled, filtered or be further merged

[28] see `http://togetherjs.com`

by so called “rewrite rules”, effectively giving users the option to improve the automated clustering. Similar to our approaches, the overall aim is to uncover shortcomings and misconceptions in students’ understanding, as well as to find potentially valuable mistakes in order to pinpoint common pitfalls. The current implementation works on Python code, however the authors state that their pipeline can be generalized for other programming languages.

A limitation of this approach is that it only works on syntactically correct solutions, which also “have already been marked correct by an autograder”. It therefore focuses on the more advanced steps taken when learning programming, i.e., judging solutions based on their style, expressivity and complexity. The most common cases occurring within introductory programming courses, having a program that is not compiling, or having a program that compiles but is unable to satisfy the tests of the autograder, are not tackled. With their aim on supporting tutors in their grading work, this constraint does not limit the applicability, as solutions that are syntactically incomplete or are not satisfying the required unit tests, are usually graded with zero points (or the share of the passed unit tests).

For our use case, supporting struggling students, this constraint however massively reduces the potential benefits. Given the aforementioned scenario, that most struggling students will provide submissions that are failing the unit tests or are not compiling at all, we focus on approaches that are able to cope with faulty or even broken code. The two main directions that satisfy these requirements, are giving individual human feedback on submissions (1), and building on error messages resulting from broken code (2).

Concerning video tutoring, there also have been former attempts to utilize video conferencing in xMOOCs, e.g., Collab Spaces in openHPI or Talkabout in Coursera [85, 150]. While these attempts highlight the importance of collaboration and the positive effects of connecting students, they currently have a number of drawbacks based on their proprietary foundation. Their dependence on Google Hangouts and requires an additional account and prevents a deeper integration into the course platform to enhance grouping quality or provide better feedback. This also prevents recordability and might lead to legal issues, as data sovereignty is also not given and content is shared with third parties. openHPI’s Collab Spaces focus on creating purposeful groups. Three different kinds of groups are usually distinguishable: study groups, that progress through the course together and are connected by external factors (language, location, age, employer), topic focused groups that evolve around a certain (often times specific or especially demanding) topic, and teams. Teams are formed when a certain task has to be solved in cooperation.

Joseph and McKinsey surveyed the adoption of remote pair programming in 2013, and came to the conclusion that organization is one of the major issues [105]. Also, they encountered problems with participants just wanting a “free ride”, joining remote programming sessions with the primary goal to copy solutions for the exercises. For production usage a distinguished tool to form groups and plan shared time slots is therefore recommended. In their experiments with such a tool, Talkabout, Kulkarni et al. found that students collaborating in diverse discussions were significantly more likely to answer quizzes and score higher on exams [85]. They also underline that a pedagogical concept that accompanies the group discussions is important. Depending on the course type and the expected learning results, certain strategies can increase sharing of self-references or encourage students to re-evaluate their own opinion.

Omitting explanations of all agenda items encourages students to ask questions about them. When testing a very rigid agenda, Kulkarni et al. found that the discussions were less motivating and that students were less inclined to meet the same group again. While Coursera, the platform that the respective courses were conducted on, uses an open-source, web based development environment for some of their programming classes, there is no deeper integration of this into Talkabout. Talkabout also did not try the concept of video tutoring in a programming related class. Additionally, it became apparent that just putting people together does not lead to effective progress. Staubitz et al. concluded that an elaborated team composition increases learning success [150].

7.1.2 Coding Tutorials

In addition to the approaches directly dealing with tutoring, Chen and Guo also developed Improv, a tool to better combine teaching efforts via live coding with common teaching via slide presentations [22]. This improvement promises to be especially fitting for online education efforts. By combining slide presentations with live coding, the benefits of slide presentations, e.g., a better structure, can be intertwined with the flexibility and authentic nature of live coding. Context switches are minimized, while the dynamic setting enabled via runnable code examples can be preserved. Furthermore, so called "code waypoints" enable instructors to create predetermined states they want to reach during their presentations. In case the live programming goes off track, e.g., due to mistakes, code waypoints act as a safety net and allow to either glimpse at the correct solution while typing or to jump to the desired next state directly.

Another approach to better utilize videos featuring live programming is Codemotion by Khandwala and Guo [72]. Using a computer vision algorithm, they extract presented source code from the videos in order to enable use cases surpassing the mere visual presentation. Based on the extracted code, also the editing steps can be traced. This allows to annotate the underlying video parts with the respective progress being made and the concepts being explained. It further enables advanced usage scenarios such as inline code editing for embedded exercises, an improved search, and video skimming. Since we have supplied all source code presented in our MOOCs as accompanied downloads and offering dedicated exercises for training, the benefit for this approach is limited in our case. Nonetheless, further annotation for improved navigation will still provide additional benefits.

With Codecast, Sharrock et al. created a solution to ease the creation of interactive coding tutorials [141]. Allowing individuals to record the steps taken during program creation within an online IDE, while simultaneously recording audio from the programmer, simplifies the overall effort to create coding tutorials.

While not directly connected to the improvement of students struggling with specific programming exercises, all described research projects in this subsection are likely to reduce struggle caused by weak educational material. We thus see all of the mentioned tools as complementary to our approaches.

7.2 Interventions and Struggle Detection

Interventions

Davis et al. found that individualistic promotional intervention messages which motivate users to post in the forum improved scores for highly educated students [34]. They express the need for a system, that is also (or especially) suitable to target students with a lower prior knowledge. Our approach, using just-in-time interventions promoting Request for Comments delivers that.

The findings made by Carini et al. [19] and Yang et al. [180] support the claim that course-related discussions of students can have a positive influence on student learning results. As students cannot be approached individually and directly, Chaturvedi et al. [21] and Chandrasekaran et al. [20] propose to use a machine learning approach to detect threads in forums which need to be addressed by the instructors. Agrawal et al. [1] also try to reduce the work for instructors by identifying confusion in forum threads and automatically suggesting a ranked list of potential one-minute-resolution videos. However, this type of intervention also requires students to ask questions in the forums. Since students often do not reach out for help, being it because they do not realize that they need help [2] or because they are too shy, instructors cannot help them directly in most cases.

Especially weaker students hesitate to reach out for help in forums according to Onah et al. [114], who also mention that there is a lack of research on instructorless, personal interventions within MOOCs. Therefore, researchers try to automatically detect students who are at risk of dropping out and intervene on them in order to motivate them. Whitehill et al. [174] used their dropout predictions to survey students for feedback about why these students left the course. They noticed that the surveys motivated students to come back sooner than those students who received no survey. Mailing specific subgroups of a course with targeted content may also increase the effect of an intervention [80, 128, 162].

The aforementioned studies show that small interventions, like sending emails to students, can have positive impact on students' behavior. Oftentimes, the effect of interventions might however also be non-existent, or at least not verifiable, as shown in a review by Kizilcec and Brooks [74]. This is especially likely if the interventions are either not suitable or far too late to help students who struggle with exercises and thus face demotivation. When a rather generic intervention reaches the students days later and thus can no longer help them with the actual problem they were facing, it is unlikely that they will come back. Pre-emptive interventions, such as plan making interventions [182], affirmation interventions [76] and self-regulation interventions [75] are suited to prevent undesired behavior before negative consequences come to effect. In line with our findings, just-in-time interventions are therefore considered superior to actions which are performed at a much later point in time.

Struggle Detection

While learning, all students will likely experience struggle with one of their exercises at some time. As previously outlined, this is not necessarily harmful, as light struggle, often resulting in time-boxed trail and error, promotes learning and improves retention of the knowledge. Given the setting of in-class education, teachers may intervene on students struggling too hard, as they express their struggles through raising their hand and questions, or negative emotions such as anger or resignation.

Students struggling with content in MOOCs will show the same reactions, with the difference that instructors will never receive the direct, visual clues caused by negative emotions. The online equivalent to raising a hand and asking a question is opening up a thread in the course forum. However, research has already shown that students are reluctant to use the forum, as only about 5% of all students use the forum [65], albeit forum usage correlates with a better course performance, which is visible in our own data as well as the literature [24, 71]. In order to enable automated just-in-time interventions a capable struggle detection based on students' platform interaction is necessary.

In 2017, Drosos et al. proposed HappyFace, a survey application used on Python Tutor [36]. It collects students' experienced emotions via their clicks on one of five different expression options of smileys. The smileys expressions range from very happy, happy, over neutral to unhappy and very unhappy. Optional free-text answers were possible in order to describe the reason for the experienced feeling (e.g. "I finally understand the code" or "My output is wrong"). Whenever a student submits a response, the current status of the accompanied code submission is analyzed. Based on the abstract syntax tree (AST), features of syntactical correctness, complexity and style are extracted and related to the surveyed emotion. They found that even with a relatively low response rate of 2.36% (from 101,044 visitors), they could identify features of code that are related to frustration. Especially syntax errors and usage of niche features, such as global variables or `isinstance` calls were found to likely induce high frustration.

Sharma et al. noticed different programming strategies leading students' to success in programming exercises and claim to be able to identify low performers early [139]. Based on their metrics, including semantic-less measures (e.g. the assessment frequency) and code-based measure (e.g. growth in program size) they thus also perform struggle detection. Their study was based on four exercises solved by ≈600 computer science-students. While performed on a smaller scale with more homogeneous participants, their results comply with ours and affirm further potential for increasing intervention precision based on additional metrics as suggested within the future work chapter.

Several recent studies further examined the influence of a higher age [54], English as a non-native language in programming [55], and the dedication and usage intention of programming skills of students [170]. These factors all apply to our programming MOOCs to differing extent. The majority of our students are non-native English speakers and we also found in our data that the average age of MOOC students is higher than the average age of university students. With our introductory courses, we further attracted a high number of so-called "conversational programmers", who will be involved in programming related activities, but will not be actively coding themselves. Their main motivation to learn programming is to improve their understanding of the field and communication with their coworkers. All three studies outline the importance of interaction with tutors and peers to support learning success, especially for those students facing a higher risk of quitting the course. The learnings from these studies may also further outline potential metrics improving struggle detection, however we purposely abstained from integrating demographic data into our algorithms.

7.3 Bonus Exercises and Recommender Systems

Recommender systems are often used to improve the learning experience of students in MOOCs. Bauman et al. [9] identified knowledge gaps in student knowledge and proposed an algorithm to recommend remedial learning material they crawled from the Internet. To predict the topics for which students have knowledge gaps they use the number of points a student received on a quiz and derive a score for each topic associated with the quiz.

Our approach shares many concepts with the aforementioned. In order to determine deficits in content understanding, we also rely on annotation of learning material and quizzes (in our case, programming exercises). In contrast to quizzes, programming exercises in introductory MOOCs are often pursued until a full score is achieved as they can be reattempted as many times as students wish. Therefore we added the metric how long students needed to complete the exercise compared to other students.

Recommender systems have also been used to recommend learning items such as websites, articles, books or exercises in [108, 137]. Particular interesting for our work is the research from Michlík et al. [108], as they deal with recommending exercises to students in an e-learning context. They annotated the learning objects with concepts and created a vector-space model to map the knowledge of the students. After students solve an exercise, they are asked to provide feedback which is then used to calculate the knowledge model using computer adaptive testing (CAT). From Michlík et al. we adapted the idea of concept tagging and creating a vector-space knowledge model. Since their feedback system relies on user feedback and not on automated tests, we developed a new knowledge model suited for our specific use case and the increased user base within a MOOC.

EduRank, developed by Segal et al. [137], uses collaborative filtering in order to create a personalized difficulty ranking for exercises. However, their algorithm works best if students have only partial overlap of accessed exercises. This is not given in MOOCs because all students follow the same order of exercises, making it difficult to use in this domain.

8 Conclusion

MOOCs are an effective means to convey knowledge to a broad audience, overcoming social and territorial barriers. For the field of information technology in general and programming in particular, practical application of presented concepts via exercises is valued by students and crucial for a lasting learning effect. Extended struggle, often caused by high difficulty due to abstract concepts and cryptic feedback, however, impedes learning success, especially for beginners. Counteracting measures, such as tutoring, employed in in-place education, lack applicability in the online domain. Additional social factors, passively promoting learning and establishing a proven learning environment, cannot be transferred to the field of e-learning. The lack of direct social recognition of progress, adaption to problems, and individual feedback results in a social gap, resulting in reduced impact and higher dropout rates.

We conducted research to narrow this social gap induced by current technical shortcomings and developed suitable approaches as well as implementations.

Within this thesis, we presented a model encompassing actions for instructors to *understand* issues within their MOOC, *adapt* course material accordingly, and to automatically *intervene* on students. We focused on interventions, promising the greatest leverage, being applied on an individual level based on situational factors and driven by the student's specific progress metrics.

The data gathered over the course of several MOOCs support that interventions in online courses are suitable to bring struggling students into the zone of proximal development and increase their learning success. Of the applied interventions, especially the request for comments, in combination with just-in-time interventions, have proven to be effective and beneficial approaches. Course scores improved by up to 13%, with certification rates having improved by up to 17%. Detailed analysis showed that especially students with lower prior knowledge profited from the interventions, thus further improving education permeability.

Tailored bonus exercises are valued by students. Despite the fact that significant effects of bonus exercises on learning success could not be verified, we were able to demonstrate that the overall approach was applicable to the given scenario and adopted by the students. Data further indicate learning improvements promoted by limited struggle, thus offering additional insights into learning processes from the researchers' perspective.

All main experiments have been revalidated, showing coherent tendencies on large audiences, further assuring credibility and generalizability.

Video tutoring did not result in the desired outcome of broad adaption as a means of intervention. Missing availability of technical requirements on the student side, as well as privacy concerns, hindered the widespread usage of the developed prototype. The concept was therefore refined to support instructors' understanding by unveiling a student's learning progress and residing issues.

The adaption approach to automatically outline and subsequently improve weak course material through anomaly detection has proven to be effective.

Our developed approaches, their resulting technical implementations, as well as the gathered data provide a rich foundation for ongoing research. The programming platform CodeOcean is available as open-source software and has proven to support MOOC scale workloads. The courses developed in the progress of this research attracted large audiences from Europe respectively the whole world. The courses, including all complementary educational material, remain available for everyone in German as well as in English. Up to today, they receive constant interest from motivated individuals but also from teachers and school classes, adopting the content into their own teaching. Our goal to improve programming education in MOOCs is therefore not only fully achieved but has even extended into physical classrooms.

Beyond the scope of programming education, the concepts we developed are generalizable to other areas, e.g, exercises in social sciences, and provide a promising opportunity for e-learning in general.

Revisiting Greek history, in which education was first formalized, brings us to a quote of Epictetus. He noted: "Only the educated are free". Despite being a former slave, he did not refer to materialistic constraints and the question of freedom as self-determination, but the boundaries of his mind and the option to question current beliefs in order to, e.g., improve society and better understand the processes of nature. Regardless of the times we live in, successful societies depend on well-educated citizens, prepared to tackle the challenges facing them. MOOCs are a modern answer to the challenges of modern times - with our research and results, we are confident and proud to have contributed novel knowledge to an essential achievement of humanity: open education.

9 Appendix

9.1 Permission for Reuse of Publications

REUSE (ACM) Authors can reuse any portion of their own work in a new work of their own (and no fee is expected) as long as a citation and DOI pointer to the Version of Record in the ACM Digital Library are included.
Contributing complete papers to any edited collection of reprints for which the author is not the editor, requires permission and usually a republication fee.
Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. Authors can use any portion of their own work in presentations and in the classroom (and no fee is expected).

REUSE (IEEE) In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Hasso Plattner Institute's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/right_link.html` to learn how to obtain a License from RightsLink.

9.2 List of Published Results

Results of this dissertation have been published in the following papers:

Effects of Automated Interventions in Programming Assignments: Evidence from a Field Experiment. [160]
Ralf Teusner, Thomas Hille, Thomas Staubitz
ACM Learning@Scale (2018), DOI: 10.1145/3231644.3231650

What Stays in Mind? - Retention Rates in Programming MOOCs. [161]
Ralf Teusner, Christoph Matthies, Thomas Staubitz
IEEE Frontiers in Education (2018), DOI: 10.1109/FIE.2018.8658890

On the Impact of Programming Exercise Descriptions - Effects of Programming Exercise Descriptions to Scores and Working Times. [158]
Ralf Teusner, Thomas Hille
IEEE Learning With MOOCS (2018), DOI: 10.1109/LWMOOCS.2018.8534676

Video Conferencing as a Peephole to MOOC Participants. [163]
Ralf Teusner, Nicholas Wittstruck, Thomas Staubitz
IEEE TALE (2017)

Aspects on Finding the Optimal Practical Programming Exercise for MOOCs. [159]
Ralf Teusner, Thomas Hille, Christiane Hagedorn
IEEE Frontiers in Education (2017), DOI: 10.1109/FIE.2017.8190587

Taking Informed Action on Student Activity in MOOCs. [162]
Ralf Teusner, Kai-Adrian Rollmann, Jan Renz
ACM Learning@Scale (2017), DOI: 10.1145/3051457.3053971

CodeOcean - A Versatile Platform for Practical Programming Exercises in Online Environments. [146]
Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz, Christoph Meinel
IEEE EDUCON (2016), DOI: 10.1109/EDUCON.2016.7474573

9.3 Additional Figures

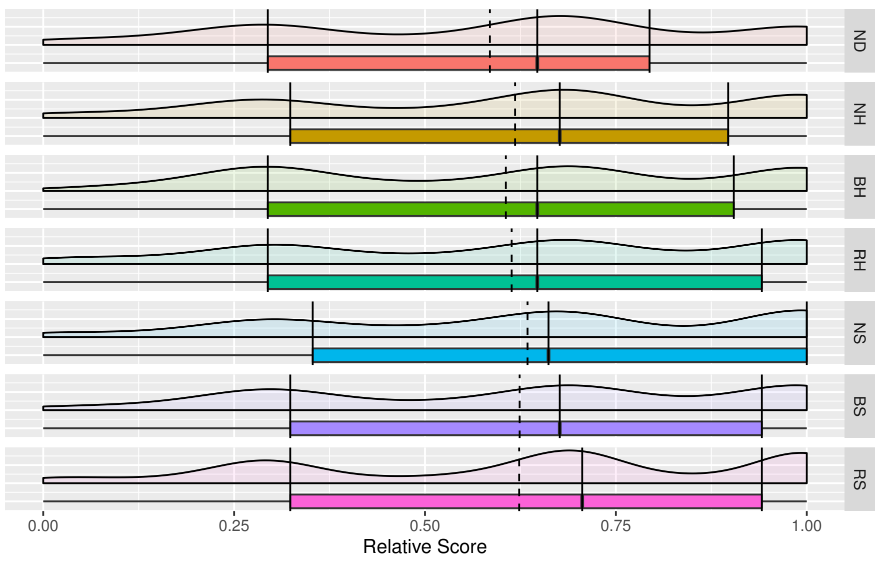


Figure 9.1: Combined density- and box-plots of the scores achieved in the five exercises being successfully finished least often.

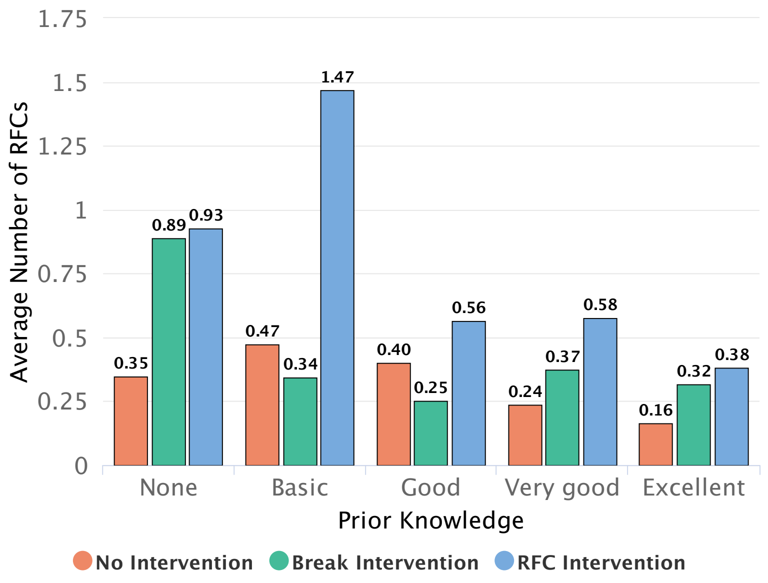


Figure 9.2: Average amount of RFCs per experiment group and skill level per student.

9.4 Additional Tables

Group	Exercise	Too Difficult	Superfluous	OK	Helpful
1	dummy	3.46%	6.54%	51.15%	38.85%
2	random	5.20%	4.83%	32.34%	57.62%
3	recommended	4.21%	5.37%	31.06%	59.36%

Table 9.1: Students' perception of bonus exercises per experiment group in the course Java 2017. Perceptions did vary between experiment groups.

Skill Level	Privacy	Tutors	Students	Contribute	Team Submission	Technical Issues
1	39,13%	47,83%	26,09%	17,39%	30,43%	21,74%
2	24,37%	45,80%	27,73%	27,31%	35,71%	15,55%
3	32,69%	42,79%	23,08%	27,40%	29,33%	13,46%
4	30,32%	41,29%	26,45%	27,74%	30,97%	14,84%
5	39,39%	39,39%	24,24%	16,67%	27,27%	13,64%
Avg.	29,80%	44,05%	26,01%	26,14%	31,37%	14,25%

Table 9.2: Students' answers on question to video tutoring.

Group	CoP	RoA	Avg. Score	Avg. Score Finisher
ND	59.57%	53.19%	51.02%	83.21%
NH	56.41%	48.72%	51.13%	86.16%
BH	58.06%	45.16%	47.23%	86.10%
RH	64.29%	54.76%	49.65%	80.78%
NS	57.50%	50.00%	50.33%	88.13%
BS	63.16%	63.16%	56.79%	83.39%
RS	55.29%	40.00%	43.75%	84.36%

Table 9.3: Key metrics for women per experiment group.

Group	CoP	RoA	Avg. Score	Avg. Score Finisher
ND	51.37%	36.07%	40.07%	83.73%
NH	60.39%	49.76%	49.30%	83.43%
BH	51.90%	44.76%	46.08%	86.49%
RH	52.51%	46.12%	47.18%	86.73%
NS	61.39%	49.50%	50.00%	84.47%
BS	57.53%	49.32%	48.10%	84.79%
RS	59.46%	48.70%	49.46%	85.86%

Table 9.4: Key metrics for men per experiment group.

Group	CoP	RoA	Avg. Score	Avg. Score Finisher
ND	43.53%	33.75%	36.32%	82.19%
NH	45.36%	34.44%	38.22%	83.84%
BH	46.67%	35.87%	39.49%	82.59%
RH	47.16%	37.79%	39.68%	84.22%
NS	46.95%	36.28%	39.81%	86.38%
BS	47.19%	37.29%	40.76%	85.56%
RS	49.13%	38.89%	40.83%	83.43%

Table 9.5: Key metrics for students who have not shared gender information per experiment group.

Group	Women	Men	Unknown	Sum	Share Women	Share Men	Share Unknown
ND	47	183	317	547	8.59%	33.46%	57.95%
NH	39	207	302	548	7.12%	37.77%	55.11%
BH	31	210	315	556	5.58%	37.77%	56.65%
RH	42	219	299	560	7.50%	39.11%	53.39%
NS	40	202	328	570	7.02%	35.44%	57.54%
BS	38	219	303	560	6.79%	39.11%	54.11%
RS	170	809	1260	2239	7.59%	36.13%	56.28%
Sum	407	2049	3124	5580	7.29%	36.72%	55.99%

Table 9.6: Gender distribution within experiment groups. No significant differences are visible.

Platform	URL
Bountify.co	`https://bountify.co`
clojureScript	`http://clojurescript.net`
CodeAnywhere	`https://codeanywhere.com`
CodeAvengers	`https://www.codeavengers.com`
CodeBoard	`https://codeboard.io`
Codecademy	`https://www.codecademy.com`
CodeCombat	`https://codecombat.com`
CodeGolf	`https://codegolf.stackexchange.com`
CodePen	`https://codepen.io`
Codeschool	`https://www.pluralsight.com`
CodeSignal	`https://codesignal.com`
CodeWars	`https://www.codewars.com`
Codility	`https://www.codility.com`
Coding Bat	`https://codingbat.com`
CodinGame	`https://www.codingame.com`
Codio	`https://codio.com`
DOMjudge	`https://www.domjudge.org`
Eclipse,Orion, CHE, Theia	`https://projects.eclipse.org/projects/ecd/`
exercism.io	`https://exercism.io`
GitPod	`https://www.gitpod.io`
HackerEarth	`https://www.hackerearth.com`
HackerRank	`https://www.hackerrank.com`
JACK	`https://www.s3.uni-duisburg-essen.de/en/jack/`
JSFiddle	`https://jsfiddle.net`
Kaggle	`https://www.kaggle.com`
Khan Live Editor	`https://github.com/Khan/live-editor`
Mooshak	`https://mooshak.dcc.fc.up.pt`
Opal	`https://opalrb.com`
Project Euler	`https://projecteuler.net`
PythonTutor	`http://pythontutor.com`
Qualified.io	`https://www.qualified.io`
repl.it	`https://repl.it`
repl.it Classrooms	`https://repl.it/site/classrooms/`
Scratch	`https://scratch.mit.edu`
Skulpt	`http://www.skulpt.org`
Small Basic Online	`http://smallbasic.com`
Snap	`https://snap.berkeley.edu`
SoloLearn	`https://www.sololearn.com`
Sphere Online Judge	`https://spoj.com`
Stepik	`https://welcome.stepik.org`
TopCoder	`https://www.topcoder.com`
Treehouse	`https://teamtreehouse.com`
WebCat	`http://web-cat.org`
WebLinux	`https://codecast.wp.imt.fr/weblinux-2/`

Table 9.7: URLs to coding platforms presented in Chapter 2, sorted alphabetically (all last accessed on 07.01.2020).

Description	URL
MOOC “Intro to Artificial Intelligence”	`https://www.udacity.com/course/intro-to-artificial-intelligence--cs271`
MOOC “Machine Learning”	`https://www.coursera.org/learn/machine-learning`
Dropout Data from MOOCs	`http://www.katyjordan.com/MOOCproject.html`
Definition Tutoring	`https://www.collinsdictionary.com/dictionary/english/tutoring`
German portal for household tips	`https://www.frag-mutti.de`
German recipe portal	`https://www.chefkoch.de`
GitHub	`https://github.com`
BitBucket	`https://bitbucket.org`
GitLab	`https://gitlab.com`
Phabricator	`https://www.phacility.com/phabricator`
AMCAT	`https://www.myamcat.com`
openHPI privacy policy	`https://open.hpi.de/pages/data-protection`
Blog article from Justin Reich	`http://blogs.edweek.org/edweek/edtechresearcher/2014/07/the_ethics_of_educational_experiments.html`
Announcement of “DigitalPakt Schule”	`https://www.bmbf.de/de/bund-und-laender-ueber-digitalpakt-schule-einig-8141.html`
Docker	`https://www.docker.com`
Canvas (open source LMS)	`https://github.com/instructure/canvas-lms`
Coursera	`https://www.coursera.org`
LTI Standard	`https://www.imsglobal.org/activity/learning-tools-interoperability`
StackOverflow Community Survey 2019	`https://insights.stackoverflow.com/survey/2019`

Table 9.8: List of URLs mentioned in the thesis, in order of appearance (all last accessed on 07.01.2020).

Analyzed Questions from the Course-End Survey of the Java 2018 Course ($N = 927$)	
Q1:	The possibility to request comments directly in CodeOcean when solving the practical programming exercises... (Multiple Choice)
A1	was not necessary for me, I went along without it.
A2	lacked visibility. I did not notice that it was there and therefore did not use it.
A3	was helpful. I received helpful feedback.
A4	was not helpful. I did not receive any feedback.
A5	was mediocre. Mostly, my question was not specifically answered. I received only complete solutions that solved the exercise but did add to my understanding.
Q2:	After solving an exercise, some participants were redirected to answer programming questions of other participants. Which of the following options reflect your opinion? (Multiple Answer)
A1	I enjoyed helping others.
A2	I learned something when answering the questions.
A3	I felt disturbed by the requests.
A4	I never encountered questions of other participants.
A5	Usually I was not able to answer the questions, as they were too difficult.
A6	I would have answered more questions if there was an option for that.
Q3:	The interventions that encouraged me to ask for help or take a break... (Multiple Choice)
A1	did not influence me.
A2	never occurred, I did not see them.
A3	helped me. Taking a break or describing my problem helped me when I was struggling.
A4	bugged me. They appeared too often, therefore I ignored them.
Q4:	The interventions that encouraged me to ask for help or take a break... (Multiple Choice)
A1	appeared too early.
A2	appeared after I solved the exercise.
A3	never appeared.
A4	appeared when I was actually struggling.
Q5:	The ungraded bonus exercises... (Multiple Choice)
A1	were too difficult.
A2	were good in general. However, I was given exercises that did not further help my understanding.
A3	were superfluous, I did not work on them.
A4	were helpful and fitting my weaknesses.
Q6:	We plan to introduce video conferences in CodeOcean in the future. Which of the following options reflect your opinion? (Multiple Answer)
A1	I would not use the feature, because I worry about my privacy.
A2	I would like to use video conferences to receive feedback from the teaching team or tutors.
A3	I would be willing to contribute for individual tutoring by the teaching team.
A4	I would like to use video conferences to collaborate with another user on a team submission.
A5	I do not have the technical requirements to take part in video conferences (e.g. no webcam, microphone, low bandwidth).

Table 9.9: Questions from the course-end survey of the Java 2018 course.

List of Figures

List of Tables

Glossary

A/B-testing

Method to test hypotheses. A group is split into two (or more) similar and homogeneous subgroups. Then, treatments are applied to some subgroups and the effects can be compared to the behavior of an untreated control group.

Active Student

A student in our MOOCs who achieved at least one graded point.

API

Abbreviation for Application Programming Interface.

Blended Learning

Combination of online and in-class education. Often, concepts are prepared or reinforced with online material before and after in-person classes.

Bloom's Taxonomy

In this work, Bloom's taxonomy refers to a hierarchical model of cognitive skills.

Bonferroni Correction

Statistical method to counteract the accumulation of alpha errors on multiple comparisons.

BPMN

Abbreviation for Business Process Model and Notation.

Break Interventions

Popups displayed in CodeOcean that suggest students to take a break.

cMOOCs

Abbreviation for connectivist Massive Open Online Courses, courses which focus and rely on content creation of the audience.

CodeHarbor

Sharing platform for programming exercises.

CodeOcean	Code execution and assessment platform used in the experiments of this thesis. Integrates via LTI.
CodePilot	Video conferencing implementation used in CodeOcean for some experiments of this thesis.
Collab Spaces	Functionality on openHPI / openSAP to support teamwork, offering a dedicated forum, file uploads, and other tools such as a synchronized text editor.
Collaboration-based Recommendation / Collaborative Filtering	Recommendation technique that uses knowledge about similar users to recommend potentially suitable items.
Computer Adaptive Testing	A computer-based test that adapts to the student's test result by, e.g., increasing or decreasing the difficulty of questions.
Confirmation of Participation	Confirmation that a student took part in a course. It is issued if the student accessed at least 50% of the course material.
Content-based Recommendation / Content-based Filtering	Recommendation technique that uses prior interaction data of the specific user with the system to recommend potentially suitable items.
Correlation coefficient	Metric between -1 and 1, expressing the strength of a linear correlation between two variables. 0 means no correlation, 1 a perfect positive correlation, and -1 a perfect negative correlation. If not stated otherwise, the Pearson correlation is used in this work.
Digital Literacy	Describes the skillset to assess and create content in digital environments, usually the internet. Comprises knowledge concerning tools (e.g. a word processor), architectures (e.g. principles of programming and the relationship between websites and servers), and mechanisms in virtual communities (e.g. awareness of privacy).
Docker	Containerization solution used in CodeOcean to enable isolated execution of students' submissions at large scale.
eXtreme Apprenticeship	Educational model focusing on personal instruction, coaching, and instructional scaffolding.

Flipped Classroom

Instructional strategy as part of blended learning that reverses the traditional approach of presenting content in-class and having subsequent homework tasks for training. New content is prepared via, e.g., online lectures before in-class sessions, leaving more time to discuss specific problems and engage in active collaboration in presence phases.

Flow State

Mental state of being fully immersed in an activity, being productive and enjoying the activity.

GDPR

Abbreviation for General Data Protection Regulation, a EU regulation on data protection and data privacy.

HDI

Abbreviaton for Human Development Index, a metric reflecting the grade of development of a country.

HPI

Abbreviation for Hasso Plattner Institute.

HPI Schul-Cloud

A platform to technically support K-12 education in Germany.

HTTP

Abbreviation for Hypertext Transfer Protocol, foundation for most of the data transfer in the world wide web.

Hypothesis Test

Statistical test in which two conflictive hypotheses are tested on a given dataset for their probability of being correct.

ICT

Abbreviation for Information and Communication Technology.

IDE

Abbreviation for Integrated Development Environment.

iFrame

HTML element to embed another website into the current web page.

In-Memory Data Management

Data management approach keeping the primary data instance in main memory, resulting in improved processing speed for, e.g., enterprise applications.

Instructional Design

Practice of systematically turning content into educational content, considering the necessary structure, consistency, and examples helpful to acquire knowledge.

Intelligent Tutoring Systems	Learning systems that aim to offer students individualized but automated and immediate feedback.
Intervention	An external modification to the current situation to usually achieve a positive, desired effect.
Item Response Theory	Theory analyzing how for example students assignment answers can be mapped to underlying misunderstandings. Learnings can be used to determine the suitability of subsequent questions, enabling improved learning effects.
Item-based collaborative filtering	Recommendation technique that uses knowledge about similar items to determine similar users, whose combined data can subsequently be used to find additional suitable items.
Ivy League	Group of eight private universities in the US, often being considered the most prestigious universities in the world.
Java	An object oriented-programming language.
JSON	Abbreviation for JavaScript Object Notation, a compact, text-based data interchange format.
JUnit	A popular unit testing framework for Java.
Just-in-Time Intervention	Interventions issued while a student is working on an exercise. In our case, popups during the work on programming exercises.
K-12	Timespan from Kindergarten to 12$^{\text{th}}$ grade.
Krathwohl's taxonomy	A revision of blooms taxonomy, adding a second “knowledge” dimension to the existing “process” dimension, allowing to distinguish between different categories of knowledge in the model.
LTI	Abbreviation for Learning Tools Interoperability, a standard to securely connect e-learning tools, allowing to authenticate users, transfer exercise specific information, and safely communicate students' results.

MOOC

Abbreviation for Massive Open Online Course.

Normal Distribution

A statistical distribution required for many statistical tests. Also called Gauß distribution.

OAuth

Standardized protocol allowing a secure authentication for applications.

OOP

Abbreviation for Object Oriented Programming.

openHPI / openSAP

MOOC platforms of the HPI respectively SAP. These course platforms served as the backbone of our research and interacted with CodeOcean to provide the data for our evaluation.

Pair Programming

Development technique for two programmers who work together on one piece of code. They alternate taking the roles of "driver" and "observer", writing respectively reviewing code.

Pearson Correlation

Metric defined as the covariance of two variables divided by the product of their standard deviations.

Recommender System

A system predicting the suitability of items for a specific user. Usually, these predictions are used to subsequently present the user with the most appropriate items to, e.g., improve sales in e-commerce or the learning success in online courses.

Record of Achievement

Certificate that is issued on openHPI and openSAP if a student reaches at least 50% of the total achievable score within a course.

Request for Comments

Request of a student reaching out for help. Contains the current status of the exercise solution, its run output including stack traces, and the results of the teacher supplied unit tests.

REST

Abbreviation for Representational State Transfer, a software architecture defining the capabilities of web services.

Rhizomatic Learning

Approach to let a community build up learning resources or the curriculum in general.

Rubber Duck Effect

An effect encountered in the process of debugging, referring to the phenomenon that developers often succeed in solving their problem themselves if they explain their code step by step to another person. The person can often be substituted by a rubber duck without losing the benefits.

Ruby / Ruby on Rails

Ruby on Rails is a web-application framework based on the object-oriented programming language Ruby. CodeOcean, CodePilot, and the MOOC platforms of the HPI are using Ruby on Rails within their backend.

Self-Regulated Learning

Mode of learning that is guided by actions including self-monitoring, strategic planning, and reflection.

Server-Sent Events

Standard to allow data transferred to a client after the client established an initial connection. Does not bi-directional communication.

t-test

Statistical hypothesis test usable if the statistic follows Student's t-distribution. Usually, a null hypothesis is tested against an alternative hypothesis.

Tailored Bonus Exercise

Programming exercise that is selected from a pool of potential exercises based on the weaknesses expressed within a student's knowledge model.

Test-driven Development

Software development approach usually used in agile development, requiring tests to be written before the actual implementation is done.

User-based Collaborative Filtering

Statistical hypothesis test usable if the statistic follows Student's t-distribution. Usually, a null hypothesis is tested against an alternative hypothesis.

webRTC

Abbreviation for web Real-Time Communication, a standard for enabling real-time audio and video communication directly within the browser.

WebSocket

Network protocol to enable bi-directional communication between a client and a server via an open connection.

Welch Test

Statistical two-sample t-test based on mean values that does not require equal variances.

xMOOC

Abbreviation for extended MOOC. xMOOCs, as opposed to cMOOCs, often resemble a traditional university course structure and emphasize on the distribution of existing educational content.

Zone of Proximal Development

Concept introduced by Lev Vygotsky, organizing tasks into three categories: tasks that learners can do without help, tasks that learners can do with help (zone of proximal development), and tasks that learners cannot do even with external help.

References

[1] Agrawal, A., Venkatraman, J., Leonard, S., and Paepcke, A. YouEDU: Addressing Confusion in MOOC Discussion Forums by Recommending Instructional Video Clips. In *Proceedings of the 8th International Conference on Educational Data Mining* (Madrid, Spain, June 2015), vol. 8, International Educational Data Mining Society, pp. 297–304.

[2] Aleven, V., and Koedinger, K. R. Limitations of Student Control: Do Students Know when They Need Help? In *Intelligent Tutoring Systems*. Springer, Berlin, Germany, June 2000, pp. 292–303.

[3] Allen, V. L., and Feldman, R. S. Learning through Tutoring: Low-Achieving Children as Tutors. *The Journal of Experimental Education 42*, 1 (Sept. 1973), 1–5.

[4] Altebarmakian, M., and Alterman, R. Design Heuristics to Support Cohesion within Online Collaborative Learning Groups. In *Frontiers in Education* (Cincinnati, 2019), IEEE.

[5] Annis, L. F. The Processes and Effects of Peer Tutoring. *Human Learning: Journal of Practical Research & Applications 2*, 1 (1983), 39–47.

[6] Ariga, A., and Lleras, A. Brief and Rare Mental "Breaks" Keep you Focused: Deactivation and Reactivation of Task Goals Preempt Vigilance Decrements. *Cognition 118*, 3 (Mar. 2011), 439–443.

[7] Auletta, K. Get Rich U - There are no walls between Stanford and Silicon Valley. Should there be? *The New Yorker, Annals of Higher Education* (Apr. 2012). Issue April 30th.

[8] Balaban, N. Seeing the Child, Knowing the Person. In *To Become a Teacher: Making a Difference in Children's Lives*, W. Ayers, Ed. Teachers College Press, New York, June 1996, pp. 49–57.

[9] Bauman, K., and Tuzhilin, A. Recommending Remedial Learning Materials to Students by Filling Their Knowledge Gaps. *Management Information Systems Quarterly 42*, 1 (Mar. 2018), 313–332.

[10] Bergin, S., and Reilly, R. The Influence of Motivation and Comfort-Level on Learning to Program. In *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group, PPIG 05. University of Sussex, Brighton, UK, June 2005.* Psychology of Programming Interest Group, 2005, pp. 293–304.

[11] Bloom, B. S. *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*, 2nd ed. Addison-Wesley Longman Ltd, New York, USA, June 1956.

[12] Bonk, C. J., and Graham, C. R. *The Handbook of Blended Learning: Global Perspectives, Local Designs.* John Wiley & Sons, 2006.

[13] Boud, D. *Enhancing Learning Through Self-Assessment.* Routledge, Oct. 2013.

[14] Broadbent, J., and Poon, W. L. Self-Regulated Learning Strategies & Academic Achievement in Online Higher Education Learning Environments: A Systematic Review. *The Internet and Higher Education 27* (2015), 1–13.

[15] Brown, M. Comfort Zone: Model or Metaphor? *Journal of Outdoor and Environmental Education 12*, 1 (2008), 3–12.

[16] Bruff, D. O., Fisher, D. H., McEwen, K. E., and Smith, B. E. Wrapping a MOOC: Student Perceptions of an Experiment in Blended Learning. *Journal of Online Learning and Teaching 9*, 2 (June 2013), 187–199.

[17] Brusilovsky, P., and Peylo, C. Adaptive and Intelligent Web-Based Educational Systems. In *International Journal of Artificial Intelligence in Education* (Amsterdam, The Netherlands, Apr. 2003), vol. 13, IOS Press, pp. 159–172.

[18] Buckingham, D. Defining Digital Literacy – What do Young People Need to Know About Digital Media? *Nordic Journal of Digital Literacy 1*, 04 (Dec. 2006), 263–277.

[19] Carini, R. M., Kuh, G. D., and Klein, S. P. Student Engagement and Student Learning: Testing the Linkages. *Research in Higher Education 47*, 1 (Feb. 2006), 1–32.

[20] Chandrasekaran, M. K., Kan, M.-Y., Tan, B. C. Y., and Ragupathi, K. Learning Instructor Intervention from MOOC Forums: Early Results and Issues. In *Proceedings of the 8th International Conference on Educational Data Mining* (New York, New York, USA, 4 2015), ACM Press, pp. 512–513.

[21] Chaturvedi, S., Daum, H., and Goldwasser, D. Predicting Instructor's Intervention in MOOC Forums. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Baltimore, Maryland, June 2014), Association for Computational Linguistics, pp. 1501–1511.

[22] Chen, C. H., and Guo, P. J. Improv: Teaching Programming at Scale via Live Coding. In *Proceedings of the Sixth ACM Conference on Learning at Scale* (Chicago, IL, USA, June 2019), ACM.

[23] Chen, P.-S. D., Gonyea, R., and Kuh, G. Learning at a Distance: Engaged or Not? *Innovate: Journal of Online Education 4*, 3 (2008).

[24] Cheng, C. K., Paré, D. E., Collimore, L.-M., and Joordens, S. Assessing the Effectiveness of a Voluntary Online Discussion Forum on Improving Students' Course Performance. *Computers & Education 56*, 1 (Jan. 2011), 253–261.

[25] Christensen, G., Steinmetz, A., Alcorn, B., Bennett, A., Woods, D., and Emanuel, E. The MOOC Phenomenon: Who Takes Massive Open Online Courses and Why? *SSRN Electronic Journal* (Nov. 2013).

[26] Cohen, P. A., and Kulik, J. A. Synthesis of Research on the Effects of Tutoring. *Educational Leadership 39*, 3 (Dec. 1981), 227–229.

[27] Cohen, P. A., Kulik, J. A., and Kulik, C.-L. C. Educational Outcomes of Tutoring: A Meta-Analysis of Findings. *American Educational Research Journal 19*, 2 (Jan. 1982), 237–248.

[28] Cormier, D. *Making the Community the Curriculum*. Pressbooks, Montreal, Canada, 2014.

[29] Csikszentmihalyi, M. *Finding Flow: The Psychology of Engagement with Everyday Life*. Finding Flow: The Psychology of Engagement with Everyday Life. Basic Books, New York, NY, US, 1997.

[30] Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper Perennial Modern Classics, New York, July 2008.

[31] Csikszentmihalyi, M., Rathunde, K., and Whalen, S. *Talented Teenagers: The Roots of Success and Failure*. Cambridge University Press, 1997.

[32] Cuban, L. Whatever Happened to MOOCs?, Oct. 2017. Available at `https://larrycuban.wordpress.com/2017/10/28/whatever-happened-to-moocs/`, last accessed on December 10, 2019.

[33] Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (New York, NY, USA, 2012), CSCW '12, ACM, pp. 1277–1286.

[34] Davis, D., Jivet, I., Kizilcec, R. F., Chen, G., Hauff, C., and Houben, G.-J. Follow the Successful Crowd: Raising MOOC Completion Rates through Social Comparison at Scale. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference* (Mar. 2017), ACM, pp. 454–463.

[35] Douce, C., Livingstone, D., and Orwell, J. Automatic Test-Based Assessment of Programming: A Review. In *Journal on Educational Resources in Computing* (New York, USA, Sept. 2005), vol. 5, ACM.

[36] Drosos, I., Guo, P. J., and Parnin, C. HappyFace: Identifying and Predicting Frustrating Obstacles for Learning Programming at Scale. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Raleigh, NC, Oct. 2017), IEEE, pp. 171–179.

[37] D'Souza, D., Hamilton, M., Harland, J., Muir, P., Thevathayan, C., and Walker, C. Transforming Learning of Programming: A Mentoring Project. In *Proceedings of the Tenth Conference on Australasian Computing Education* (Darlinghurst, Australia, 2008), vol. 78 of *ACE '08*, Australian Computer Society, Inc., pp. 75–84.

[38] Dunkin, M. J. Concepts of Teaching and Teaching Excellence in Higher Education. *Higher Education Research & Development 14*, 1 (Jan. 1995), 21–33.

[39] Ebert, C., and Neve, P. D. Surviving Global Software Development. *IEEE Software 18*, 2 (Mar. 2001), 62–69.

[40] Edwards, S. H. Rethinking Computer Science Education from a Test-First Perspective. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (New York, NY, USA, 2003), OOPSLA '03, ACM, pp. 148–155.

[41] Elbaum, B., Vaughn, S., Hughes, M. T., and Moody, S. W. How Effective Are One-to-One Tutoring Programs in Reading for Elementary Students at Risk for Reading Failure? A Meta-Analysis of the Intervention Research. *Journal of Educational Psychology 92*, 4 (2000), 605–619.

[42] Ellis, C. A., and Gibbs, S. J. Concurrency Control in Groupware Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1989), vol. 18, ACM, pp. 399–407.

[43] Entwistle, N., Tait, H., and McCune, V. Patterns of Response to an Approaches to Studying Inventory across Contrasting Groups and Contexts. *European Journal of Psychology of Education 15*, 1 (Mar. 2000), 33–48.

[44] Gašević, D., Dawson, S., and Siemens, G. Let's Not Forget: Learning Analytics Are about Learning. *TechTrends 59*, 1 (2015), 64–71.

[45] Glance, D. Universities Are Still Standing. The MOOC Revolution That Never Happened. http://theconversation.com/universities-are-still-standing-the-mooc-revolution-that-never-happened-29187, July 2014.

[46] Glassman, E. L., Scott, J., Singh, R., Guo, P. J., and Miller, R. C. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *ACM Transactions on Computer-Human Interaction 22*, 2 (Mar. 2015), Article 7, 1–35.

[47] Goldschmid, B., and Goldschmid, M. L. Peer Teaching in Higher Education: A Review. *Higher Education 5*, 1 (1976), 9–33.

[48] Goldstein, I. P. The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. In *International Journal of Man-Machine Studies* (1979), vol. 11, pp. 51–77.

[49] Gomes, A., and Mendes, A. J. Learning to Program-Difficulties and Solutions. In *International Conference on Engineering Education–ICEE* (2007).

[50] Grünewald, F., Mazandarani, E., Meinel, C., Teusner, R., Totschnig, M., and Willems, C. openHPI - A Case-Study on the Emergence of Two Learning Communities. In *IEEE Global Engineering Education Conference (EDUCON)* (Mar. 2013), IEEE, pp. 1323–1331.

[51] Grünewald, F., Meinel, C., Totschnig, M., and Willems, C. Designing MOOCs for the Support of Multiple Learning Styles. In *Proceedings of the 8th European Conference on Technology Enhanced Learning* (Heidelberg, Germany, Sept. 2013), D. H. Leo, T. Ley, R. Klamma, and A. Harrer, Eds., vol. 8095 of *Lecture Notes in Computer Science*, Springer, pp. 371–382.

[52] Guo, P. J. Online Python Tutor: Embeddable Web-Based Program Visualization for Cs Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), ACM, pp. 579–584.

[53] Guo, P. J. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (New York, NY, USA, 2015), UIST '15, ACM, pp. 599–608.

[54] Guo, P. J. Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2017), CHI '17, ACM, pp. 7070–7083.

[55] Guo, P. J. Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2018), no. 396, ACM, pp. 1–14.

[56] Guo, P. J., Kim, J., and Rubin, R. How Video Production Affects Student Engagement: An Empirical Study of MOOC Videos. In *Proceedings of the First ACM Conference on Learning at Scale* (New York, NY, USA, 2014), L@S '14, ACM, pp. 41–50.

[57] Guo, P. J., White, J., and Zanelatto, R. Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2015), IEEE, pp. 79–87.

[58] Haavind, S., and Sistek-Chandler, C. The Emergent Role of the MOOC Instructor: A Qualitative Study of Trends Toward Improving Future Practice. *International Journal on E-Learning 14*, 3 (July 2015), 331–350.

[59] Hartley, S. S. *Meta-Analysis of the Effects of Individually Paced Instruction in Mathematics.* PhD thesis, University of Colorado at Boulder, 1977.

[60] Hauke, K. A Versatile Platform for Practical Programming Exercises in Massive Open Online Courses. Master's thesis, Hasso Plattner Institute, Dec. 2014.

[61] He, H., Zheng, Q., Di, D., and Dong, B. How Learner Support Services Affect Student Engagement in Online Learning Environments. *IEEE Access 7* (2019), 49961–49973.

[62] He, J., Bailey, J., Rubinstein, B., and Zhang, R. Identifying At-Risk Students in Massive Open Online Courses. *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), 1749–1755.

[63] Heininger, R., Seifert, V., Prifti, L., Utesch, M. C., and Krcmar, H. The Playful Learning Approach for Learning How to Program: A Structured Lesson Plan. In *Proceedings of the 30th Bled eConference* (June 2017), University of Maribor Press.

[64] Hoeffding, W., and Robbins, H. The Central Limit Theorem for Dependent Random Variables. In *The Collected Works of Wassily Hoeffding*, N. I. Fisher and P. K. Sen, Eds. Springer New York, New York, NY, 1994, pp. 205–213.

[65] Huang, J., Dasgupta, A., Ghosh, A., Manning, J., and Sanders, M. Superposter Behavior in MOOC Forums. In *Proceedings of the First ACM Conference on Learning at Scale* (New York, NY, USA, 2014), L@S '14, ACM, pp. 117–126.

[66] Hunt, A., Thomas, D., and Cunningham, W. *The Pragmatic Programmer. From Journeyman to Master*, 1 ed. Addison Wesley, Reading, Mass, Oct. 1999.

[67] Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2010), Koli Calling '10, ACM, pp. 86–93.

[68] Jackson, D., and Usher, M. Grading Student Programs Using ASSYST. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium*

on Computer Science Education (New York, NY, USA, 1997), SIGCSE '97, ACM, pp. 335–339.

[69] Jiang, S., Williams, A. E., Schenke, K., Warschauer, M., and Dowd, D. O. Predicting MOOC Performance with Week 1 Behavior. *Proceedings of the 7th International Conference on Educational Data Mining* (2014), 273–275.

[70] Johnson, H. Goodbye, Chalkboard — Hello, Chat Room. *Time Ideas* (Mar. 2015). Available at `http://time.com/3747816/education-chalkboard-chatroom/`, last accessed on December 10, 2019.

[71] Joksimović, S., Gašević, D., Kovanović, V., Riecke, B. E., and Hatala, M. Social Presence in Online Discussions as a Process Predictor of Academic Performance. *Journal of Computer Assisted Learning 31*, 6 (2015), 638–654.

[72] Khandwala, K., and Guo, P. J. Codemotion: Expanding the Design Space of Learner Interactions with Computer Programming Tutorial Videos. In *Proceedings of the Fifth ACM Conference on Learning at Scale* (2018), vol. 57, pp. 1–10.

[73] Kim, P. *Massive Open Online Courses: The MOOC Revolution.* Routledge, New York, Nov. 2014.

[74] Kizilcec, R. F., and Brooks, C. Diverse Big Data and Randomized Field Experiments in Massive Open Online Courses. In *The Handbook of Learning Analytics*, 1 ed. Society for Learning Analytics Research (SoLAR), Alberta, Canada, 2017, pp. 211–222.

[75] Kizilcec, R. F., and Cohen, G. L. Eight-Minute Self-Regulation Intervention Raises Educational Attainment at Scale in Individualist but not Collectivist Cultures. *Proceedings of the National Academy of Sciences 114*, 17 (2017), 4348–4353.

[76] Kizilcec, R. F., Davis, G. M., and Cohen, G. L. Towards Equal Opportunities in MOOCs: Affirmation Reduces Gender & Social-Class Achievement Gaps in China. In *Proceedings of the Fourth ACM Conference on Learning at Scale* (New York, NY, USA, 2017), ACM, pp. 121–130.

[77] Kizilcec, R. F., and Halawa, S. Attrition and Achievement Gaps in Online Learning. In *Proceedings of the Second ACM Conference on Learning at Scale* (New York, NY, USA, 2015), L@S '15, ACM, pp. 57–66.

[78] Kizilcec, R. F., Pérez-Sanagustín, M., and Maldonado, J. J. Self-Regulated Learning Strategies Predict Learner Behavior and Goal Attainment in Massive Open Online Courses. *Computers & Education 104* (2017), 18–33.

[79] Kizilcec, R. F., and Schneider, E. Motivation As a Lens to Understand Online Learners: Toward Data-Driven Design with the OLEI Scale. *ACM Transactions on Computer-Human Interaction 22*, 2 (3 2015), 1–24.

[80] Kizilcec, R. F., Schneider, E., Cohen, G. L., and McFarland, D. A. Encouraging Forum Participation in Online Courses with Collectivist, Individualist and Neutral Motivational Framings. *Proceedings of the EMOOCS Conference 2014* (2014), 80–87.

[81] Kloft, M., Stiehler, F., Zheng, Z., and Pinkwart, N. Predicting MOOC Dropout over Weeks Using Machine Learning Methods. In *Proceedings of the EMNLP 2014 Workshop on Analysis of Large Scale Social Interaction in MOOCs* (2014), pp. 60–65.

[82] Kloos, C. D., Muñoz-Merino, P. J., Alario-Hoyos, C., Ayres, I. E., and Fernández-Panadero, C. Mixing and Blending MOOC Technologies with Face-to-Face Pedagogies. In *IEEE Global Engineering Education Conference (EDUCON)* (2015), IEEE, pp. 967–971.

[83] Koedinger, K. R., Kim, J., Jia, J. Z., McLaughlin, E. A., and Bier, N. L. Learning is Not a Spectator Sport: Doing is Better Than Watching for Learning from a MOOC. In *Proceedings of the Second ACM Conference on Learning at Scale* (New York, NY, USA, 2015), L@S '15, ACM, pp. 111–120.

[84] Krathwohl, D. R. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice 41*, 4 (2002), 212–218.

[85] Kulkarni, C., Cambre, J., Kotturi, Y., Bernstein, M. S., and Klemmer, S. R. Talkabout: Making Distance Matter with Small Groups in Massive Classes. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (2015), ACM, pp. 1116–1128.

[86] Kulkarni, C. E., Bernstein, M. S., and Klemmer, S. R. PeerStudio: Rapid Peer Feedback Emphasizes Revision and Improves Performance. In *Proceedings of the Second ACM Conference on Learning at Scale* (New York, NY, USA, 2015), ACM, pp. 75–84.

[87] Kurtin, K. S., O'Brien, N., Roy, D., and Dam, L. The Development of Parasocial Interaction Relationships on YouTube. *The Journal of Social Media in Society 7*, 1 (May 2018), 233–252.

[88] Lackner, E., Kopp, M., and Ebner, M. How to MOOC? –A Pedagogical Guideline for Practitioners. In *Proceedings of the 10th International Scientific Conference" eLearning and Software for Education"* (2014).

[89] Lajoie, S. P., and Azevedo, R. *Teaching and Learning in Technology-Rich Environments*. Routledge Handbooks Online, May 2006.

[90] Lang, C., Siemens, G., Wise, A., and Gasevic, D. *Handbook of Learning Analytics*. SOLAR, Society for Learning Analytics and Research, 2017.

[91] Leberman, S. I., and Martin, A. J. Does Pushing Comfort Zones Produce Peak Learning Experiences? *Journal of Outdoor and Environmental Education 7*, 1 (2002), 10–19.

[92] Lee, D., Watson, S. L., and Watson, W. R. Systematic Literature Review on Self-Regulated Learning in Massive Open Online Courses. *Australasian Journal of Educational Technology 35*, 1 (2019), 28–41.

[93] Leinonen, J., Ihantola, P., and Hellas, A. Preventing Keystroke Based Identification in Open Data Sets. In *Proceedings of the Fourth ACM Conference on Learning at Scale* (2017), ACM, pp. 101–109.

[94] Lemov, D. Bloom's Taxonomy —That Pyramid is a Problem, Mar. 2017. Available at `https://teachlikeachampion.com/blog/blooms-taxonomy-pyramid-problem/`, last accessed on December 10, 2019.

[95] Linden, A., and Fenn, J. Understanding Gartner's Hype Cycles. *Strategic Analysis Report N^o R-20-1971. Gartner, Inc* (2003).

[96] Liyanagunawardena, T. R., Lundqvist, K. O., Micallef, L., and Williams, S. A. Teaching Programming to Beginners in a Massive Open Online Course. In *Proceedings of the OER14 Conference* (Apr. 2014), Association for Learning Technology, pp. 1–7.

[97] Löwis, M., Staubitz, T., Teusner, R., Renz, J., Meinel, C., and Tannert, S. Scaling Youth Development Training in IT Using an xMOOC Platform. In *IEEE Frontiers in Education Conference (FIE)* (Oct. 2015), pp. 1–9.

[98] Manouselis, N., Drachsler, H., Vuorikari, R., Hummel, H., and Koper, R. Recommender Systems in Technology Enhanced Learning. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer US, Boston, MA, 2011, pp. 387–415.

[99] Margolis, H., and McCabe, P. P. Self-Efficacy: A Key to Improving the Motivation of Struggling Learners. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas 77*, 6 (2004), 241–249.

[100] Martin, F., Wang, C., and Sadaf, A. Student Perception of Helpfulness of Facilitation Strategies That Enhance Instructor Presence, Connectedness, Engagement and Learning in Online Courses. *The Internet and Higher Education 37* (2018), 52–65.

[101] Marwick, A. E. You May Know Me from YouTube: (Micro-)Celebrity in Social Media. In *A Companion to Celebrity*, S. R. P. David Marshall, Ed. John Wiley & Sons, Inc, Oct. 2015, pp. 333–350.

[102] McCabe, T. J. A Complexity Measure. *IEEE Transactions on Software Engineering SE-2*, 4 (Dec. 1976), 308–320.

[103] McKeachie, W. J. Critical Elements in Training University Teachers. *International Journal for Academic Development 2*, 1 (May 1997), 67–74.

[104] McKenna, L. The Big Idea That Can Revolutionize Higher Education: 'MOOC'. https://www.theatlantic.com/business/archive/2012/05/the-big-idea-that-can-revolutionize-higher-education-mooc/256926/, May 2012.

[105] McKinsey, J. Remote Pair Programming in a Visual Programming Language. *Technical Report No. UCB/EECS-2015-139* (2015).

[106] Means, B., Toyama, Y., Murphy, R., Bakia, M., and Jones, K. *Evaluation of Evidence-Based Practices in Online Learning: A Meta-Analysis and Review of Online Learning Studies.* US Department of Education, May 2009.

[107] Meinel, C., Willems, C., Renz, J., and Staubitz, T. Reflections on Enrollment Numbers and Success Rates at the openHPI MOOC Platform. *Proceedings of the EMOOCS Conference 2014* (2014), 101–106.

[108] Michlik, P., and Bielikova, M. Exercises Recommending for Limited Time Learning. In *Proceedings of the 1st Workshop on Recommender Systems for Technology Enhanced Learning* (2010), vol. 1, Elsevier, pp. 2821–2828.

[109] Murray, T. Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art. *International Journal of Artificial Intelligence in Education*, 10 (1999), 98–129.

[110] Naumann, F., Jenders, M., and Papenbrock, T. Ein Datenbankkurs mit 6000 Teilnehmern. *Informatik-Spektrum 37*, 4 (Aug. 2014), 333–340.

[111] Nkambou, R., Bourdeau, J., and Psyché, V. Building Intelligent Tutoring Systems: An Overview. In *Advances in Intelligent Tutoring Systems*, J. Kacprzyk, R. Nkambou, J. Bourdeau, and R. Mizoguchi, Eds., vol. 308. Springer Berlin Heidelberg, 2010, pp. 361–375.

[112] Nwana, H. S. Intelligent Tutoring Systems: An Overview. *Artificial Intelligence Review 4*, 4 (Dec. 1990), 251–277.

[113] Ohta, A. S. Interlanguage Pragmatics in the Zone of Proximal Development. *System 33*, 3 (Sept. 2005), 503–517.

[114] Onah, D. F., Sinclair, J., and Boyatt, R. Dropout Rates of Massive Open Online Courses: Behavioural Patterns. *Proceedings of the 6th International Conference on Education and New Learning Technologies 1* (2014), 5825–5834.

[115] Panadero, E. A Review of Self-regulated Learning: Six Models and Four Directions for Research. *Frontiers in Psychology 8* (Apr. 2017).

[116] Papadopoulos, K., Sritanyaratana, L., and Klemmer, S. R. Community TAs Scale High-Touch Learning, Provide Student-Staff Brokering, and Build Esprit de Corps. In *Proceedings of the First ACM Conference on Learning at Scale* (2014), ACM, pp. 163–164.

[117] Pea, R. Practices of Distributed Intelligence and Designs for Education. *Distributed Cognitions Psychological and Educational Considerations* (1993), 47–87.

[118] Pintrich, P. R. Chapter 14 - The Role of Goal Orientation in Self-Regulated Learning. In *Handbook of Self-Regulation*, M. Boekaerts, P. R. Pintrich, and M. Zeidner, Eds. Academic Press, San Diego, Jan. 2000, pp. 451–502.

[119] Price, L., Richardson, J. T. E., and Jelfs, A. Face-to-Face Versus Online Tutoring Support in Distance Education. *Studies in Higher Education 32*, 1 (Feb. 2007), 1–20.

[120] Prinsloo, P., and Slade, S. Ethics and Learning Analytics: Charting the (Un) Charted. In *The Handbook of Learning Analytics*. SOLAR, 2017, pp. 49–57.

[121] Queirós, R., and Leal, J. P. Programming Exercises Evaluation Systems - An Interoperability Survey. In *Proceedings of the 4th International Conference on Computer Supported Education* (2012).

[122] Ramsden, P. A Performance Indicator of Teaching Quality in Higher Education: The Course Experience Questionnaire. *Studies in Higher Education 16*, 2 (Jan. 1991), 129–150.

[123] Rasch, D., Kubinger, K. D., and Moder, K. The Two-Sample t Test: Pre-Testing Its Assumptions Does Not Pay Off. *Statistical Papers 52*, 1 (Feb. 2011), 219–231.

[124] Reek, K. A. The TRY System -or- How to Avoid Testing Student Programs. In *Proceedings of the Twentieth SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 1989), SIGCSE '89, ACM, pp. 112–116.

[125] Rees, J. Massive Online Courses are Terrible for Students and Professors. https://slate.com/technology/2013/07/moocs-could-be-disastrous-for-students-and-professors.html, July 2013.

[126] Reich, J. Rebooting MOOC Research. In *Science*, vol. 347. American Association for the Advancement of Science (AAAS), Washington, USA, Jan. 2015, pp. 34–35.

[127] Reich, J., and Ruipérez-Valiente, J. A. The MOOC Pivot. In *Science*, vol. 363. American Association for the Advancement of Science (AAAS), Washington, USA, Jan. 2019, pp. 130–131.

[128] Renz, J., Hoffmann, D., Staubitz, T., and Meinel, C. Using A/B Testing in MOOC Environments. In *Proceedings of the Sixth International*

Conference on Learning Analytics & Knowledge (New York, NY, USA, 2016), LAK '16, ACM, pp. 304–313.

[129] Richardson, J. T. E., Long, G. L., and Foster, S. B. Academic Engagement in Students with a Hearing Loss in Distance Education. *Journal of Deaf Studies and Deaf Education 9*, 1 (2004), 68–85.

[130] Robins, A., Rountree, J., and Rountree, N. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education 13*, 2 (June 2003), 137–172.

[131] Rohloff, T., Sauer, D., and Meinel, C. On the Acceptance and Usefulness of Personalized Learning Objectives in MOOCs. In *Proceedings of the Sixth ACM Conference on Learning at Scale* (New York, USA, 2019), no. 4, ACM, pp. 1–10.

[132] Roser, M., and Ortiz-Ospina, E. Literacy. *Our World in Data* (Aug. 2016). retrieved from `https://ourworldindata.org/literacy`.

[133] Ross, J. A. The Reliability, Validity, and Utility of Self-Assessment. *Practical Assessment Research & Evaluation* (Oct. 2006), 1–13.

[134] Sandeen, C. Essay: Looking Back at Predictions about MOOCs — Inside Higher Ed, June 2017. Available at `https://www.insidehighered.com/views/2017/06/22/essay-looking-back-predictions-about-moocs`, last accessed on December 10, 2019.

[135] Sarma, A., Maccherone, L., Wagstrom, P., and Herbsleb, J. Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development. In *IEEE 31st International Conference on Software Engineering* (May 2009), pp. 23–33.

[136] Schwinning, N., Striewe, M., Massing, T., Hanck, C., and Goedicke, M. Towards Digitalisation of Summative and Formative Assessments in Academic Teaching of Statistics. *Proceedings of the Fifth International Conference on Learning and Teaching in Computing and Engineering* (Nov. 2017).

[137] Segal, A., Katzir, Z., Gal, K., Shani, G., and Shapira, B. EduRank: A Collaborative Filtering Approach to Personalization in E-learning. *Proceedings of the 7th International Conference on Educational Data Mining* (2014), 68–75.

[138] Serth, S., Teusner, R., Renz, J., and Uflacker, M. Evaluating Digital Worksheets with Interactive Programming Exercises for K-12 Education. In *Proceedings of the49th Frontiers in Education Conference* (2019), IEEE, pp. 1–9.

[139] Sharma, K., Mangaroska, K., Trætteberg, H., Lee-Cultura, S., and Giannakos, M. Evidence for Programming Strategies in University Coding Exercises. In *Lifelong Technology-Enhanced Learning* (2018), V. Pammer-Schindler, M. Pérez-Sanagustín, H. Drachsler, R. Elferink, and M. Scheffel, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 326–339.

[140] Sharrock, R., Angrave, L., and Hamonic, E. WebLinux: A Scalable In-Browser and Client-Side Linux and IDE. In *Proceedings of the Fifth ACM Conference on Learning at Scale* (New York, NY, USA, 2018), no. 45, ACM, pp. 1–2.

[141] Sharrock, R., Hamonic, E., Hiron, M., and Carlier, S. CODECAST: An Innovative Technology to Facilitate Teaching and Learning Computer Programming in a C Language Online Course. In *Proceedings*

of the Fourth ACM Conference on Learning at Scale (New York, NY, USA, 2017), L@S '17, ACM, pp. 147–148.

[142] Snyder, C. R., and Lopez, S. J., Eds. *Handbook of Positive Psychology.* Oxford University Press, Dec. 2001.

[143] Soozandehfar, S. M. A., and Adeli, M. R. A Critical Appraisal of Bloom's Taxonomy. *American Research Journal of English and Literature 2* (2016), 1–9.

[144] Spyropoulou, N., Pierrakeas, C., and Kameas, A. Creating MOOC Guidelines Based on Best Practices. *Proceedings of the 6th International Conference on Education and New Learning Technologies* (2014), 6981–6990.

[145] Staubitz, T., Klement, H., Renz, J., Teusner, R., and Meinel, C. Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses. In *Proceedings of the Fourth IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (Dec. 2015), pp. 23–30.

[146] Staubitz, T., Klement, H., Teusner, R., Renz, J., and Meinel, C. CodeOcean - A Versatile Platform for Practical Programming Excercises in Online Environments. In *IEEE Global Engineering Education Conference (EDUCON)* (Apr. 2016), IEEE, pp. 314–323.

[147] Staubitz, T., and Meinel, C. Collaboration and Teamwork on a MOOC Platform: A Toolset. In *Proceedings of the Fourth ACM Conference on Learning at Scale* (New York, NY, USA, 2017), L@S '17, ACM, pp. 165–168.

[148] Staubitz, T., and Meinel, C. Team Based Assignments in MOOCs: Results and Observations. In *Proceedings of the Fifth ACM Conference on Learning at Scale* (2018), no. 47, pp. 1–4.

[149] Staubitz, T., Petrick, D., Bauer, M., Renz, J., and Meinel, C. Improving the Peer Assessment Experience on MOOC Platforms. In *Proceedings of the Third ACM Conference on Learning at Scale* (2016), ACM, pp. 389–398.

[150] Staubitz, T., Pfeiffer, T., Renz, J., Willems, C., and Meinel, C. Collaborative Learning in a MOOC Environment. In *Proceedings of the 8th International Conference of Education, Research and Innovation* (Seville, Spain, Nov. 2015), IATED, pp. 8237–8246.

[151] Staubitz, T., Renz, J., Willems, C., and Meinel, C. Supporting Social Interaction and Collaboration on an xMOOC Platform. *Proceedings of the 6th International Conference on Education and New Learning Technologies* (2014), 6667–6677.

[152] Staubitz, T., Teusner, R., and Meinel, C. MOOCs in Secondary Education - Experiments and Observations from German Classrooms. In *IEEE Global Engineering Education Conference (EDUCON)* (Apr. 2019), IEEE, pp. 173–182.

[153] Staubitz, T., Teusner, R., Meinel, C., and Prakash, N. Cellular Automata as Basis for Programming Exercises in a MOOC on Test Driven Development. In *Proceedings of the Fifth IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (Dec. 2016), IEEE, pp. 374–380.

[154] Striewe, M. An Architecture for Modular Grading and Feedback Generation for Complex Exercises. *Science of Computer Programming 129* (Nov. 2016), 35–47.

[155] Sun, J. C.-Y., and Rueda, R. Situational Interest, Computer Self-Efficacy and Self-Regulation: Their Impact on Student Engagement in Distance Education. *British Journal of Educational Technology 43*, 2 (2012), 191–204.

[156] Taylor, C. *Stopout Prediction in Massive Open Online Courses.* Thesis, Massachusetts Institute of Technology, 2014.

[157] Taylor, C., Veeramachaneni, K., and O'Reilly, U.-M. Likely to Stop? Predicting Stopout in Massive Open Online Courses. *Computing Research Repository on arXiv (CoRR)* (8 2014), 25.

[158] Teusner, R., and Hille, T. On the Impact of Programming Exercise Descriptions. In *Proceedings of the Fifth Conference on Learning With MOOCS* (Sept. 2018), pp. 51–54.

[159] Teusner, R., Hille, T., and Hagedorn, C. Aspects on Finding the Optimal Practical Programming Exercise for MOOCs. In *IEEE Frontiers in Education Conference (FIE)* (Oct. 2017), pp. 1–8.

[160] Teusner, R., Hille, T., and Staubitz, T. Effects of Automated Interventions in Programming Assignments: Evidence from a Field Experiment. In *Proceedings of the Fifth ACM Conference on Learning at Scale* (New York, NY, USA, 2018), no. 60 in L@S '18, ACM, pp. 1–10.

[161] Teusner, R., Matthies, C., and Staubitz, T. What Stays in Mind? - Retention Rates in Programming MOOCs. In *IEEE Frontiers in Education Conference (FIE)* (2018), IEEE, pp. 1–9.

[162] Teusner, R., Rollmann, K.-A., and Renz, J. Taking Informed Action on Student Activity in MOOCs. In *Proceedings of the Fourth ACM Conference on Learning at Scale* (New York, NY, USA, 2017), L@S '17, ACM, pp. 149–152.

[163] Teusner, R., Wittstruck, N., and Staubitz, T. Video Conferencing as a Peephole to MOOC Participants: Understanding Struggling Students and Uncovering Content Defects. In *IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (2017), IEEE, pp. 100–107.

[164] Topping, K. J. The Effectiveness of Peer Tutoring in Further and Higher Education: A Typology and Review of the Literature. *Higher Education 32*, 3 (Oct. 1996), 321–345.

[165] Treinen, J. J., and Miller-Frost, S. L. Following the Sun: Case Studies in Global Software Development. *IBM Systems Journal 45*, 4 (2006), 773–783.

[166] Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A., and Saleem, F. The Effect of Automatic Assessment on Novice Programming: Strengths and Limitations of Existing Systems. *Computer Applications in Engineering Education 26*, 6 (2018), 2328–2341.

[167] Vihavainen, A., Paksula, M., and Luukkainen, M. Extreme Apprenticeship Method in Teaching Programming for Beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (2011), ACM, pp. 93–98.

[168] Šimko, M., Barla, M., and Bieliková, M. ALEF: A Framework for Adaptive Web-Based Learning 2.0. In *Key Competencies in the Knowl-*

edge Society (2010), IFIP Advances in Information and Communication Technology, Springer, Berlin, Heidelberg, pp. 367–378.

[169] Vygotsky, L. S. *Mind in Society: The Development of Higher Psychological Processes.* Harvard University Press, Cambridge, Massachusetts, USA, July 1978.

[170] Wang, A. Y., Mitts, R., Guo, P. J., and Chilana, P. K. Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Apr. 2018), no. 511, ACM, pp. 1–13.

[171] Warner, J., and Guo, P. J. CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (2017), ACM, pp. 1136–1141.

[172] Wen, M., Yang, D., and Rose, C. Sentiment Analysis in MOOC Discussion Forums: What Does It Tell Us? In *Proceedings of the 7th International Conference on Educational Data Mining* (2014), International Educational Data Mining Society, pp. 130–137.

[173] Whitehill, J., Mohan, K., Seaton, D., Rosen, Y., and Tingley, D. MOOC Dropout Prediction: How to Measure Accuracy? In *Proceedings of the Fourth ACM Conference on Learning at Scale* (New York, NY, USA, 2017), L@S '17, ACM, pp. 161–164.

[174] Whitehill, J., Williams, J., Lopez, G., Coleman, C., and Reich, J. Beyond Prediction: First Steps Toward Automatic Intervention in MOOC Student Stopout. *Proceedings of the 8th International Conference on Educational Data Mining* (2015), 171–178.

[175] Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. Strengthening the Case for Pair Programming. *IEEE Software 17*, 4 (2000), 19–25.

[176] Wilson, T. D. Models in Information Behaviour Research. *Journal of Documentation 55*, 3 (1999), 249–270.

[177] Wise, A. F., and Shaffer, D. W. Why Theory Matters More than Ever in the Age of Big Data. *Journal of Learning Analytics 2*, 2 (2015), 5–13.

[178] Wong, J., Baars, M., de Koning, B. B., van der Zee, T., Davis, D., Khalil, M., Houben, G.-J., and Paas, F. Educational Theories and Learning Analytics: From Data to Knowledge. In *Utilizing Learning Analytics to Support Study Success.* Springer, 2019, pp. 3–25.

[179] Worstall, T. What MOOCs Will Really Kill Is The Research University, July 2013. Available at `https://www.forbes.com/sites/timworstall/2013/07/27/what-moocs-will-really-kill-is-the-research-university/`, last accessed on December 10, 2019.

[180] Yang, D., Adamson, D., and Rosé, C. P. Question Recommendation with Constraints for Massive Open Online Courses. In *Proceedings of the 8th ACM Conference on Recommender Systems* (New York, USA, 2014), ACM, pp. 49–56.

[181] Yang, D., Sinha, T., and Adamson, D. Turn on, Tune in, Drop out: Anticipating Student Dropouts in Massive Open Online Courses. In *Proceedings of the NIPS 2013 Workshop on Data Driven Education* (2013), Stanford Lytics Lab, pp. 1–8.

[182] Yeomans, M., and Reich, J. Planning Prompts Increase and Forecast Course Completion in Massive Open Online Courses. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference* (New York, NY, USA, 2017), ACM, pp. 464–473.

[183] Yousef, A. M. F., Chatti, M. A., Schroeder, U., and Wosnitza, M. What Drives a Successful MOOC? An Empirical Examination of Criteria to Assure Design Quality of MOOCs. In *IEEE 14th International Conference on Advanced Learning Technologies* (2014), IEEE, pp. 44–48.

[184] Zeller, A. *Why Programs Fail: A Guide to Systematic Debugging.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[185] Zheng, S., Rosson, M. B., Shih, P. C., and Carroll, J. M. Understanding Student Motivation, Behaviors and Perceptions in MOOCs. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (New York, NY, USA, 2015), ACM, pp. 1882–1895.

[186] Zimmerman, B. J. Chapter 2 - Attaining Self-Regulation: A Social Cognitive Perspective. In *Handbook of Self-Regulation*, M. Boekaerts, P. R. Pintrich, and M. Zeidner, Eds. Academic Press, San Diego, Jan. 2000, pp. 13–39.

[187] Zimmerman, B. J., and Moylan, A. R. Self-Regulation: Where Metacognition and Motivation Intersect. In *Handbook of Metacognition in Education*, The Educational Psychology Series. Routledge/Taylor & Francis Group, New York, NY, US, 2009, pp. 299–315.

Eigenständigkeitserklärung
(Declaration of Authorship)

Hiermit versichere ich an Eides statt, dass die vorliegende Arbeit bisher an keiner anderen Hochschule eingereicht worden ist sowie selbstständig und ausschließlich mit den angegebenen Mitteln angefertigt wurde.

Potsdam, den 08. Januar 2020

Ralf Teusner