IT Systems Engineering
Universität Potsdam

Master's Thesis

A Versatile Platform for Practical Programming Exercises in Massive Open Online Courses

Eine vielseitige Plattform für praktische Programmieraufgaben in Massive Open Online Courses

Hauke Klement

Submitted on
Friday, December 19th, 2014

Chair
Internet Technologies and Systems

Supervisors
Prof. Dr. Christoph Meinel
Dipl.-Inf. (FH) Jan Renz
Thomas Staubitz, M.Sc.
Dipl.-Inf. Christian Willems

Abstract

Programming skills are highly demanded, but qualified employees are rare. In order to counteract the shortage of software engineers, computer science (CS) education has to be expanded and diversified. Massive open online courses (MOOCs) bear a tremendous potential for teaching programming to a large and diverse audience. However, they are usually composed of video lectures, reading material, and easily assessable quizzes, which is not sufficient for proper programming education. In order to teach programming, MOOCs have to provide practice, feedback, and assessment of students' programs.

This work deals with the question how MOOCs can integrate practical programming assignments in a manner that meets the demands of novice programmers and satisfies the inherent scalability requirements of large-scale e-learning environments.

Options for providing MOOC participants with the means for practicing programming are explored. In addition, requirements of a large-scale programming education solution are identified.

On this basis, this work introduces Code Ocean, a web-based platform for practical programming exercises, which is designed to be used in MOOCs that teach programming to beginners. The platform's concept and implementation are discussed with regard to tools provided to students and teachers, sandboxed and scalable code execution, scalable assessment, and interoperability.

Zusammenfassung

Programmierkenntnisse sind sehr gefragt; qualifiziertes Personal ist jedoch selten. Um dem Fachkräftemangel entgegenzuwirken, muss die Informatikausbildung ausgeweitet und breiter gefächert werden. Massive Open Online Courses (MOOCs) weisen enormes Potenzial auf, einer großen und heterogenen Zielgruppe das Programmieren zu lehren. Jedoch bestehen sie zumeist aus Lehrvideos, Lesestoff und leicht zu bewertenden Quiz, was einer kompetenten Programmierausbildung nicht genügt. Um das Programmieren zu lehren, müssen MOOCs dezidiert praktische Erfahrung vermitteln, Feedback zur Verfügung stellen und Bewertung von Programmen ermöglichen.

Diese Arbeit befasst sich mit der Fragestellung, wie MOOCs praktische Programmieraufgaben in einer Weise integrieren können, die sowohl die Bedürfnisse von Programmieranfängern erfüllt als auch den Skalierbarkeitsanforderungen gerecht wird, welche großen E-Learning-Umgebungen innewohnen.

Möglichkeiten, MOOC-Teilnehmer mit Mitteln zum praktischen Programmieren auszustatten, werden untersucht. Des Weiteren werden Anforderungen an Lösungen zur Programmierausbildung in großem Rahmen identifiziert.

Darauf aufbauend stellt diese Arbeit Code Ocean vor, eine webbasierte Plattform für praktische Programmierübungen, welche für MOOCs, die Programmieren lehren, konzipiert worden ist. Konzept und Implementierung der Plattform werden unter den Aspekten der Werkzeuge, welche Lernenden und Lehrenden geboten werden, isolierter und skalierbarer Code-Ausführung, skalierbarer Code-Bewertung sowie Interoperabilität diskutiert.

Acknowledgements

I want to express my gratitude to everyone who supported me in bringing this thesis to completion.

At first, I want to thank my supervisors Prof. Dr. Christoph Meinel, Jan Renz, Thomas Staubitz, and Christian Willems, whose ideas, guidance, and support were of great value to me. Likewise, I want to thank Ralf Teusner for his assistance.

Many thanks go to my fellow students who shared an office with me. The pleasant working atmosphere as well as our joint lunch, coffee, and frisbee breaks were excellent motivators. Furthermore, I will keep Fridays' vegetarian burgers in good memory. Special thanks are directed to Dominic Petrick who has gone with me through the various stages and challenges of writing a master's thesis.

I want to thank Josefine Harzmann for her steady encouragement, her culinary support, and her tolerance for late hours and weekend work.

Moreover, I want to thank the people who took the time to review parts of my work.

Finally, I want to express my gratitude to my parents for supporting me during my studies.

Contents

Chapter 1

Introduction

1.1 Motivation

In today's society, technological innovation plays an ever-increasing role for a country's development and economic growth [5]. Technology touches virtually every part of our daily lives. As a consequence, programming abilities are required in many professional areas. Programming has become a key qualification of the 21st century.

Although programming skills are highly demanded, qualified personnel are rare. The number of software engineering graduates is estimated to be far below the predicted need [87], which is attributable to problems in both attracting and keeping students [11]. Moreover, the participation of women and minorities in information technology (IT) is dramatically low [11, 87]. Retention of first-year computer science (CS) students is a worldwide concern [50] that has to be addressed.

Programming courses have high dropout rates and are generally considered to be difficult [50, 72]. In order to succeed, students have to gain competencies in different areas, including syntax, semantics, algorithmic thinking, program design, and program comprehension [59]. Lack of early success is regarded to be one of the key issues leading to low success rates in introductory programming classes [84]. In contrast, comfort level was found to have the strongest positive influence on success [96]. Consequently, providing students with a comfortable and non-intimidating learning environment can help to decrease the number of university dropouts. In addition, interesting topics that offer space for creativity, such as robotics and mobile applications, can support students' motivation [65].

In order to meet the increasing demand for IT professionals, raising completion rates is not enough, though. Governments and educational institutes have to promote CS to a greater audience of young people [28]. They are supported by non-profit organizations such as Code.org[1], which aims at expanding CS education in schools worldwide, inspiring more students, and improving diversity in CS. Supported by well-known IT companies and celebrities, such as Bill Gates, Barack Obama, and Mark Zuckerberg, the initiative develops and distributes introductory programming courses and online video tutorials that are designed to be attractive to children.

[1] `http://code.org/`

In fact, children are more likely to develop an interest in technology if they are exposed to it at an early age [48]. Introducing the young to programming in an appealing environment can help to inspire more children to engage in CS [49]. Popular means for providing young people with motivating programming experience and for changing unfavorable misconceptions about CS include summer camps [5], workshops [48], and after-school activities [52], which often revolve around topics that are attractive to adolescents. In addition, there are specialized programming environments, aimed at teaching basic programming concepts in an entertaining way. Starter programming environments such as Alice[2], Greenfoot[3], Scratch[4], and ScratchJr[5] [21] are usually visually attractive, interactive, and playful in order to appeal to children [28]. Also, drag-and-drop programming approaches, which make first programming steps more accessible, are common.

A shortcoming of most local activities for promoting CS is their limited capacity [88]. More scalable CS education can be provided by online programming courses. Web-based programming education platforms, such as Code School[6], Codecademy[7], and Khan Academy[8], offer easy-to-start programming environments, which allow those who are keen to learn to get a taste of programming with a minimal initial threshold [85]. These platforms often make use of instructional videos, themed courses, and game mechanics, such as points and badges, to keep students motivated.

An equally scalable but more social learning experience is provided by massive open online courses (MOOCs), which make high-quality courses, covering various subjects, freely available to anyone connected to the Internet. Due to these properties, MOOCs have a tremendous potential to introduce a large and diverse audience to the basics of programming. In fact, introductory courses in CS and engineering, which are already offered by the majority of MOOC providers, are regarded to be an adequate means to attract students into the subject [85]. In addition, there are courses aimed at introducing teachers to new topics that can improve the appeal of their teachings [43].

Google[9] recently published a guide for technical development[10] that suggests software engineering students to supplement their university knowledge and to develop technical skills through self-paced online learning by means of MOOCs.

Within the scope of an initiative to create new jobs in the digital sector, the European Commission published a study [14] investigating the demand and supply of MOOCs related to web skills. Results of an associated survey show that IT professionals consider MOOCs the best way to learn such abilities. The responses to the survey also indicate that learners are less interested in theoretical content but value practical experience. According to the study, neither the standard formulas of academic courses nor the prevalent MOOC format are optimal for teaching web-related skills. Instead, survey participants noted the importance of learning-by-doing practices.

[2] http://www.alice.org/
[3] http://www.greenfoot.org/
[4] http://scratch.mit.edu/
[5] http://www.scratchjr.org/
[6] https://www.codeschool.com/
[7] http://www.codecademy.com/
[8] https://www.khanacademy.org/
[9] http://www.google.com/
[10] http://www.google.com/edu/tools-and-solutions/guide-for-technical-development/

Traditionally, MOOCs are composed of video lectures, reading material, and assessment tools that are limited to a set of automatically gradable assignment types, such as quizzes. However, these means are not sufficient for teaching programming, which requires practice, feedback, and code assessment. In order to provide an attractive and supporting platform for teaching programming to the masses, MOOCs have to fit the requirements of programming education. While MOOCs can deliver course contents to tens of thousands of students, providing appropriate tools for practice and offering assessable practical programming assignments usually exceeds their built-in capabilities.

1.2 Research Question

This work deals with the question how MOOCs can integrate practical programming assignments in a manner that meets the demands of novice programmers and satisfies the inherent scalability requirements of large-scale e-learning environments.

In order to answer this question, three important partial aspects are addressed. Firstly, we investigate how learners can be provided with a development environment for practicing programming that fits the needs of beginning programmers. Secondly, we explore how scalable execution and assessment of programming assignments can be achieved. Thirdly, we examine how the integration of practical programming exercises into existing e-learning systems, such as learning management systems (LMSs) and MOOC platforms, can be facilitated. While providing a secure platform for the execution of learners' code is a relevant issue that is to be addressed in this thesis, in-depth security considerations are not within the scope of this work.

As a solution to the research question, Code Ocean is presented. Being a web-based platform providing practical programming tasks for MOOCs and other e-learning environments, Code Ocean aims at facilitating the entry into programming and at attracting a diverse audience of interested learners. While the application is designed to be novice-friendly, it is not tailored to beginner-oriented programming paradigms. Rather, it is designed to support a wide range of various programming languages in a fashion that encourages novices yet is not limited to trivial programming tasks.

1.3 Outline

The subsequent chapters are organized as follows: Chapter 2 provides background on several subjects associated to the topic of this work. Chapter 3 covers the concepts that Code Ocean is based upon. Chapter 4 focuses on the implementation of the application. Chapter 5 presents an evaluation in terms of scalability and versatility. Chapter 6 proposes future work. Finally, Chapter 7 draws a conclusion.

Chapter 2

Background

This chapter provides background on several subjects associated to the topic of this work. Section 2.1 focuses on MOOCs. Subsequently, Section 2.2 presents the German MOOC provider openHPI, which constitutes the research context of this thesis. Section 2.3 discusses the role of practice in programming education. Section 2.4 deals with web-based applications in general and web-based development tools in particular. Section 2.5 covers assessment of programming assignments and describes approaches that permit scalable assessment in large-scale programming courses. Section 2.6 introduces options for providing learners in MOOCs with programming tools and compares their advantages and drawbacks. Finally, Section 2.7 examines a handful of exemplary development environments used in MOOCs that teach programming to beginners.

2.1 Massive Open Online Courses

MOOCs are large-scale online courses that combine education, entertainment, and social networking [64]. They are typically free of charge and open to anyone who is interested. A MOOC is usually structured around a set of learning goals from a certain area of study, which are presented over a period of a few weeks. Topics are diverse and cover various field, including arts, CS, medicine, philosophy, and social sciences. In order to synchronize the learning process, MOOCs typically have a fixed beginning.

MOOCs are usually not for credit. However, they often award digital badges or certificates that reward participants for their accomplishments [13]. Despite these incentives, the percentage of enrolled MOOC participants who satisfy the criteria to earn a reward is typically low. The majority of courses have completion rates of less than ten percent [40].

While the three most prominent MOOC providers Coursera[1], edX[2], and Udacity[3] are all based in the United States, several European platforms, such as Future-

[1] `https://www.coursera.org/`
[2] `https://www.edx.org/`
[3] `https://www.udacity.com/`

Learn[4], Iversity[5], and OpenupEd[6], came into existence. By removing geographical location as a barrier, MOOCs have the potential to make high-quality education accessible for everyone with decent Internet connectivity. In this way, MOOCs can help people in the most remote regions of the planet to promote their careers and expand their intellectual and personal networks [64]. Additionally, learning becomes more flexible since online courses are not bound to fixed locations and daytimes [62]. While idealists consider MOOCs to be an educational revolution, able to provide free education to the disadvantaged around the world, research suggests that MOOCs may primarily favor people who are already educationally privileged [40].

A MOOC provides course materials plus assessment tools for independent studying [14]. A MOOC's content is usually packaged into videos of a few minutes in length, following a highly efficient approach to convey information, since videos are dense in information but can be paused and reviewed at any point [25]. Video content is complemented by reading material as well as optional lessons for highly ambitious learners. Lectures are usually separated by self-tests that enable learners to reflect upon the knowledge they gained. Graded assignments are used as a means of performance evaluation.

In order to fit the massive educational context, MOOCs' assignments must be evaluated in a scalable fashion. While modern web technologies enable scalable delivery of lecture videos as well as real-time learner collaboration and interaction, the abilities to evaluate complex student assignments remain limited [67]. As a consequence, MOOC assignments are usually restricted to quiz tasks that are easily automatically assessable, such as multiple-choice questions, fill-in-the-blank questions, and ordering tasks [93].

Depending on the openness of their content and learning process, MOOCs can be divided into two categories: xMOOCs and cMOOCs [31]. The majority of MOOCs belong to the group of xMOOCs. They follow a traditional classroom model, including a predefined schedule, instructor-led video lectures, and graded exercises. Furthermore, xMOOCs tend to use learning material with proprietary licenses. In contrast, cMOOCs are based on open educational resources and rely on learners who actively create knowledge and shape the learning process through their interactions. cMOOCs offer learners a more autonomous, self-organized, and social learning experience.

Most MOOC platforms offer collaboration tools, such as discussion forums, chats, and private groups, which enable course participants to discuss course contents with fellow learners and teaching assistants. These tools facilitate the creation of a virtual community of learners. Nevertheless, a common criticism of MOOCs is that they are unable to replicate the social experience of traditional classroom education due to their massive size and sole reliance on technology [91].

These days, two years after the New York Times labeled 2012 the "year of the MOOC" [64], MOOCs are a widely recognized phenomenon. They continue to grow and are expected to see a rise in student registrations [13]. Enrollment numbers up to six figures have attracted considerable media attention. However, such massive numbers are not representative for a typical MOOC. The average course has around

[4] `https://www.futurelearn.com/`
[5] `https://iversity.org/`
[6] `http://www.openuped.eu/`

43,000 enrollments [40]. As the number of available courses is increasing, individual participant counts are expected to decline on average.

2.2 openHPI

This thesis was created in the research context of openHPI[7], which is a non-profit provider of MOOCs, hosted by the Hasso-Plattner-Institut (HPI) in Potsdam, Germany. Current research around openHPI deals, among others, with the questions how to raise completion rates in MOOCs through gamification [92] and social engagement [26], how to integrate peer assessment into the grading process [66], and how to provide a flexible framework for learning analytics.

Introduction openHPI started in September 2012, making the HPI the first European university institute to offer interactive online courses [94]. The first course "In-Memory Data Management"[8] was offered by the HPI's founder and patron Hasso Plattner. Two months later, the second course "Internetworking with TCP/IP"[9] represented the first xMOOC held in German language. Since then, 13 different courses have been offered to a general audience, reaching enrollment counts in the lower five-figure range.

openHPI's online courses are adaptations of the HPI's on-campus courses. They cover CS topics, such as databases [61, 69], Internet technologies [94], and programming concepts [88, 58]. According to Meinel and Willems [56], openHPI courses typically follow the same didactic pattern: They have a fixed start date, are split up into six consecutive one-week modules, and are concluded with a final exam. After a course has ended, its contents usually remain accessible for self-paced learning.

Course content mainly comprises video lectures, reading material, optional self-tests for diagnostic assessment, as well as weekly mandatory homework for performance evaluation. Practical exercises and assignments, however, are rare but demanded. When the instructors of openHPI's first course asked for missing features, a remarkable number of users explicitly asked for practical tasks [93].

Practical Assignments During the evolution of the openHPI platform [55, 56], the capabilities for providing practical tasks have undergone considerable changes.

In the early days of openHPI, the platform did not support integrated hands-on exercises. Instead, some courses involved optional practical tasks, which had to be solved on students' personal computers (PCs). These tasks were not assessed [61] or were assessed indirectly through multiple-choice questions relating to the students' practical work [93].

Although practical exercises were optional and entailed a comparatively low reward for the effort to be taken, the vast majority of active course participants took the chance to participate [80]. A survey among the course participants revealed that about 80 percent of them considered practical exercises being useful for their learning outcomes. Also, the majority of course participants wished for more practical exercises, making them the most desired type of course content.

[7] `https://open.hpi.de/`
[8] `https://open.hpi.de/courses/imdb2012`
[9] `https://open.hpi.de/courses/internetworking`

However, the inhomogeneity among learners' hardware and software configurations resulted in many support requests, which required significant commitment and goodwill of the teaching team [93].

The subsequent course "Web-Technologien"[10] continued to leverage standard quiz assessment capabilities as a workaround for assessing practical exercises. However, instead of using learners' PCs as execution platform for practical tasks, the course focused on utilizing the freely available third-party tool JSFiddle[11]. Learners were invited to solve publicly available programming tasks, which yield a code word when solved correctly. Although this approach provides practical tasks in an installation-free and homogeneous environment, Staubitz et al. [80] consider it to be too limited and underline the need for a full-fledged integrated programming tool that provides an automated workflow, more complex programming tasks, and better feedback mechanisms.

In 2014, the original LMS-based openHPI platform has been replaced by a newly developed in-house solution. Since then, openHPI supports the integration of external special-purpose tools, which allow conducting workshops with a strong focus on hands-on tasks and practical exercises [88].

2.3 Practice in Programming Education

CS educators seem to agree that programming is best taught using a learning-by-doing approach. Learning to program does not only involve acquiring complex knowledge but also related practical skills [72]. Therefore, gaining programming expertise requires rigorous practice [85].

Programming assignments can help students to become familiar with programming languages and tools, and to understand how the principles of software design and development can be applied in practice [15]. On-campus programming courses usually make use of practical assignments that build up on theoretical content presented in lectures. These assignments are regarded to be an indispensable part of the educational framework [62] and are used for assessment by the majority of CS academics [80]. Feldman and Zelenski [19] believe that the major part of students' learning outcomes in a beginners' programming course originates from completing programming projects.

The most important deficits of novice programmers relate to designing problem solutions and express them as actual programs. Frequent practical programming exercises are a common way for addressing these issues [72]. A complete solution to a programming task is considered to be an important step in building the confidence of student programmers [15].

As introduced in Section 2.1, the majority of MOOCs are composed of static learning content, tools that facilitate social interaction, and quizzes, which are used for self-evaluation and assessment. As discussed earlier, the diversity of these quizzes is quite limited since grading has to be performed in a scalable fashion. Although multiple-choice questions can be helpful to reflect upon what the student has learned, this kind of assessment is no substitute for hands-on experience and practice when learning to program. According to Neuhaus et al. [62], the current generation of MOOC platforms is well suited for presenting teaching material, but it

[10] `https://open.hpi.de/courses/www`
[11] `http://jsfiddle.net/`

provides only inadequate possibilities for hands-on experiments. Supported assignments are essentially non-interactive and do not allow a step-by-step development of solutions.

However, in order to enable a more holistic learning process, MOOCs need to integrate activities that allow active experimentation and that relate to concrete experience [31]. Willems et al. [93] state that the implementation of systems that allow the assessment of practical exercises can be a great challenge for course creators and platform designers. Nevertheless, the authors see the ability to offer classes with a high share of practical tasks and assignments as a key feature of MOOC platforms, which will have a crucial impact on a platform's competitive position.

2.4 Web-based Applications

Within the last decades, the World Wide Web (WWW) has become a platform for full-fledged applications that provide desktop-like user experience (UX). The trend towards web-based applications is driven by the strongly evolved capabilities of web technologies and by the advantages that web-based delivery can offer application producers and users.

Web applications entail a set of inherent benefits. Given Internet connectivity, web-based applications are globally accessible on any device without prior installation and configuration. They can also be maintained and updated remotely at minimal cost. By using a web browser as execution platform, web applications are virtually independent of clients' system configurations. Apart from considering browser compatibility issues, a single codebase is sufficient for cross-platform delivery. The practically homogeneous software environment provided by web applications can also reduce the frequency of user support.

2.4.1 Web-based Development Tools

Not only end-user applications are affected by the trend towards web-based delivery, but also development tools find their way into the WWW. These tools are often based on embeddable JavaScript code editors, such as Ace[12] and CodeMirror[13], which offer rich code editing capabilities that are comparable to those provided by native desktop editors, such as TextMate[14] and Sublime Text[15].

Educational Tools

When attempting to start programming, novices may be hindered by the difficulties involved in installing and using the necessary tools [84]. The installation-free availability and ease of use of web-based applications make them a predestined means for providing development tools to beginners.

As briefly introduced in Section 1.1, state-of-the-art online programming courses teach programming with a strong focus on practical exercises. These courses facilitate the entry into programming by offering web-based development environments

[12] `http://ace.c9.io/`
[13] `http://codemirror.net/`
[14] `http://macromates.com/`
[15] `http://www.sublimetext.com/`

that allow solving programming problems within the web browser. Moreover, the employed tools usually provide step-by-step instructions and a compelling UX.

Similarly, sites such as Codewars[16] and CodingBat[17] supply collections of practical programming problems to be solved in the web browser. These tools do not provide a course framework, but they can support novices on their way to mastery by offering an engaging opportunity to practice.

Educational programming games are designed to maximize the appeal of learning to program. Learners' motivation is raised by using inciting game elements, such as increasingly challenging levels, scores, and leaderboards. CodeHunt[18] [83] is a web-based coding game, aimed at teaching programming at scale. It challenges students to complete skeletal methods, given in either Java or C#, so that they satisfy a hidden specification, which is only given by input/output (I/O) pairs. Similarly, Xiao and Miller [98] describe a many-player online programming game that is aimed at teaching novice CS students best practices for collaborative programming in large software projects.

Impromptu Development Tools

Even though websites such as CodePen[19], jsFiddle, and repl.it[20] have no primary educational objective, they provide developers with in-browser programming environments for impromptu development and execution of short programs. Such platforms' use cases include trying out libraries, constructing minimal programs for troubleshooting, and sharing code snippets.

Integrated Development Environments

Full-featured web-based integrated development environments (IDEs) are mentioned in research [2, 29, 90, 97], available as open-source software, and provided as hosted solutions, for example by Cloud9[21], Codio[22], and Nitrous.IO[23].

Web-based IDEs usually make use of traditional desktop user interface (UI) patterns, such as menu bars, file trees, content tabs, context menus, and drag-and-drop operations. Besides sophisticated code editing capabilities, such applications' features may include customizability, project management, version management, and full Linux environments for building and executing applications. Since computationally intensive tasks are performed on a remote server, low-end PCs and mobile devices can be used as development machines.

Web-based IDEs often facilitate the deployment of applications to infrastructures supplied by Platform as a Service (PaaS) providers, such as Google App Engine[24], Heroku[25], and Microsoft Azure[26]. Therefore, anybody with modest soft-

[16] `http://www.codewars.com/`
[17] `http://codingbat.com/`
[18] `https://www.codehunt.com/`
[19] `http://codepen.io/`
[20] `http://repl.it/`
[21] `https://c9.io/`
[22] `https://codio.com/`
[23] `https://www.nitrous.io/`
[24] `https://appengine.google.com/`
[25] `https://www.heroku.com/`
[26] `http://azure.microsoft.com/`

ware development skills is able to deploy applications to the Cloud with small effort and low budget [2].

Another feature that is predestined for web-based IDEs is collaborative editing, as known from Etherpad[27] and Google Docs[28]. Multiple developers who are working at the same time are provided with a consistent view of a project since they receive real-time updates of their collaborators' changes. Collaborative coding facilitates side-by-side pair programming, benefits communication and team knowledge sharing, and may increase productivity and software quality [29].

2.5 Assessment

As discussed in Section 2.3, practical programming exercises are an important component of CS education. According to Pieterse [68], offering assignments as opportunities to practice is essential for students in order to develop programming skills. However, assignments are considerably more valuable if fast and accurate feedback is supplied. Assessment allows providing both learners and teachers with feedback about the learning process and is essential to guide students' learning [38].

Traditionally, teachers perform assessment manually. However, manual assessment of code submissions is a time-consuming and error-prone activity that involves a time delay before learners receive feedback. Beyond that, educators' time is limited and may be spent better on other aspects than assessing students' work.

Student evaluation is particularly a challenge for courses with many participants. While manual grading might be realizable for on-campus courses using a sufficient number of teaching assistants, huge student numbers in large-scale e-learning environments make manual grading performed by the teaching team infeasible [77]. MOOCs, which aim at providing unlimited participation, face a scalability problem in this regard [73]. In order to provide assessment to an open-ended number of learners, MOOCs have to employ scalable assessment approaches, such as automated assessment and peer assessment.

2.5.1 Automated Assessment

Automated assessment refers to assessment approaches that are solely based on automatable workflows and automatically collectable properties. Automated techniques have been used for the assessment of programming assignments almost as long as programming has been taught [68].

Advantages

On campus, automated assessment approaches are used to keep teachers' workload within reasonable limits despite growing students numbers [87]. This way, the time required for assessment activities can be cut down without reducing quantity and quality of practical exercises. Furthermore, the amount of time that instructors can spend on mentoring and supporting students is increased [86].

Automated program evaluation can also be beneficial to students. While human graders and especially teams of multiple graders usually judge subjectively and

[27] `http://etherpad.org/`
[28] `https://docs.google.com`

inconsistently, automated assessment can provide objective and consistent evaluation [3]. Furthermore, students are provided with immediate feedback, which is an important benefit in programming education. Receiving instant feedback is particularly useful for novice programmers since misconceptions are uncovered as early as possible [89]. The concept of providing feedback at any time and any place applies notably well to virtual courses [51], such as MOOCs, and provides learners a unique advantage [13]. Since assessment resources are virtually unrestricted, automated grading allows students to increase mastery by iteratively improving and resubmitting their homework [25].

Evolution

Systems that automatically assess students' programming assignments have been designed and used for over fifty years. Systematic overviews of assessment systems' approaches and capabilities have been published by Ala-Mutka [3], Douce et al. [15], and Ihantola et al. [38].

Douce et al. present a historical overview of automated assessment systems that focuses on systems that are based on executing tests in an automated fashion. The authors classify these systems into three generations.

The first generation covers the initial attempts to automate the assessment of programming assignments. In general, first-generation systems were specifically tailored solutions that required modifications to compilers and operating systems (OSs), demanded a great deal of expertise, and were limited to the usage in their particular setting. The very first system has been described by Hollingsworth [36]. Its purpose was to evaluate programs written in assembly language, which had to be handed in on punched cards. The system was not only useful for saving teacher resources but also for allocating computing resources, which were severely limited at that time.

The second generation of automated assessment systems is characterized by the adoption of automated tools and utilities, provided by increasingly advanced OSs and tool sets. The systems could be operated by instructors and students using a command-line interface (CLI) or a graphical user interface (GUI). Second-generation systems introduced more sophisticated assessment strategies involving multiple assessable properties, such as correctness, efficiency, and style. Several systems also include management capabilities for courses and assignments. Well-known representatives of second-generation systems are ASSYST [39], BOSS [41], and Ceilidh [10].

Third-generation automated assessment systems took advantage of advancing web technologies. They comprise features such as web interfaces, increasingly sophisticated testing approaches, interactive feedback, richer content management features, and plagiarism detection. Systems of the third generation include instances of second-generation systems that continued to develop, such as BOSS, and successors of former systems, such as CourseMarker [35], which evolved from Ceilidh.

While Douce et al. assign state-of-the-art automated assessment systems to the third generation, their work cannot cover trends that emerged after 2005, for instance the growing demand for practical assignments in e-learning. In that respect, Ihantola et al. [38] report an increasing interest in extending LMSs with automated assessment capabilities in order to fit the special needs of CS education better. For

the same reason, programming MOOCs should be provided with modern capabilities for automatic code assessment.

Design Challenges

High-quality assignments are seen as a vital part of a successful course [19]. While manual assessment allows compensating for poor assignment design, the use of automated assessment techniques increases the need for carefully designed assignments [68]. The creation of automatically assessable programming assignments is considered a challenging task that requires special attention [3].

Whereas automated assessment saves instructors' time by outsourcing formerly manually performed grading activities, a considerable amount of the gained time should be allocated for designing and implementing resources for automated assessment. While efforts may only be shifted from grading activities to design activities for small class sizes, the trade-off increasingly shows its strengths with rising student numbers.

Whenever assessment is performed without human intervention, the assignment specification should be provided as unambiguous as possible. Ambiguous specifications permit different interpretations, which can lead to technically valid student solutions being rejected by an automatic grader. Within programming, interpretation is key to success, which is why assignment instructions must guide interpretation precisely for successful automated assessment [15]. In contrast, careless formulation of assessment criteria can result in improper assessment [68]. Therefore, ambiguity must be minimized in order to increase fairness and quality of assessment [71]. Cerioli and Cinelli [12] even regard an extremely precise problem specification, which allows a completely predictable behavior of implementations, as a prerequisite for automated grading based on functional correctness. However, a reasonable balance between the risk of misinterpretation and excessive detail has to be found because wordier specifications, which point out every detail, can result in trivial assignments lacking any demand to reason about the problem [71].

Besides addressing the problem of ambiguity, the definition of pedagogically sound test cases is a time-consuming activity [12] that requires both expertise and experience [85]. Pieterse [68] names test data "the Achilles' heel of any system that applies automated assessment of programming assignments". In order to enable accurate assessment and prevent incorrect solutions from passing the evaluation, tests must be designed well. Otherwise, learners might submit deficient solutions but remain unaware of their incorrectness.

Approaches

There are several approaches for performing automated assessment of programming assignments. They can be split up into dynamic approaches, which require execution of the program under test, and static approaches, which do not. While most approaches focus on evaluating the functional completeness and correctness of a program, others aim at evaluating aspects of quality and style.

I/O-based Assessment I/O-based assessment refers to assessing a program solely by using a standard I/O interface. The program under test is supplied with predefined values and is verified to produce expected output values.

The advantage of this approach is its versatility. I/O-based assessment can be applied to any program using an I/O interface and to any programming language that can be executed on the same test environment [38]. Moreover, test cases may be reused across multiple languages since a universal interface is sufficient for their execution.

A shortcoming of the approach is that it may fail to give an appropriate mark if a student program's output does not exactly match the expected format [68]. Therefore, I/O-based assessment techniques are not usable if strict format requirements are not feasible or if freedom in formatting should be allowed. However, implementing I/O handling that is robust to irrelevant output differences, such as whitespace and orthographic mistakes, is a challenge [15].

Due to lacking insights into the inner mechanics of a code submission, I/O-based assessment is limited to testing side effects that are exposed in the form of program output. For the same reason, I/O-based assessment is not qualified for providing the learner with feedback regarding why her submission deviates from the specification.

Assessment Using Industrial Testing Tools In-depth feedback can be provided by utilizing industrial-strength testing tools and frameworks. Such tools are widely used, are actively developed, and can supply deeper insights into the program under test. Since testing is an established practice in industry, myriads of testing frameworks exist for virtually every programming language and application domain.

Ihantola et al. [38] name three classes of industrial testing tools that are used by automated assessment systems: xUnit-based frameworks, acceptance testing frameworks, and web testing frameworks.

xUnit is a collective term for numerous testing frameworks that derive their design from SUnit [7], an influential testing framework for Smalltalk, which is considered "the mother of all unit testing frameworks" [16]. Widespread xUnit derivatives include CUnit[29] for C, HUnit[30] for Haskell[31], and JUnit[32] for Java. These language-specific testing frameworks enable assessment techniques that can evaluate the functionality of entities smaller than a complete program, such as single classes, methods, and even statements [3].

Acceptance testing refers to an industrial testing technique that is based on customers specifying test scenarios that have to be passed so that user stories are considered to be correctly implemented. This testing approach helps customers and developers to foster a common understanding of how software under development should work once it is finished. Acceptance testing frameworks, such as Cucumber[33], FitNesse[34], and Lettuce[35], usually rely on easily understandable plain-text domain-specific languages (DSLs), similar to natural language. This allows non-technical stakeholders to contribute their domain knowledge by providing scenarios that specify navigation through the application, inputs to the application, and expected outputs [24]. Scenarios are turned into executable tests whose successful

[29] `http://cunit.sourceforge.net/`
[30] `http://hunit.sourceforge.net/`
[31] `https://www.haskell.org/`
[32] `http://junit.org/`
[33] `http://cukes.info/`
[34] `http://www.fitnesse.org/`
[35] `http://lettuce.it/`

execution is to be achieved. When used for student assessment, acceptance testing offers the advantage that a single specification can serve as both assessment basis and exercise instructions since it is given in easily understandable form and expected to be complete.

Web testing frameworks, such as Selenium[36] and Watir[37], are useful tools for assessing web application exercises. Instead of accessing low-level application programming interfaces (APIs), web testing tools test web applications using their public web interfaces. This can either be done by controlling a real web browser in an automated fashion or by simulating a web browser by means of Hypertext Transfer Protocol (HTTP) requests.

Assessment of Testing Skills Modern software development processes, such as Scrum [75] and Extreme Programming (XP) [8], promote test-first practices, which help to discover design flaws as early as possible in the development cycle and underline the value of regression tests for continuous delivery.

When novice programmers are assessed using traditional automated approaches, they are neither encouraged nor rewarded for performing testing themselves since an automated grader verifies their programs' correctness anyhow. As a result, learners might not reflect upon the behavior of their code, but they might solely focus on providing a solution that satisfies the automated approach [17]. However, efficient automatic testing approaches should not invite students to get careless. Instead, students should learn to design and test their programs thoroughly before submitting them [3]. In this sense, Edwards [17] argues that students need to acquire software testing skills. He suggests exposing students to test-driven development (TDD), so that they perform more testing and eventually appreciate its value for the development process. Moreover, a testing-oriented assessment approach empowers students with the responsibility of demonstrating their own programs' correctness and validity. As a result, the learning experience is enhanced and learners produce higher-quality code. According to Pieterse [68], the application of test-based assessment combined with training in software testing can provide a learning experience where students learn to favor robust and precise solutions over improvised ones.

The term meta testing refers to a test-based assessment approach that evaluates students' software testing skills. Instead of providing learners with prepared tests, be it explicitly as visible part of an exercise or implicitly as the basis for program evaluation, this approach demands learners to write tests themselves. They are required to submit working program code along with proper tests. Grading can be based on judging the extent to which the student-written code fulfills the accompanying tests, the tests' level of quality, and the fraction of code covered by tests. Additionally, the teacher might incorporate her own tests into the assessment in order to validate that the student's submission indeed satisfies the exercise specification.

Assessment of GUI Applications Even though focusing solely on CLI applications may be perfectly sufficient for conveying programming skills, such an educational approach may be seen as uninspiring by learners to whom graphical applications are familiar and much more attractive than CLI-based ones [15]. Instead, students are interested in learning how to build programs with GUIs [18].

[36] `http://www.seleniumhq.org/`
[37] `http://watir.com/`

Likewise, applications that produce animations or perform 3D rendering are usually appealing to learners.

However, GUI applications are difficult to assess since I/O redirection, as used for the assessment of CLI applications, is not applicable. Developing software tests for programs involving significant GUIs is ranked beyond the typical abilities of students and educators [82].

A response to this problem are educational GUI libraries, such as presented by English [18] and Thornton et al. [82]. These libraries are designed for novice programmers and provide built-in means for automated testing and assessment. Moreover, the latter framework is explicitly aimed at allowing students to write tests themselves. Therefore, it can enable automated assessment for GUI applications that follows a TDD-based assessment approach.

The edX course "Foundations of Computer Graphics"[38] provides scalable assessment of graphically sophisticated student programs, based on image comparison. Visual output generated by a learner's program is exported to an image file, uploaded by the learner, and automatically compared with a reference image. In order to be considered a correct solution, the output of the learner's program must not vary considerably from a reference image. The degree of permitted deviation is controlled using a threshold of pixels that are allowed to differ.

Assessment of Style Besides functional completeness and correctness, there are further aspects that are crucial to the quality of software, such as its complexity, extensibility, and maintainability.

Writing code in good style is important because program code is read much more often than it is written [73]. Since software projects are usually carried out in groups, developers need to follow established coding conventions that facilitate a common understanding among them and guarantee a certain degree of quality. In general, good coding style promotes readability, absence of errors, security, extensibility, and modularity [73].

However, novice programmers are reported to commonly perceive programming style as less significant [4] and to have little appreciation for best practices, which are required for successful long-term multi-person programming projects [98]. Therefore, programming style is an important issue to teach beginning programmers. It is often neglected in education, though [4].

Automated techniques can help to involve programming style into the assessment process. In contrast to functionality, which is usually assessed by executing a program submission, properties of style are typically collected using static evaluation approaches.

A common practice for judging a student program's quality is detecting so-called code smells, such as unused variables, redundant logical expressions, and implicit constants [84]. Furthermore, automated style evaluation can examine programs' adherence to given coding guidelines in terms of indentation size, mandatory source code documentation, and more [4]. High-complexity program submissions can be detected by employing software metrics, such as Halstead's complexity measures [33] and McCabe's cyclomatic complexity [54], and by comparing their structure to that of a model solution [84].

[38] `https://www.edx.org/course/foundations-computer-graphics-uc-berkeleyx-cs-184-1x`

However, there are program characteristics that are hard to assess automatically, for instance quality of comments, meaningfulness of variable names, and adherence to good practices, such as the Single Responsibility Principle [53]. Evaluating such subtle or complex software properties requires the trained eye of a human assessor.

2.5.2 Peer Assessment

A different approach towards scalable assessment is peer assessment. Instead of relying on automation, peer assessment involves learners into the assessment process. Students' submissions are graded by peers who take the same course, usually as part of mandatory course tasks. Since learners cannot provide expert grading, every student solution is typically graded by three to seven peers. The final grade is determined by aggregating the individual scores using basic statistical means or more sophisticated probabilistic models [67].

Peer assessment is a popular choice for assessing subjective topics and complex problems for which accurate machine grading is difficult to provide [77]. With this in mind, peer assessment seems to be a qualified means for assessing subtle properties of programming assignments. Peer assessment for programming assignments is based on the philosophy that learning to program should be a social experience emphasizing human-human interaction [47]. In this sense, it is similar to code review among colleagues, which is a common procedure for quality assurance in industrial software engineering and open-source software development.

Peer assessment can overcome some limitations that apply to automated assessment. Automatic approaches can only evaluate clearly defined problems with specified interfaces. They cannot award points for creative solutions, though. Peer assessment can be a valuable tool for issues that are hardly automatically testable [74]. While some quality metrics can be assessed automatically, subtle software properties, such as the qualification of a chosen approach, the elegance of an implementation, portability, and reusability, are more difficult to quantify [15]. Peer assessment also offers the possibility to provide more and better feedback for individual student solutions [74].

The requirement for learners to grade their peers' program submissions should not be seen as a burdensome duty but as an opportunity. Seeing fellow students' approaches to the same programming problems may provide new ideas and reveal different paths to a viable solution [34, 70]. Moreover, students who are familiar with peer assessment might improve the readability of their programs since they are reviewed manually [34]. Lastly, giving peer reviews is reported to increase overall learning outcomes [70] and improve the learning process due to a deeper occupation with the subject [74].

However, peer assessment also has some shortcomings. The time it takes to receive quality feedback is much longer than with automated assessment techniques. Moreover, in contrast to objective results provided by automatic graders, peer assessment involves the potential for biased and inaccurate feedback. This applies especially to beginners' courses where the majority of students are struggling with the course content [78].

2.6 Development Environments

MOOCs that cover programming topics are numerous and multifaceted. Based on contents and target audience, courses follow different approaches for providing learners with the means to write and execute code. Approaches can be divided into desktop solutions and web-based solutions.

2.6.1 Desktop Solutions

Desktop development tools include standard IDEs, which are used by professionals, and educational tools, addressing beginners.

Industrial Development Tools

Some CS educators believe that novice programmers should use practices and tools that are similar to those used in the software industry [86]. Learners are required to utilize industrial-strength development tools that are freely available, such as Eclipse[39], IntelliJ IDEA[40], and NetBeans[41]. This approach is common when on-campus courses are transformed into MOOCs.

Relying on standard IDEs has the advantage that students become familiar with established development tools, which can be useful in contexts other than a single MOOC. Moreover, full-fledged desktop IDEs almost always provide built-in debugging and testing capabilities, customizability, and advanced features that promote productivity, such as automated source code refactoring.

In order to bring desktop IDEs and MOOCs closer together, IDEs' common extension capabilities can be leveraged to implement connecting features, such as download of programming assignments, transmission of analytical data, and submission of finished exercises to an assessment server [86]. Without a proper coupling, using desktop IDEs in MOOCs might necessitate a development workflow that is heavily based on downloading, unzipping, zipping, and uploading code, which is a procedure that does not belong to the best practices of professional software engineers and that should not be taught to beginners [86].

Programming MOOCs that rely on desktop IDEs entail the disadvantage that even absolute beginners are put under the obligation to download, install, and configure software that is built for experienced programmers. This may also require installing third-party dependencies, such as software development kits (SDKs). However, installing and configuring an IDE requires a level of digital literacy that cannot be expected by all participants of a MOOC.

Liyanagunawardena et al. [49] describe their experiences during a MOOC teaching Android[42] programming to complete beginners, which required participants to set up several software components. Due to lacking skills in setting up software, the teaching team had to support many learners in order not to lose them at the beginning. Some of them were not even able to meet the challenge of unzipping a compressed archive.

Moreover, solving practical assignments on learners' PCs can create "a challenging inhomogeneity amongst the involved machines, operating systems and network

[39] `https://www.eclipse.org/`
[40] `https://www.jetbrains.com/idea/`
[41] `https://netbeans.org/`
[42] `https://android.com/`

infrastructures" [93], which makes it close to impossible to identify all potential issues for single users during exercise design. With tens of thousands of students, troubleshooting of individual problems is not feasible. However, minor problems can build a significant barrier to success for beginning students, keeping them from starting programming [47].

To circumvent installation problems and platform peculiarities, a fully configured desktop development environment can be provided in the form of a virtual machine (VM) image or recipe, using cross-platform virtualization tools, such as Vagrant[43] and VirtualBox[44]. This approach supplies learners with a homogeneous development environment and limits setup tasks to a minimum. However, virtualization software still has to be installed, which might pose a challenge for some learners. Moreover, different OSs and virtualization software versions introduce inhomogeneity into the concept. Course providers should also consider that some students' PCs might be too slow for running a VM [25].

Educational Development Tools

Educational programming environments, such as BlueJ[45] [44], Hackety Hack[46], Processing[47], and Squeak Etoys[48] [42], aim at making programming easily available for beginners by providing all-in-one packages that supply development tools and an execution platform. Such environments are usually easy to install or available as installation-free, portable packages. Therefore, they require only basic computer skills to start programming.

However, the approach of using educational development environments in the context of a MOOC is restricted to the few ones that are available. Besides, these tools are often limited to built-in capabilities and might therefore not fit all requirements of a course.

2.6.2 Web-based Solutions

As discussed in Section 2.4, web-based development tools provide homogeneous, installation-free development environments. By eliminating the need for setup and configuration, they lower students' barriers to start programming [87]. Since participants of a MOOC already have access to a web browser, web-based development tools are virtually predestined in this context [99]. Furthermore, the web-based nature of MOOC platforms enables a tight integration of web-based special-purpose tools.

Web-based development environments can either be provided by bringing dedicated tools into operation or by leveraging third-party tools that are already existent.

Dedicated Development Tools

Dedicated development tools are supplied as tightly integrated parts of e-learning platforms. They can be distinguished based on their approach for the execution of

[43] `https://www.vagrantup.com/`
[44] `https://www.virtualbox.org/`
[45] `http://www.bluej.org/`
[46] `http://www.hackety.com/`
[47] `http://www.processing.org/`
[48] `http://www.squeakland.org/`

learners' code. Student-written code is either executed in the client's web browser or transmitted to the server for remote execution.

Client-Side Code Execution Executing a learner's code on her own machine is a resource-efficient approach since no server-side resources are claimed for code execution. Furthermore, there is no need for security considerations in terms of dealing with potentially untrustworthy code. Moreover, since no client-server round trips are involved, client-side code execution promotes interactivity and avoids potential delays during high-demand periods before assignment deadlines [47].

Using the learner's web browser as execution platform is particularly suitable for teaching client-side web technologies, such as Hypertext Markup Language (HTML), JavaScript, and Cascading Style Sheets (CSS), since interpreters for these languages are built into browsers.

The major drawback of client-side code execution is its limitation to browser-supported programming languages and APIs as well as special JavaScript-based derivatives of non-native languages, such as ClojureScript[49], Opal[50], and Skulpt[51], which are in-browser implementations of Clojure[52], Ruby[53], and Python[54].

Server-Side Code Execution Compared to its client-side equivalent, server-side code execution offers much more flexibility since the set of executable programming languages is virtually unlimited. Moreover, code evaluation for both exploration and assessment is performed in one place and can use the same procedure. Additionally, sending partial solutions for execution to the server allows reproducing the iterations of a learner's development cycle and can provide valuable insights into students' problem-solving strategies.

The advantages of server-side code execution come at the cost of increased computational load and feedback latency. Furthermore, careful security considerations are necessary.

Third-Party Development Tools

Web-based development tools can be realized without the need for self-hosted solutions. Instead of providing dedicated development environments and allocating platform resources, programming MOOCs can leverage third-party services for several or even all aspects of the development process [24, 80]. Software as a Service (SaaS) and PaaS providers typically offer free plans for starters, which fit the needs of MOOC participants and can provide the tools that are needed for practical programming assignments. For instance, novice programmers' demands could be covered based on third-party services by leveraging Cloud9 as a web-based IDE (see Section 2.4.1), GitHub[55] for code hosting and issue tracking, Heroku as execution platform, and Travis CI[56] for continuous testing.

[49] `http://clojurescript.net/`
[50] `http://opalrb.org/`
[51] `http://www.skulpt.org/`
[52] `http://clojure.org/`
[53] `https://www.ruby-lang.org/`
[54] `https://www.python.org/`
[55] `https://github.com/`
[56] `https://travis-ci.org/`

Not only does this approach save the resources of the e-learning platform, but it also enables learners to gain practice in working with tools and services that are used by professionals. According to Fox and Patterson [24], deploying their projects in the same scalable environment as used by professional developers supplies learners with valuable experience. Moreover, the approach can provide students a feeling of accomplishment when shipping working code that can be used by people other than their instructors.

Relying on freely available online services involves the drawbacks that learners are required to register with third-party companies, that individual tools are spread over different platforms, and that MOOCs following this approach are highly dependent on the availability and reliability of external parties.

2.7 Programming Tools Used in MOOCs

In order to determine the status quo of programming tools used in MOOCs, we regarded a number of courses offered by the major international MOOC providers Coursera, edX, and Udacity, as well as two German MOOC providers, which are Iversity and openHPI.

2.7.1 Usage of Quiz Capabilities

Besides offering practical tasks, many programming courses employ standard built-in assessment capabilities, such as fill-in-the-gap and multiple-choice questions, for teaching programming. For example, learners are presented with some lines of code and are asked to identify mistakes or to reason about the state of variables after execution. For the sake of automated assessability, a number of possible answers are given, which the learner has to pick the correct ones from.

Such quiz-based exercises can only supplement the content of a programming course. While tasks of this nature require learners to understand code written by others and to reason about its behavior, there is no creative problem-solving process involved. However, for learning programming it is important that students program by themselves. According to Lahtinen et al. [47], course materials should rather have a problem-solving nature than only representing concepts since success in creating a functional program is a major positive force for programming beginners.

2.7.2 Comparison of Specialized Programming Tools

This section presents a comparison of programming tools used in exemplary MOOCs with regard to approach, functionality, and usability.

Python, being considered as an intuitive, powerful, and easy to learn programming language [11] is taught in half of the regarded courses due to its suitability as a beginners' language. In fact, Python has recently become the primary language for teaching introductory CS courses at top-ranked universities in the United States [32]. In line with this, almost all of the regarded MOOC providers offer introductory programming courses in Python.

Coursera: "Programming for Everybody"

Coursera's course "Programming for Everybody"[57] is specifically designed to be a first programming course. It teaches the Python programming language.

The course makes use of an open-source, web-based development environment[58] for writing and assessing practical programming exercises. The tool is based on CodeMirror and Skulpt, an in-browser implementation of Python, providing client-side code execution.

The development environment's UI contains exercise instructions, a single editable code area, an area for program output as well as buttons for running and resetting code.

Since no request to the server is required, the tool's client-side code evaluation approach has the advantage of short response times. However, we experienced that infinite loops are not identified and suspended after a certain time but occupy client-side computing power, eventually rendering the browser window unresponsive.

Apart from this, program errors are reported using native browser alert dialogs. Error messages are reduced to the essential. They neither contain a traceback nor provide additional clues to the error's origin.

Whenever code is executed, it is also checked against the exercise specification. The programming tool performs automatic grading, based on I/O matching and basic invocation checks at runtime. The grading approach can only grant full score or zero points. Partial solutions, however, are not awarded.

Besides auto-graded programming assignments, Coursera's course contains two optional peer-graded essays.

edX: "Engineering Software as a Service"

The course "Engineering Software as a Service"[59], offered by edX, teaches the fundamentals of developing SaaS using agile techniques and Ruby on Rails.

The course does not make use of a web-based development environment, but it relies on desktop applications and third-party infrastructure. To facilitate the start, the instructors provide learners with a VM image. Alternatively, they propose to use a web-based IDE. Moreover, the instructors encourage students to perform remote pair programming for solving the course assignments.

The course advances from the basics of Ruby, through introducing Rails, to teaching behavior-driven development (BDD) and TDD. Analogous to the course content, practical assignments range from writing short Ruby programs, through extending a Rails application and deploying it to Heroku, to implementing features that have to be backed by passing unit tests.

Exercise solutions are handed in by uploading code or entering the uniform resource locator (URL) of the deployed application, respectively. Viewing the exercise specification, submitting a solution, and receiving grading feedback are facilitated through a one-page workflow.

In order to assess practical exercises, the course utilizes a tailor-made automatic grader[60], which is based on established Ruby development tools for BDD, TDD, and quality control. The grading feedback that is provided to learners equals the

[57] `https://www.coursera.org/course/pythonlearn`
[58] `https://github.com/csev/tsugi`
[59] `https://www.edx.org/course/engineering-software-service-uc-berkeleyx-cs169-1x`
[60] `https://github.com/saasbook/rag`

standard output of the RSpec[61] testing framework, including entire backtraces in the case of errors. Although RSpec examples are usually defined in a meaningful, readable manner, unaltered framework output might not be sufficiently clear for novices in search of remaining code deficiencies.

Udacity: "Intro to Computer Science"

Udacity's course "Intro to Computer Science"[62] is an introductory CS course that teaches Python by the example of building a basic search engine.

The course makes use of a lightweight web-based development tool, which is based on CodeMirror and seamlessly integrates into the Udacity platform.

The editor is easy to operate, but it provides no means for editing more than one unit of code. Exercise instructions are provided as comments in the skeleton source code and sometimes in the form of a short introductory video. Buttons below the editor enable learners to start the exercise from scratch, execute their code for exploration, submit their code for evaluation, recapitulate the instructional video, or view a sample solution, which is presented in a step-by-step fashion.

When an error occurs during program execution, Python's standard traceback is presented, which might be too cryptic for beginners. However, when the same error occurs during test execution, basic hints pointing to corrective actions are provided, though not adjusted to include actual identifiers from the student's code. The result of successful test-based code assessment is briefly presented in natural language, which benefits comprehensibility but lacks valuable details, such as expected and actual program behavior.

Iversity: "Algorithmen und Datenstrukturen"

Iversity's course "Algorithmen und Datenstrukturen"[63] teaches algorithms and data structures using the Java programming language.

Although the course addresses beginners, a web-based code editor is not provided. Instead, course participants are required to write, compile, and run code on their PCs.

The course employs a hybrid assessment approach. Hands-on assignments are assessed using both I/O-based automated assessment and peer evaluation. In the case of errors, the automated assessor does not provide guidance but only non-specific proposals to review the submission for syntactical errors, infinite loops, and faulty recursion.

Apart from this, programming assignments entail other shortcomings in usability. If a program skeleton is provided, it has to be copied manually from the exercise instructions. Likewise, code to be evaluated by peers has to be pasted into a text input area although already submitted as a ZIP archive for the purpose of automatic evaluation. Furthermore, evaluation results are not displayed within the submission form, but they must be accessed separately as soon as available.

[61] `http://rspec.info/`
[62] `https://www.udacity.com/course/cs101`
[63] `https://iversity.org/de/courses/algorithmen-und-datenstrukturen`

openHPI: "Spielend Programmieren lernen"

openHPI's very first special-purpose tool for practical programming exercises is used in the beginners' Python programming course "Spielend Programmieren lernen"[64]. The course covers basic programming constructs within a four-week timespan and is primarily aimed at school children.

Programming exercises are performed using a web-based development tool that is started from the openHPI platform. The tool's UI is rather simple. Its window contains plain-text exercise instructions on the top as well as some hints on how to use the tool in the bottom. The greater part of the window is split up into a code editor and an area for program output. The code editor is based on CodeMirror and is limited to a single editable area, which might be sufficient for short programs to be expected in a beginners' course.

One button allows to execute the program. A second button triggers evaluation for correctness. Learners can execute and validate their code as often as desired. When an error occurs, the standard stack trace is presented to the student. If a solution is free of programming errors but does not fulfill the exercise specification, the tool provides textual feedback in a stepwise fashion. For example, in order to solve an exercise, the tool suggests to define a certain method, to call that method, and to store the result into a variable that is checked during program evaluation.

Assessment is based on automated evaluation. The grader does not grant points for partial solutions. A participant can either receive the full score or no points at all.

Von Löwis et al. [88] report that the Python course was well received and gained very positive feedback. Nevertheless, users reported problems with the automatically graded programming exercises. Learners either felt that assessment was too restrictive or would have preferred an explanation as to why their solutions are incorrect. Also, programming errors that were caused by participants were sometimes not recognized as self-inflicted but were perceived as flaws in the system.

openHPI: "Parallel Programming"

Whereas the first iteration of openHPI's course "Parallel Programming"[65] included only optional programming tasks, which had to be executed on learners' PCs and were not assessable, possible future iterations should feature assessable hands-on exercises.

Müller [58] presents a web-based programming evaluation system that is targeted at being used in this context. The system is an extension of OpenSubmit[66], an assignment management application used in the context of some of the HPI's on-campus courses. The tool is primarily designed as an execution platform providing resource fairness, performance isolation, and multi-threading support, which are important characteristics for practical assignments concerning parallel programming techniques.

In terms of assessment, the web application provides a front-end for submitting complete program solutions, which are checked for successful compilation and are optionally validated using a custom validation script. The system can neither provide supportive feedback nor fine-grained assessment beyond a binary decision.

[64] `https://open.hpi.de/courses/pythonjunior2014`
[65] `https://open.hpi.de/courses/parprog2014`
[66] `https://github.com/troeger/opensubmit`

While the system follows a flexible code execution approach, content management capabilities offer potential for improvement. The preparation of exercises and OS templates is reported to be a manual, tedious task [58].

Chapter 3

Concept

This chapter covers the concepts that Code Ocean is based upon. At first, Section 3.1 states the requirements of the system. Subsequently, Section 3.2 deals with our decision in favor of a web-based development environment. Section 3.3 describes the secure and flexible approach for executing learners' code submissions. Section 3.4 discusses why learners should be supplied with helpful hints and outlines Code Ocean's method for providing them. Section 3.5 presents the selected approach towards the assessment of code submissions. Finally, Section 3.6 states the need for interoperability with existing e-learning tools and introduces Code Ocean's approach in this regard.

3.1 Requirements

We identified a number of requirements that Code Ocean should fulfill. Most of them also apply to large-scale programming education solutions in general.

Versatility Many programming tools used in the MOOCs that we regarded are designed for a single use case. In contrast, in order to fit the needs of a wide range of programming courses, our system should not be tailored to a specific set of programming languages, but it should be designed for supporting a variety of programming languages and application domains. Furthermore, teachers should be able to realize practical assignments that make use of third-party applications and libraries. As a consequence, Code Ocean's approach for program execution must be chosen with flexibility in mind.

Flexibility is also important in terms of assessment. According to Pieterse [68], the quality of automated assessment is largely dependent on the quality of test cases used. Therefore, Code Ocean should promote teachers' creativity in assessment design by providing them the freedom to decide which program aspects to assess and which tools to use for this purpose. Our application should not dictate a universal assessment approach but should permit the usage of any desired tool that fits the particular use case best, such as an established testing framework or a tailor-made solution.

Novice-Friendliness The large diversity of MOOC participants implies that classmates lack a common knowledge base and educational background [64]. There-

fore, learners' prior knowledge and digital literacy vary considerably. Since Code Ocean should be usable for teaching programming to complete beginners, we want to provide learners with a homogeneous development environment that has a simple and appealing UI, requires no prior knowledge, and supports them in many aspects of their endeavor to learn programming. Code Ocean should also minimize the challenges that the usage of an automated assessment tool may entail [68].

A very important aspect of the learning process is feedback. Feedback towards assignments allows students both to understand their mistakes and to revise their work [51]. Compared to a traditional learning setting, feedback quality is even more important in MOOCs because communication opportunities are limited [68]. Since both MOOCs and programming courses in general are affected by high dropout rates [47], we see providing students with understandable and useful feedback as crucial for their long-term motivation and as an important requirement of our application.

Scalability As MOOCs are aimed at unlimited numbers of participants, they need to be inherently scalable [85]. In our specific use case, many students must be able to write and execute code in parallel. Moreover, a certain level of responsiveness is required in order to achieve a satisfying UX [9]. Therefore, Code Ocean should follow a code execution approach that provides fast feedback and that scales for the number of users to be expected in a MOOC.

These scalability requirements also apply to assessment. As discussed in Section 2.5, huge enrollment numbers in MOOCs make manual feedback and grading impossible. Instead, Code Ocean should provide a scalable assessment approach that fits the needs of large-scale education.

Security Server-side execution of student-written programs implies that arbitrary code is executed within the boundaries of an e-learning system. This constitutes a considerable risk. Faulty student programs could excessively consume server resources; intendedly malicious programs could try to cause damage or obtain unauthorized access. In fact, automated assessment systems that are integrated into LMSs are considered a tempting target for attackers [38].

Due to these risks, providing a secured execution environment for running students' programs is regarded to be an essential requirement for automated assessment systems that employ dynamic evaluation techniques [3]. Code Ocean should provide means for the sandboxed execution of learners' code that guarantee that untrusted code can neither harm the platform nor influence other learners' code submissions.

Interoperability We believe that educational programming platforms are more widely adopted if they can be integrated into existent e-learning infrastructures easily. Therefore, our tool should be interoperable with existing e-learning systems, such as LMSs and MOOC platforms, including openHPI.

In order to extend their courses' contents with practical programming tasks, instructors should be able to prepare assignments on the Code Ocean platform and embed them into their courses. Learners, on the other hand, should be able to solve these assignments in a transparent manner, without the need for registration.

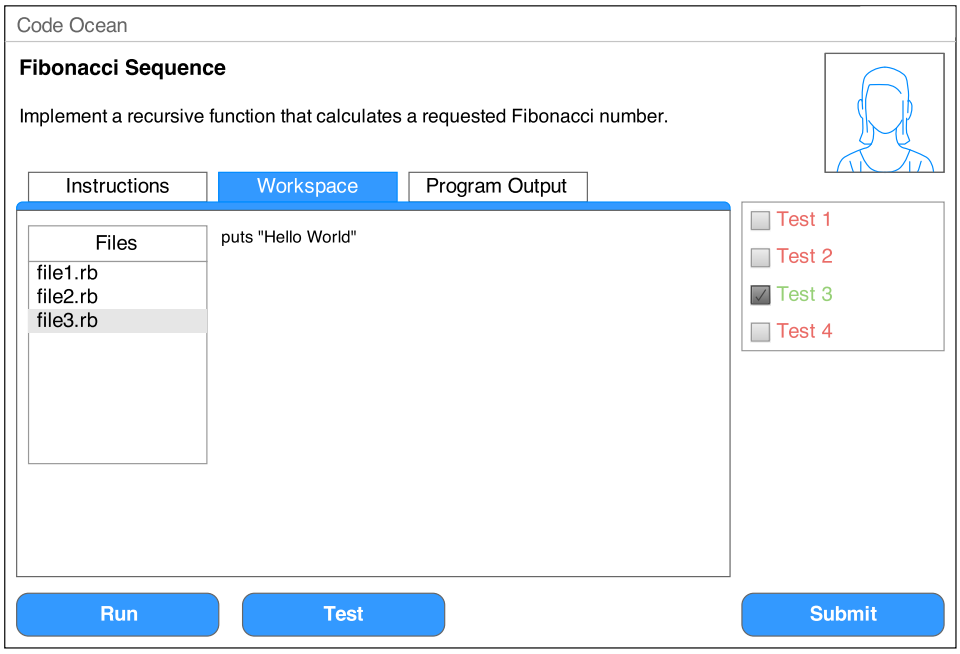

Figure 3.1: Mockup of the Development Environment

3.2 Development Environment

Section 2.6 introduced options for providing learners with programming tools and discussed their advantages and drawbacks. Subsequently, Section 2.7 examined exemplary development tools employed in MOOCs that teach programming to beginners. Based on the gained insights and the identified requirements, we decided in favor of a web-based development environment composed of a client-side code editor and a server-side component for code execution.

This approach entails a number of advantages. At first, it allows us to provide learners with a homogeneous and novice-friendly programming environment. Secondly, the approach supports a variety of programming languages and third-party libraries while providing a consistent workflow for both code execution and assessment. Thirdly, the approach enables insights into learners' problem-solving strategies by analyzing their code submissions.

All web-based programming tools used in MOOCs that we regarded are restricted to a single unit of editable code. In contrast, we decided that Code Ocean should promote the concept of files. We believe that it is important to support multiple editable files and the creation of new files since this enables more profound programming exercises, fosters learners' creativity and flexibility, and empowers learners to practice program design [68].

Furthermore, we want to provide learners with the ability to explore the behavior of the code they wrote by running it as frequently as desired. In addition, code should also be assessable as frequently as demanded.

Figure 3.1 depicts a mockup of a web-based development environment that incorporates our design decisions.

Code Ocean's development environment should be based on widespread web standards that are natively supported by current web browsers. Non-native technologies, such as Java applets [84] and third-party plugins, have to be avoided.

Currently out of scope of the web-based development environment are features for customization, debugging, and refactoring. However, we believe that the absence of these features will be negligible in the context of teaching programming to novices.

3.3 Code Execution

As stated in Section 3.1, the execution of student-written code demands security measures since learners may submit programs that excessively consume resources or even cause damage to the system.

A number of possible attacks against systems using automatic code evaluation have been described by Forišek [22]. Although it is far more likely that student programs are rather erroneous than deliberately malicious, providing our system in a MOOC context to a large number of learners makes it plausible that individual users may attempt to gain unauthorized access or do harm [68].

Therefore, Code Ocean has to provide a secured environment for running student programs that restricts the amount of consumable resources and withstands the damage that faulty or malicious programs may cause.

3.3.1 Container-based Virtualization

Isolation and resource control have traditionally been achieved through the use of VMs. However, abstraction provided by VMs comes at the cost of reduced performance. In order to meet the scalability requirements of MOOCs, OS–level virtualization techniques present an interesting alternative to traditional VMs since they impose almost no overhead. Rather than running a full OS on virtual hardware, OS–level virtualization approaches leverage built-in OS capabilities that enable isolated environments without starting a VM. Unlike a VM, such an environment can comprise as little as a single process and only owns the resources that it actively consumes.

Linux Containers (LXC) is one of several OS–level virtualization methods that provide multiple virtual environments on a single host system. LXC is based on control groups[1] and namespaces, which are OS features that allow limiting and isolating the resources used by groups of processes. Therefore, virtual environments, so-called containers, have their own process and network spaces and cannot see or access objects on the outside. Although LXC's underlying concepts are well known and mature, it has only recently been adopted and standardized in mainstream OSs.

Due to its benefits, container-based virtualization seems to be predestined for our use case. It allows to run students' programs in isolated environments that are practically unrestricted in terms of executable software while involving hardly any virtualization overhead. Containers represent a very flexible platform for diverse code evaluation tasks that is open to any programming language and third-party

[1] `https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt`

library available for Linux. The isolation provided by LXC dispenses us from measures such as integrating a stand-alone sandboxing solution [38] and detecting illegal code submissions by means of static analysis [79].

Since every single code submission is executed in a new container starting in a known state, start-up time is important. Low latencies for code execution are a precondition for providing results and feedback in a timely manner. Small feedback times grant learners a more interactive development process and facilitate iterative problem-solving strategies. In contrast to traditional VMs, LXC entails much less overhead for starting the virtualization platform, which is why the execution of a code submission usually involves an overhead of less than a second. Therefore, an interactive development workflow is enabled.

3.3.2 Docker

The open-source software Docker[2] provides an abstraction layer on top of LXC, including an image format and convenient tools for building, versioning, distributing, and deploying containers. Although several management tools for LXC exist, Docker has emerged as the de facto standard [20].

We decided to employ Docker as the execution platform for students' code submissions because it offers competitive performance [20] as well as user-friendly tools that allow instructors to define custom execution environments by themselves.

Docker involves the concept of images, which are stateless templates for containers that are used to prepare applications for execution in a Docker container. Since most applications rely on third-party utilities, libraries, or services, images enable users to package such applications along with their dependencies [57]. Existing Docker images can be used as starting points for the definition of new ones. Therefore, common dependencies can be bundled in a general base image to be used by multiple other ones. Besides open-source software, Docker offers Docker Hub[3], a web-based repository for Docker images. Users can push their images to the repository and fetch them on another machine. By providing images publicly, they can easily be shared.

In order to prepare an execution environment that is tailored to the needs of a particular course, teachers are expected to create a corresponding Docker image and publish it at Docker Hub. After that, Code Ocean can pull the image from the repository and utilize it for the execution of students' code submissions.

Docker provides two means for the creation of images. Firstly, an image can be created manually by making changes to a container and committing the results to a new image. This approach allows to evolve existing images in a simple way, but it does not promote automation and collaboration.

Secondly, Docker provides a tool for building images automatically from a list of instructions, as well as a DSL[4] for specifying the concrete steps to be taken. Using so-called Dockerfiles, images can be composed and adjusted in a textual fashion, which allows automated and reproducible creation. In addition to the capabilities provided by Docker Hub, Dockerfiles enable collaboration and version control using standard source code management tools, repositories, and practices.

[2] `https://www.docker.com/`
[3] `https://hub.docker.com/`
[4] `http://docs.docker.com/reference/builder/`

```
FROM ubuntu:14.04
MAINTAINER Hauke Klement <dev@hklement.de>
RUN apt-get update
VOLUME /workspace
WORKDIR /workspace
```

Listing 3.1: Code Ocean's Base Dockerfile

Listing 3.1 depicts the Dockerfile that describes the necessary steps for creating the common base image for all other Docker images used by Code Ocean. The initial `FROM` instruction indicates that the base image itself uses another Docker image as its foundation. In this example, *ubuntu:14.04*[5] (tag *14.04* from the *ubuntu* repository), a public Docker image providing a Ubuntu[6] basis is used. The following `MAINTAINER` instruction provides information about the image's creator. The `RUN` instruction allows to specify arbitrary commands to be executed during image creation. In our example, it is used for updating the system's package index. The `VOLUME` instruction creates a mount point to be used for mounting a directory containing student-written files for execution. Finally, the `WORKDIR` instruction sets the working directory for subsequent `RUN` instructions.

```
FROM hklement/ubuntu-base
MAINTAINER Hauke Klement <dev@hklement.de>
RUN apt-get install -y g++ make wget
RUN wget -O /tmp/python.tgz \
  https://www.python.org/ftp/python/3.4.2/Python-3.4.2.tgz
RUN mkdir /tmp/python
RUN tar xfz /tmp/python.tgz -C /tmp/python --strip-components=1
RUN cd /tmp/python && ./configure && make && make install
```

Listing 3.2: Dockerfile Describing a Python Environment

Listing 3.2 shows the Dockerfile corresponding to the image that Code Ocean uses for executing Python programming assignments. This time, the `FROM` instruction specifies to use the Docker image depicted in Listing 3.1 as a foundation. In order to install the most recent release of Python, the `RUN` instructions state to set up some dependencies, download the Python source to a temporary directory, extract the source archive, and start compilation.

The presented Dockerfiles illustrate the means that instructors are expected to use for defining custom execution environments for their courses. We believe that Docker provides teachers with a wide range of flexibility and a user-friendly tool chain.

3.4 Hint Generation

Learning programming is a trial-and-error process. As a part of this process, learners make mistakes, identify their misconceptions, and revise them. However, making

[5] https://registry.hub.docker.com/_/ubuntu/
[6] http://www.ubuntu.com/

mistakes can be discouraging if mistakes' origins remain unknown. In the event of a program error, standard interpreter output can be confusing for beginners and might not provide sufficient hints towards corrective actions. Common mistakes, such as confusing operators, using reserved keywords, or invoking methods with wrong arguments, might be easy to eliminate for advanced programmers, but they can be hard to identify for beginners [37]. In the worst case, learners might lose their ambition to continue.

According to Truong et al. [84], it is essential that students are given the opportunity to practice in an environment where they can receive constructive and corrective feedback. Therefore, we want to provide learners with hints that facilitate the understanding of program errors. Such hints could explain an error in a more accessible way, supply context-specific information, or suggest specific corrective actions.

Hints have to be defined by instructors. In order to provide instructors with a guideline towards which errors are common, Code Ocean logs and aggregates errors that occur during the execution of students' code. In order to associate context-specific error messages with certain error classes, errors are matched to hints using regular expressions, which allow ignoring insignificant textual differences, such as line numbers and identifier names. Hence, teachers have to provide a regular expression and a helpful message in order to define a hint.

In a large-scale e-learning environment, teachers cannot provide hints for every possible mistake, given the enormous number of code submissions and the large variety of errors. However, students who are solving the same assignments after having attended the same lectures make mistakes that tend to follow predictable patterns [78]. Therefore, we think that addressing the most common errors allows covering a good share of mistakes with manageable effort.

Since instructors author hints, they are able to control frequency and explicitness. Novice learners might be provided with precise instructions on how to face a problem, whereas more advanced students might only be supplied with subtle clues. Such inexplicit hints point at the origin of a problem but enable learners to discover the solution themselves [85]. Alternatively, teachers are also free to decide against providing hints at all.

3.5 Assessment

Section 2.5 underlined the relevance of assessment for the learning process and presented multiple approaches towards scalable assessment of programming assignments. Due to its focus on large-scale e-learning environments, such as MOOCs, scalable assessment is a crucial requirement of Code Ocean.

3.5.1 Approach

Since Code Ocean should provide an appropriate platform for teaching programming to everyone, including complete beginners, we decided to rely on automated assessment techniques rather than peer assessment. Automated assessment provides highly available and objective evaluation which makes it a predestined approach to supply learners with means for step-by-step refinement of their solutions.

Integration of additional manual assessment capabilities, however, is not within the scope of Code Ocean. Firstly, manual assessment is infeasible for large numbers

of learners and cannot provide general value. Secondly, a coexistence of automatic and manual assessment can easily create confusion among both teachers and learners if the origins of grades are not completely transparent [38].

We decided not to dictate a universally applicable assessment approach, such as I/O-based testing, but to grant instructors the freedom to select an assessment strategy that they consider appropriate for a specific use case. Besides being used for running students' submissions, Docker can provide a versatile platform for executing tests for assessment. In this way, instructors are very flexible in their choice of a particular assessment strategy. For instance, teachers are free to use an arbitrary testing framework, such as the one that is best practice for the language they teach, the one that fits the application domain best, or the one that they are most experienced with.

Besides relying on industry-strength testing tools, instructors can also choose to utilize tailor-made scripts for their assessment workflows. However, we encourage instructors to favor well-known solutions over improvised ones since established testing frameworks usually supply greater functionality and robustness, and can provide learners with relevant experience in using them.

Since the system's underlying assessment approach is visible to learners, they get in contact with the concept of software testing from the very beginning. Besides imparting an objective quality to the assessment, test-based evaluation provides learners with an understanding of the primary method for verifying industrial software [15]. Moreover, learners become accustomed to the idea of software testing as a means for controlling software quality and might be more willing to write their own tests later [86]. Instructors are free to provide the tests they use for assessment as visible part of the exercise skeleton, in this way permitting even deeper insights into the testing approaches used by professional developers.

3.5.2 Feedback

To take full advantage of the role of assessment as a feedback channel for learners, the results of teacher-provided test cases should convey learners a good understanding regarding the extent to which their code adheres to the exercise specification. Just as error messages provided by programming language interpreters can be too cryptic for novice programmers (see Section 3.4), the output generated by a testing framework can be confusing for beginners [47]. Inexperienced programmers might find it difficult to match the feedback supplied with a failing test to errors in their code [78].

In order to facilitate learners' troubleshooting, we want to supplement test frameworks' low-level output with instructor-provided feedback that is more easily understandable. Analogous to providing hint messages for common program mistakes, teachers are encouraged to provide understandable natural-language feedback for every test. Consequently, in the case of a failing test, the learner is supplied with a useful hint on how the program's behavior does not fulfill the specification and how it can be improved. Provided with a clear understanding of her program's inadequate aspects, the student might be more motivated to revise her solution.

3.5.3 Scoring

In order to map the results of a test run to a numerical grade, Code Ocean calculates a score that is based on the ratio of passed tests to failed tests. Since practical programming assignments are provided as a service to external e-learning applications, it remains in their responsibility to incorporate Code Ocean's scores into their grading practices. Scores might directly be mapped to a number of points or might be used for a threshold-based binary decision whether the learner has succeeded or not.

In order to control the influence of particular test cases on the final score, instructors should be enabled to assign different weights to tests. This way, a teacher can emphasize the role of a test covering a challenging program aspect by awarding more points for it.

3.6 Interoperability

Instead of designing Code Ocean as a proprietary component for a single e-learning platform, such as openHPI, or adding course management features to its scope, we decided to build a lean stand-alone application that is interoperable with existent e-learning systems. This approach enables the use of Code Ocean in conjunction with established LMSs and MOOC platforms, as well as in various scenarios, such as on-campus classes, small private online courses (SPOCs) [25], and MOOCs.

Providing tools such as Code Ocean as embeddable services matches the requirements of mainstream LMSs. Such systems are regarded to be relatively mature and stable applications that are maintained rather conservatively. They favor stability and reliability over flexibility and innovation [76]. Therefore, a service-oriented approach based on LMSs implementing essential core functionality for course and user management, and external tools providing special-purpose functionality to instructors and learners allows a trade-off between stability and innovation.

3.6.1 Learning Tools Interoperability

In order to provide interoperability with e-learning systems in a standardized fashion, we chose to support Learning Tools Interoperability (LTI) [76], an established interoperability standard in the area of e-learning applications. In this way, Code Ocean is interoperable with a wide range of applications that are compliant to the same standard. These applications include the popular open-source LMSs Canvas[7], Moodle[8], and Sakai[9], as well as the MOOC platforms Coursera, edX, and openHPI.

LTI is a specification developed by the IMS Global Learning Consortium[10]. It is aimed at establishing a standard for integrating remote content and third-party services into e-learning applications. In LTI lingo, these third-party services are called tools. Tools are hosted and supplied by so-called tool providers. E-learning applications that utilize tools are referred to as tool consumers.

LTI standardizes the protocols between tool providers and tool consumers, allowing external services to function like native parts of an LMS or a MOOC plat-

[7] `https://www.canvas.net/`
[8] `https://moodle.org/`
[9] `https://sakaiproject.org/`
[10] `http://www.imsglobal.org/`

form. To provide a seamless learning experience and to blur the boundaries between third-party tools and their host applications, tools are often embedded using inline frames. Alternatively, due to the limited space in inline frames, a stand-alone window might be more favorable for embedding more complex tools such as Code Ocean.

LTI 1.1.1 Code Ocean implements the LTI specification in version 1.1.1[11]. It covers the following mechanisms for interaction between tool providers and tool consumers:

- The provisioning and installation of external tools in e-learning applications.
- A tool launch protocol for sending a tool consumer's user to a tool provider while securely providing user identity, user role, and course context.
- Runtime web services that allow tool providers to create, retrieve, and delete results for users.

Code Ocean uses these capabilities in order to provide its services to trusted consumer applications, to receive user information regarding learners who start a programming session, and to send learners' results back to their consumer applications.

LTI 2.0 At the beginning of 2014, a new version of the LTI standard has been finalized. LTI version 2.0[12] aims at providing an industrial-strength standard and enabling more sophisticated and deeper integrations [1]. However, since the new standard has only recently been finalized and is not supported by many e-learning systems yet, we chose to rely on LTI 1.1.1 for the time being. Moreover, the integration features offered by this version are sufficient for Code Ocean's requirements. Nevertheless, as soon as version 2.0 becomes more widely adopted, it might be beneficial to support both versions of LTI.

[11] `http://www.imsglobal.org/lti/v1p1p1/ltiIMGv1p1p1.html`
[12] `http://www.imsglobal.org/lti/ltiv2p0/ltiIMGv2p0.html`

Chapter 4

Implementation

This chapter focuses on the implementation of our application. Section 4.1 introduces the building blocks that Code Ocean is constructed from. After that, Section 4.2 presents the application's domain model. Section 4.3 deals with the web-based development environment provided to learners. Section 4.4 illustrates the application's code execution approach, which is based on Docker. Section 4.5 describes the process of hint generation. Subsequently, Section 4.6 looks into the internals of Code Ocean's automated assessment method. Section 4.7 covers the application's interoperability workflow. In the end, Section 4.8 depicts the usage of Code Ocean from a teacher's perspective.

4.1 Building Blocks

As depicted in Figure 4.1, our solution is composed of a three-tiered web application based on Ruby on Rails[1], and a Docker server. The web application provides the development environment for learners as well as an administration back-end for teachers. Docker is used for code execution and assessment. Both core components can either be located on the same host or on different ones.

The web application communicates with the Docker server using its HTTP-based Remote API[2]. For this purpose, it utilizes an object-oriented interface to the API, which is provided by the docker-api[3] Ruby package, a so-called gem.

Client Side Code Ocean's client-side portion is built using the open-source front-end framework Bootstrap[4] and the open-source library jQuery[5], which facilitate the construction of a visually pleasing UI that is cross-browser compatible and compliant to modern web standards. The UI's responsive layout provides proper usability on various client platforms, including mobile devices.

Client-server communication is heavily based on Asynchronous JavaScript and XML (AJAX). Asynchronous background requests enable a single-page development workflow for the web-based development environment. Moreover, Rails' Tur-

[1] `http://rubyonrails.org/`
[2] `https://docs.docker.com/reference/api/docker_remote_api/`
[3] `https://github.com/swipely/docker-api`
[4] `http://getbootstrap.com/`
[5] `http://jquery.com/`

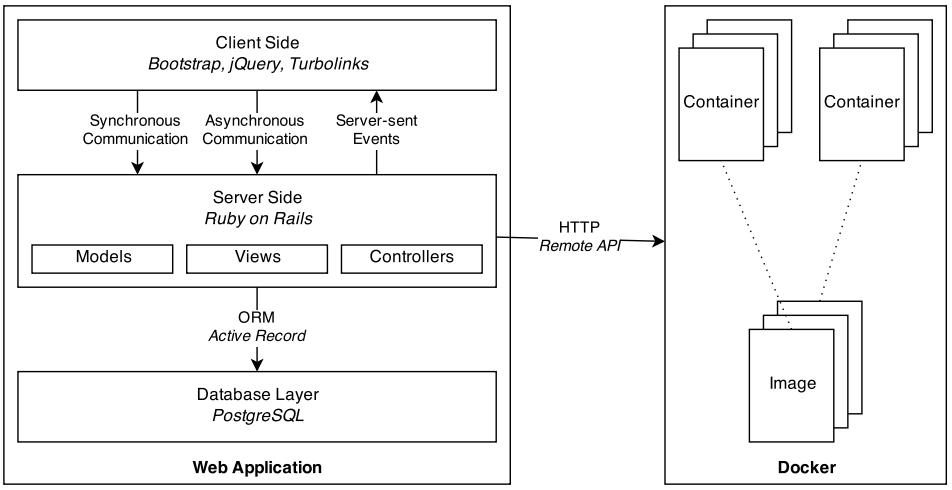

Figure 4.1: Building Blocks of Code Ocean

bolinks[6] feature improves page load times when navigating through the application by partially reloading visible content instead of performing full page loads.

Server Side Based on positive experience and due to its reputation as a highly productive web framework [24], we decided in favor of Rails. Rails produces reliable, concise code [27], not least due to the flexibility of Ruby, its underlying programming language. By favoring convention over configuration and promoting built-in automated testing, Ruby on Rails facilitates the development of powerful web applications.

As provided by Rails, Code Ocean's server-side code is organized according to the model-view-controller (MVC) architectural pattern [46].

Database In consequence of positive experience with PostgreSQL[7], Code Ocean has been developed and deployed using this database technology. However, thanks to Active Record[8], which is Rails' database-agnostic object-relational mapping (ORM) component and an implementation of the same-titled design pattern [23], the application's underlying database is easily exchangeable if desired.

4.2 Domain Model

This section presents the entities of the problem domain as well as the relationships among them. Figure 4.2 depicts the application's domain model in the form of a Unified Modeling Language (UML) class diagram. The diagram does not contain all attributes but is limited to the most relevant ones.

[6] `https://rubygems.org/gems/turbolinks`
[7] `http://www.postgresql.org/`
[8] `https://rubygems.org/gems/activerecord`

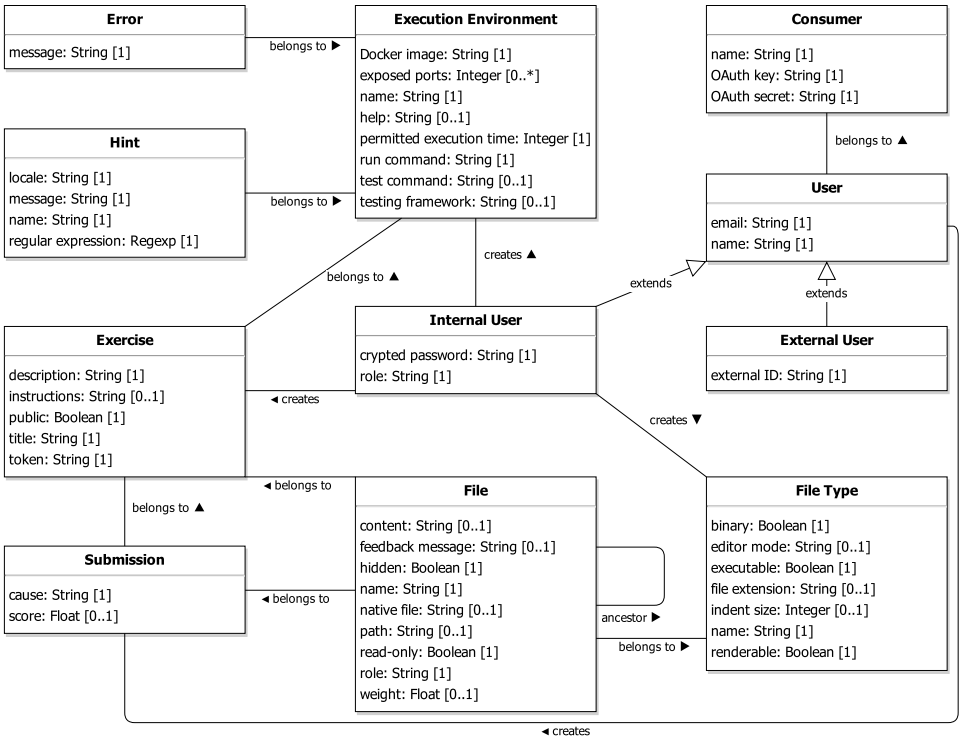

Figure 4.2: UML Class Diagram Presenting the Application's Domain Model

Consumer

As addressed in Section 3.6, Code Ocean provides its services to trusted e-learning applications, so-called consumers. Consumers have to be registered with Code Ocean and have to be provided with credentials. Every consumer is assigned a name, a unique OAuth key, and an OAuth secret.

User

We identified three types of users of Code Ocean:

- Administrators, who manage consumer applications and platform users.
- Teachers, who create content, examine learners' performance, and provide hints.
- Learners, who solve programming exercises.

These three user types fall into two different categories:

- Internal users, who are registered with Code Ocean and have privileges to perform content management.
- External users, who are mere visitors sent from a consumer application.

We decided to introduce two separate models for these conceptually different user classes. Moreover, since both user classes' sets of attributes are different and, more notably, their expected numbers vary significantly, we decided to store internal and external users in separate database tables instead of relying on single table inheritance [6], which is Rails' default strategy for mapping inheritance to relational databases.

Common to both user classes is that their instances belong to a consumer and have an email address and a name. While internal users provide this information in the registration process, external users' attributes are collected from the data provided during the launch of an LTI session.

Internal User Internal users have to sign in to Code Ocean using email address and password. Therefore, every internal user has an encrypted password. Additionally, each internal user has a role, which is either *administrator* or *teacher*.

External User In addition to the attributes shared with internal users, external users have an external identifier (ID) that is provided by the associated consumer.

Execution Environment

An execution environment describes a software platform used for the execution of code submissions. For instance, custom execution environments can be prepared for the requirements of individual courses, course modules, and even exercises.

Every execution environment is associated to the internal user who created it. An execution environment's central element is its Docker image, which provides an OS as well as third-party applications and libraries. Depending on the execution environment's needs, the permitted execution time for student-written code and a number of exposed ports can be specified. An execution environment's run command is necessary for executing code on behalf of a learner. In addition, its test command and testing framework are required for executing code for the purpose of assessment. In order to supply learners with a manual or with troubleshooting information, a help text can be provided.

Exercise

Every exercise is created by an internal user and belongs to an execution environment. The exercise's creator can decide whether it is public, and therefore visible to other internal users, or not. An exercise has a title, a short description, and explicit instructions, which should be as precise and unambiguous as possible. Every exercise has a uniquely generated token that is used for referencing the exercise when embedded by means of LTI. The core of an exercise is a collection of files, which may comprise skeleton code, tests, a reference solution, media, auxiliary scripts, and more.

Submission

A submission is a snapshot of a user's ongoing implementation of an exercise. For the main part, it consists of a number of files, which are either skeleton files that were modified or custom student-created files. Every submission has a cause, which refers to the reason why it was created.

Submissions are either created explicitly on behalf of a user or implicitly when saving code is a precondition for code execution. Submissions are created explicitly when a user saves her current progress or submits a finished implementation for grading. Submissions are created implicitly whenever custom files are added or deleted, before code is run for exploration, and before code is run for assessment.

Lastly, a submission can have a score, which is only true for submissions created for assessment.

File

As mentioned above, files exist either as part of an exercise or as part of a submission. A file that belongs to a submission and that is a modification of an instructor-provided file keeps a reference to this file, which is its ancestor.

Every file has a name and a file type. The content of a binary file is stored using a native file system file, whereas a non-binary file's content is stored in the database. A file's path refers to its path in the exercise workspace and can be used to realize an implicit workspace folder structure, for example to reflect the package structure in a Java project. A file's role is needed for detecting an exercise's main file, for collecting all test cases for assessment, and for setting up UI controls for file-related operations. During exercise creation, instructors can decide if a file is visible for learners, hidden from learners, or viewable in read-only mode.

Two further attributes exist that are only relevant for files containing test cases. A test file's weight equals the score that is awarded when all contained tests are passed. A test file's feedback message is displayed to the user when at least one test fails.

File Type

File types are created by instructors corresponding to the files they want to represent in their exercises.

Every file type has an editor mode and an indent size, which control associated files' visual properties when displayed in Code Ocean's editor. Binary attributes indicate whether files of a certain type are binary, executable, or renderable in a web browser. A file type's file extension is used for file system operations and for presenting files in the UI.

Hint

As introduced in Section 3.4, Code Ocean provides hints that facilitate beginners' understanding of programming mistakes.

Every hint belongs to a specific execution environment. A hint's regular expression is used for matching the errors that it refers to. A hint has a locale, which allows generating its message in the learner's preferred language.

Error

Errors that occur during the execution of learners' code are stored and aggregated in order to provide teachers with a guideline towards their students' common misconceptions.

Besides an association to an execution environment, errors comprise their original message.

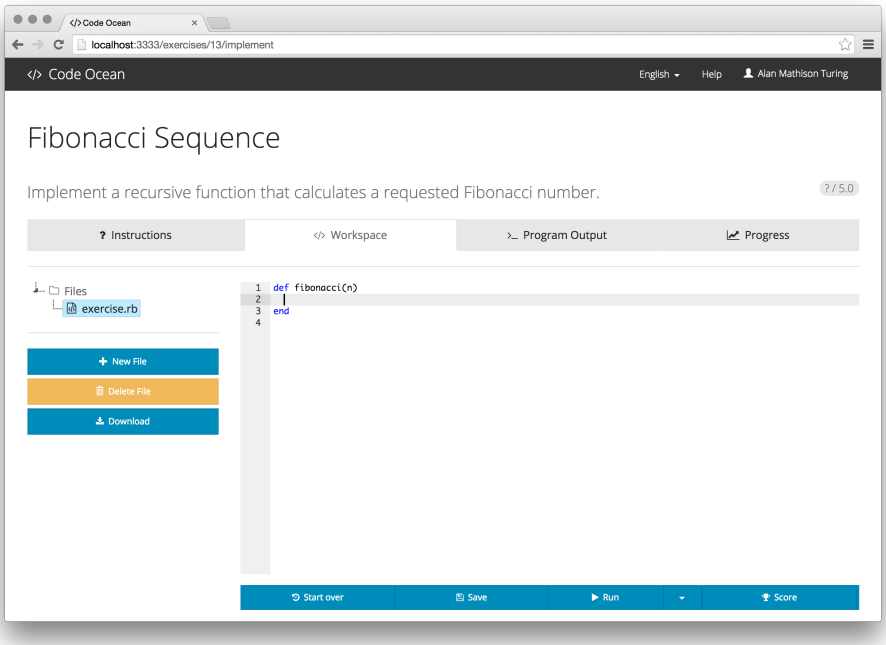

Figure 4.3: Development Environment: Workspace

4.3 Development Environment

This section presents the web-based development environment, which is the part of Code Ocean that learners are interacting with during their programming session. Figure 4.3 depicts the application from a learner's perspective.

4.3.1 Composition

The upper part of the view contains the active exercise's title and description, as well as the most recent score as determined by running the learner's solution against the exercise's tests. The navigation bar at the top of the window allows to control the UI's language and to access help.

Although a certain proficiency in English is usually required in the field of programming [80], we decided to facilitate the use of Code Ocean by performing internationalization. While the application is currently available in English and German, further translations can be added with little effort.

The help provided by Code Ocean comprises general directions on how to use the application as well as further information specific to the active exercise's execution environment. These resources are aimed to compensate for the missing face-to-face instructor involvement in the context of MOOCs [68].

The major part of the development environment is split up into four tabbed areas, each of which is associated to a step in learners' iterative development workflow depicted in Figure 4.4. As switching between these tabs is common, they can be accessed using keyboard shortcuts.

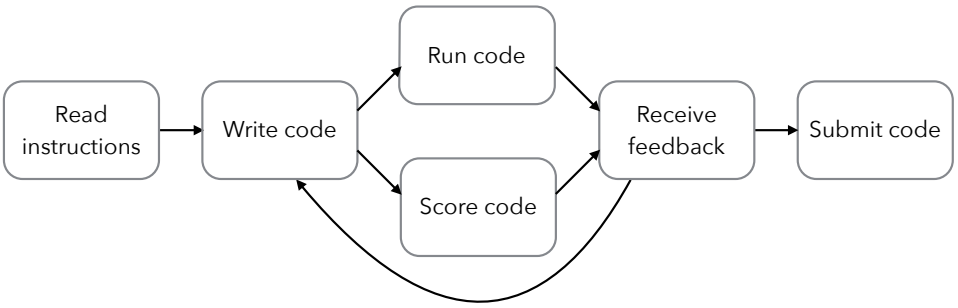

Figure 4.4: Iterative Development Process

The four areas are organized as follows:

- The first tab displays the exercise instructions. It is visible in the beginning of a programming session and can be revisited whenever the learner wants to review the specification. Exercise instructions may include the problem description, expected program behavior in edge cases, exemplary code snippets, and more. To enable a clear and visually appealing presentation, instructions are composed and rendered using Markdown[9].
- The second tab hosts the development environment's core component, which is the programming workspace. It is described in detail below.
- The third tab presents program output. Depending on the context, it contains output produced during the execution of learner-written code or during the execution of tests for the purpose of assessment. If the output refers to a program error for which a hint has been provided, the corresponding hint message is displayed along with the original output.
- The fourth tab is dedicated to the student's implementation progress. It displays an overview of the most recent test run, including the total score, partial scores, and feedback regarding the submission's deviations from the specification. A button allows to finish implementation and submit the code for final grading.

4.3.2 Workspace

The development environment's workspace is composed of a file tree, a code editing area, as well as some controls for development tasks.

File Tree

The visible files associated to an exercise are displayed in an interactive file tree, provided by jsTree[10]. In order to construct the file tree, a tree data structure is populated with the exercise's files and is transformed into the JavaScript Object Notation (JSON) representation expected by jsTree.

[9] `http://daringfireball.net/projects/markdown/`
[10] `http://www.jstree.com/`

```ruby
class FileTree < Tree::TreeNode
  def map_to_js_tree(node)
    {
      children: node.children.map { |child| map_to_js_tree(child) },
      icon: node_icon(node),
      id: node.content.try(:ancestor_id),
      state: {
        disabled: !node.is_leaf?,
        opened: !node.is_leaf?
      },
      text: node.name
    }
  end
  private :map_to_js_tree

  def to_js_tree
    {
      core: {
        data: map_to_js_tree(self)
      }
    }.to_json
  end
end
```

Listing 4.1: Excerpt from the `FileTree` Class

Listing 4.1 depicts an excerpt from the `FileTree` class, which is responsible for the second part of the process. In order to map a populated tree to the expected data format, `to_js_tree` is called, which recursively calls `map_to_js_tree` for every node in the tree, starting at the root node. Depending on the file represented by a tree node, `map_to_js_tree` returns an object describing the node's visual representation, including caption, icon, and child nodes.

Buttons below the file tree allow users to create, delete, and download files. The ability to create custom workspace files enables learners to practice program design and modularization by splitting up their code into functional units of their own choice.

Editors

Code Ocean's web-based development environment is based on Ace, an embeddable open-source code editor, written in JavaScript. We chose Ace from the set of available code editors due to its rich functionality, good reputation, and active maintenance. Ace is a popular and established piece of software, used within many web applications, such as Cloud9, Codecadamy, and GitHub, just to mention a few.

Ace offers source code editing capabilities that match the functionality and performance of native desktop editors. Its rich feature set includes syntax highlighting for a myriad of programming languages, UI theming, code folding, automatic indent, keyboard shortcuts, find-and-replace functionality, and more. Consequently, Ace offers a powerful foundation for Code Ocean's development environment.

Code Ocean utilizes multiple Ace editors for providing editing capabilities for all non-binary workspace files. At the beginning of a programming session, all editors are populated with their associated files' contents. Syntax highlighting, code indentation, and syntax validation are set up according to the files' types. Whenever a code snapshot has to be created, all editors' contents are obtained and sent to the server.

Binary workspace files are not displayed in editors. Instead, they are integrated in a way that depends on their type. Binary files that can be rendered by web browsers, such as supported audio files, image files, and video files, are directly embedded. Non-renderable binary files are linked.

Controls

A button bar underneath the editors provides controls associated to development tasks:

- The first button allows learners to start over from the beginning by rejecting their changes and resetting all files to the version that the teacher has provided.
- The second button serves to save the code. Since code is always implicitly saved before execution, explicit saving is principally useful for pausing a programming session.
- Depending on the selected workspace file, the third button serves the purpose of executing the file, stopping an active execution, or rendering the file on client side. The rendering option only applies to file types that are natively supported by web browsers, such as Graphics Interchange Format (GIF), HTML, JavaScript, and Scalable Vector Graphics (SVG).
- The fourth button triggers assessment of the current implementation.

A student can execute her code as often as desired, for instance to explore its behavior, to understand side effects, and to try out edge cases. This explorative mode of programming can be helpful to uncover flaws indicated by a failing test and to identify the necessary corrective actions [85].

Moreover, learners can also test their code as frequently as desired. Therefore, feedback received from test executions can be used to iteratively solve the exercise.

4.4 Code Execution

As introduced in Section 3.3, Code Ocean utilizes Docker to run students' code in an isolated environment. This section describes the precise workflow for running a code submission.

4.4.1 Workflow

The code execution workflow involves the web application's client side, its server side, and Docker. It is depicted in Figure 4.5.

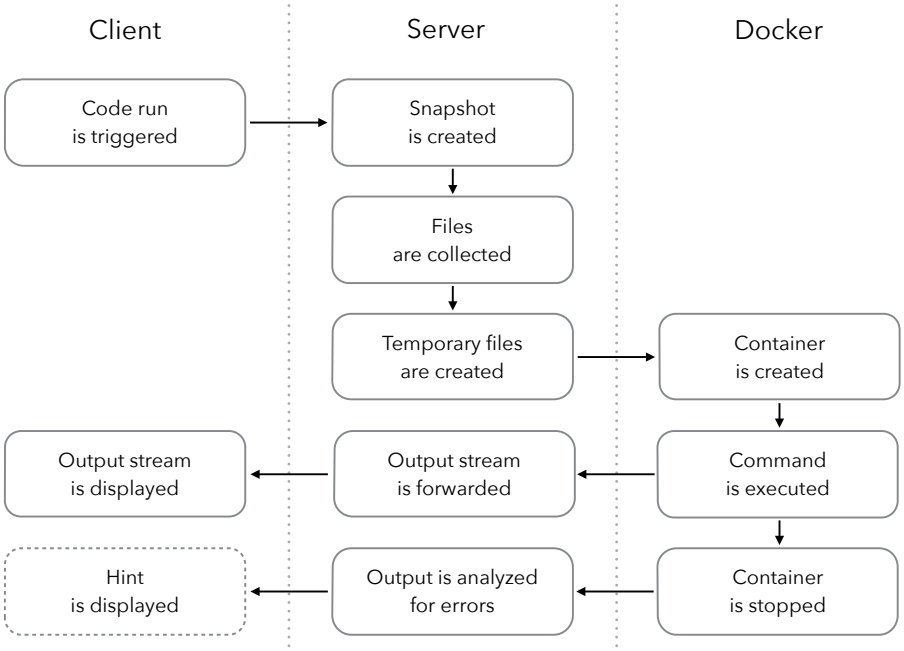

Figure 4.5: Code Execution Workflow

Whenever a learner triggers a code run from the development environment, her current implementation progress is sent to the server using AJAX. A snapshot is created and stored.

To execute a code submission, a new Docker container, based on the exercise's execution environment's Docker image, is provided. Every code execution starts with a blank slate, which prevents potential side effects of previous code executions.

Based on the execution environment's configuration, a number of network ports can be exposed by the Docker container during its runtime in order to allow a student to send and receive data. To avoid port collisions among simultaneously active learners, a pool of available ports is maintained, which provides mutually exclusive access to ports.

To supply the container with the necessary files, the exercise's skeleton files plus student-written files are collected from the database and are written to a submission-specific temporary directory. Depending on the physical location of the application server in relation to the Docker server, the files can be placed in a shared folder, explicitly transferred over a network, or made available through a network or Cloud file system. In the end, the submission-specific directory is mounted as a data volume into the container's file system.

Listing 4.2 depicts an excerpt from the `Submission` class, which illustrates how the files needed for execution are collected. At first, all skeleton files are collected and organized according to their IDs. Subsequently, student-written files are collected and organized according to their ancestors' IDs. Finally, both collections are merged, granting priority to student-written files, which can override their ancestors.

```ruby
class Submission < ActiveRecord::Base
  belongs_to :exercise
  has_many :files

  def collect_files
    ancestors = exercise.files.map(&:id).zip(exercise.files)
    descendants = files.map(&:file_id).zip(files)
    (ancestors + descendants).to_h.values
  end
end
```

Listing 4.2: Excerpt from the `Submission` Class Illustrating File Collection for Code Execution

```
docker run -v /tmp/submissions/42:/workspace \
  hklement/ubuntu-ruby bash -c 'ruby exercise.rb'
```

Listing 4.3: Exemplary Docker Invocation for Executing a Learner's Code Submission

To run a learner's submission, the execution environment's run command is executed in the prepared Docker container. Listing 4.3 shows an exemplary invocation in Docker's CLI syntax. It specifies a shared volume mapping, the Docker image to use, and which command to execute. In this example, a Ruby-specific invocation is passed to Bash[11]. The filename specified in the command is not hardcoded but dynamically inserted.

During execution, the Docker container's output is forwarded to the client side. After code execution has finished, the Docker container is discarded, allocated ports are released, and temporary files are deleted.

4.4.2 Output Streaming

The standard output streams of a Docker container are intercepted and supplied to the learner. Since rendering the entire output only after program execution is not sufficient for long-running processes, program output is streamed to the client in real time.

For that purpose, we decided to make use of server-sent events (SSE) [30], which is a technology that enables efficient server-to-client streaming of text-based event data using a long-lived traditional HTTP connection.

Since version 4, Rails offers built-in support for SSE. In this respect, Rails' `ActionController::Live`[12] module bundles functionality for real-time streaming of data to the client. In addition, the `ActionController::Live::SSE`[13] class provides the capabilities for writing events to a stream and automatically formatting the payload according to the SSE specification.

[11] http://www.gnu.org/software/bash/
[12] http://api.rubyonrails.org/classes/ActionController/Live.html
[13] http://api.rubyonrails.org/classes/ActionController/Live/SSE.html

The SSE standard's client-side API is contained in the `EventSource`[14] interface that enables clients to receive push notifications from servers as Document Object Model (DOM) events. For this purpose, the client's browser maintains a connection to the server, continuously polling for new data on the response stream. Whenever the stream's data changes, an event is fired.

SSE is available in all modern desktop browsers except Microsoft's Internet Explorer[15]. Since SSE uses a regular HTTP connection, client-side functionality can be replicated entirely in JavaScript unless natively supported.

4.4.3 Termination

More often than experienced programmers, novices write non-terminating code, which can block available server processes and waste resources [88]. In order to restrict the amount of wasted computing power, Code Ocean puts a limit on Docker containers' permitted execution time. If a running container's execution time exceeds the permitted duration, for example due to an infinite loop or never-ending recursion, the container is stopped and the learner is notified that she has built a non-terminating program.

Besides cutting down infinite loops, limiting execution time can also be used as a measure of quality assessment since an efficiency limit for code submissions can be enforced [68].

The permitted execution time of an execution environment can be tailored to the needs of a certain course. A few seconds of execution time should be enough for simple programs developed in a beginners' course, while wasted execution time is limited efficiently. A longer execution time should be granted to advanced courses involving intentionally long-running programs, complex Structured Query Language (SQL) queries, or heavy computations.

4.5 Hint Generation

Section 3.4 introduced Code Ocean's approach for providing easily accessible hints that facilitate the understanding of program errors and that can help to maintain students' motivation.

This section depicts how uncovered errors are logged, how hints are defined, and how matching hints are assigned to errors.

4.5.1 Error Logging

In order to allow teachers to tailor hints to the needs of their students, Code Ocean logs program errors produced by learners. Whenever an error is raised during the execution of student-written code that is not covered by a hint yet, its error message is stored in the database.

Errors occurring during server-side code execution are immediately collected from the executing Docker container's standard error stream and are written to the database.

[14] `http://www.w3.org/TR/eventsource/`

[15] `http://www.microsoft.com/download/internet-explorer.aspx`

In contrast, errors raised during client-side execution must be collected in a different way. In order to intercept client-side errors, a handler for the `onerror`[16] DOM event is assigned to the pop-up window used for client-side rendering of submissions, for instance corresponding to student-written HTML and other browser-renderable content. This handler serves as a global listener for any exception that is raised within the window. In this way, client-side errors can be intercepted, sent to the server, and stored in the database.

4.5.2 Hint Definition

Hints are defined by the teaching staff using Code Ocean's administration back-end. Hints are matched to error messages using regular expressions since this allows ignoring minor textual differences resulting from context-specific information. While providing a regular expression, a hint's author can use subexpressions for matching concrete identifiers contained in an error message.

In Ruby, a NoMethodError[17] is raised whenever a method is called on a receiver that cannot handle it. This type of error is often caused by typing mistakes or falsely memorized method names.

```
/undefined method `(.+?)' for (.+?):(.+?) \(NoMethodError\)/
```

Listing 4.4: Regular Expression for Matching NoMethodError Instances

Listing 4.4 shows a regular expression to match NoMethodError instances. It includes three subexpressions for matching the invoked method's name, the receiver's name, and the receiver's type.

```
Your object '$2' of class '$3' does not understand
the method '$1'. Maybe you made a typing mistake
or still have to implement that method.
```

Listing 4.5: Hint Message for NoMethodError Instances, Including Placeholders

Listing 4.5 displays a corresponding hint message, which contains three placeholders that match the regular expression's subexpressions. During hint generation, these placeholders are replaced by context-specific identifiers.

4.5.3 Hint Matching

Whenever an error occurs during the execution of student-written code, the error's message is matched against the regular expressions of all hints associated to the current execution environment. If a matching hint is found, its message is provided to the learner along with the original error message. Figure 4.6 depicts how hints are presented to learners.

Provided that a hint's regular expression contains subexpressions, the hint's message is adapted to the actual context that the error occurred in by replacing placeholders with concrete identifiers extracted from the error message. Listing 4.6

[16] https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers.onerror
[17] http://www.ruby-doc.org/core-2.1.5/NoMethodError.html

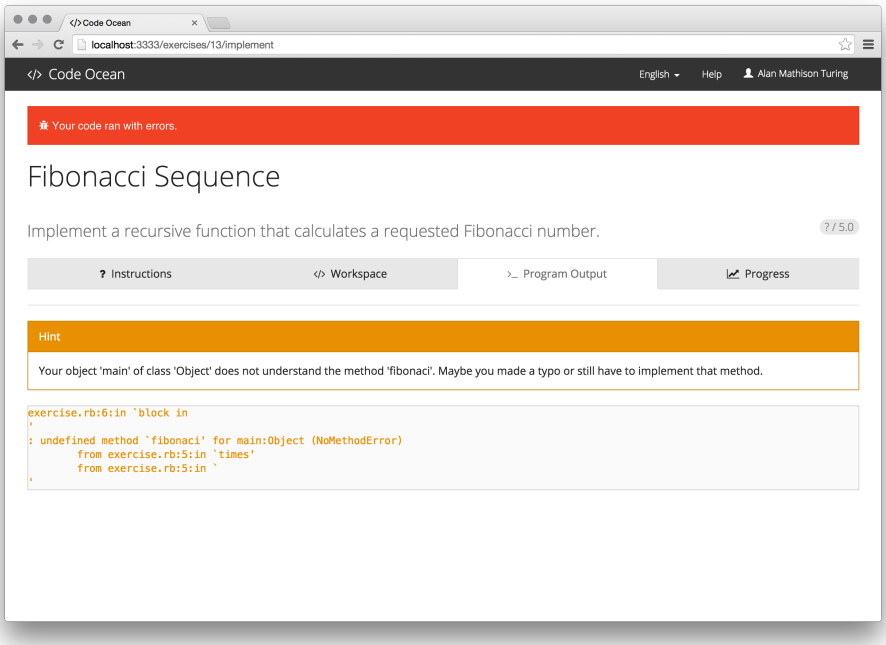

Figure 4.6: Development Environment: Hint Presentation

```
Your object 'main' of class 'Object' does not understand
the method 'fibonaci'. Maybe you made a typing mistake
or still have to implement that method.
```

Listing 4.6: Concrete Hint Message for a Specific `NoMethodError` Instance

displays the hint message from Listing 4.5 after being adapted to an exemplary error context.

Maintaining original error messages and providing them along with hints enables learners to form a mental connection between errors and hints. Therefore, when confronted with an error of the same kind again, the learner might recall what she has learned.

4.6 Assessment

Scalable assessment of assignments is a crucial requirement of tools to be employed in a large-scale e-learning context, such as a MOOC. Section 3.5 presented Code Ocean's scalable assessment strategy, which is based on automated execution of software tests and can provide learners with immediate and frequent feedback.

In order not to limit the capabilities and the value of assessment, we decided against imposing a homogeneous testing method. Instead, we allow instructors to use the testing framework that fits their requirements best.

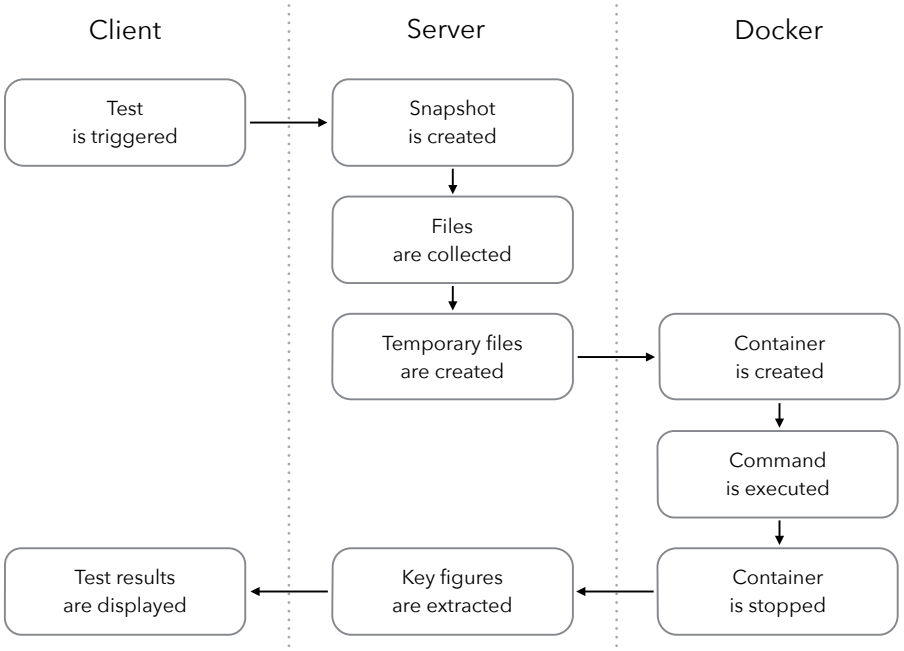

Figure 4.7: Assessment Workflow

4.6.1 Workflow

Code Ocean's assessment workflow is based on executing a learner's solution against a set of tests. Since these tests are invoked in the same manner as learners' main programs, the execution workflow, which is depicted in Figure 4.7, is very similar to that described in Section 4.4.

Initially, an up-to-date code snapshot is created. After that, the learner's work, exercise skeleton files, and, most important, the exercise's tests are written to a temporary directory, which is supplied to a Docker container.

```
docker run -v /tmp/submissions/42:/workspace \
  hklement/ubuntu-ruby bash -c 'rspec exercise_spec.rb \
    --format documentation'
```

Listing 4.7: Exemplary Docker Invocation for Assessing a Learner's Code Submission

Every test file is executed in a separate container based on the execution environment's Docker image. To do so, the execution environment's test command in invoked. Listing 4.7 shows an exemplary invocation in Docker's CLI syntax using the example of the popular Ruby testing framework RSpec.

When the test runs are finished, test results are extracted from the testing framework's output. The final assignment score is calculated based on the single test files' weighted percentages of passed tests. Finally, the assessment results are displayed to the student.

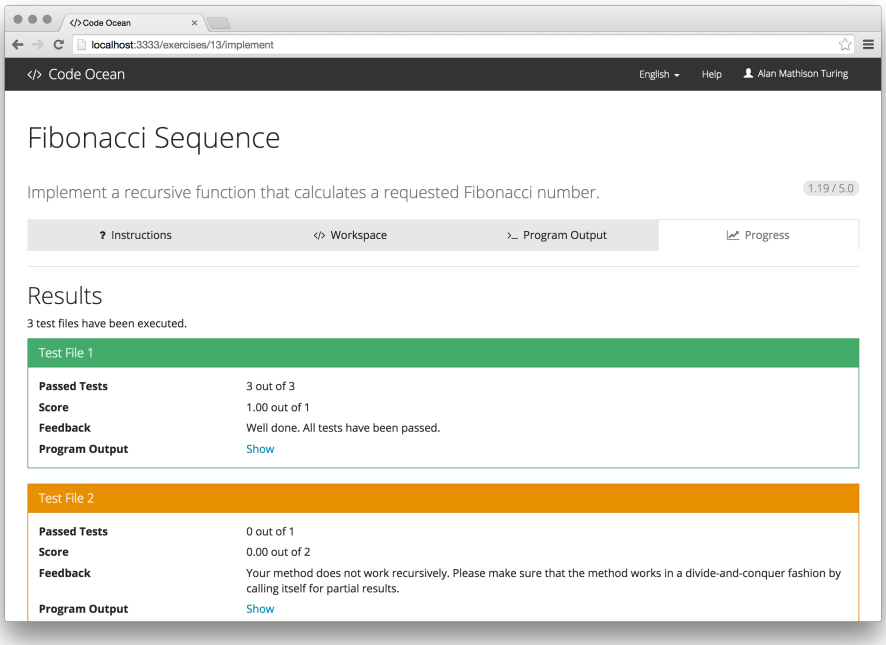

Figure 4.8: Development Environment: Assessment Presentation

Figure 4.8 depicts how assessment results are presented to the learner. The development environment's fourth tab contains the aggregated score of all test cases as well as detailed information per test file. For every file, the view displays the number of passed tests, the awarded score, and an instructor-provided feedback message if applicable.

The low-level testing framework output is presented in the adjacent tab. The output associated to a particular file can be accessed by following a link in the results overview. This allows students to form a connection between the teacher-provided feedback message and the low-level textual output.

4.6.2 Metadata

In order to control the weight of tests and to supply textual feedback, instructors need means to associate this kind of metadata with tests.

Since we decided to support a wide variety of testing frameworks, there is no universally applicable way of specifying metadata along with tests, for instance by using a DSL [68] or dedicated source code annotations [86].

Instead, instructors have to specify a test's metadata by means of attributes of the file containing the test (see Section 4.2). Therefore, in order to perform weighting and specify feedback in a fine-grained manner, a program's specification has to be expressed using multiple test files.

4.6.3 Testing Framework Adapters

In order to calculate a score and display it prominently in Code Ocean's UI, the application has to obtain key figures from the test run. Those are the number of executed test cases, the number of passed test cases, and the number of failed test cases.

Since we decided to provide teachers with the flexibility to use an arbitrary testing framework of their choice, there is no single source of data but a wide range of testing frameworks, each of which providing a proprietary API and using a proprietary output format. In order to obtain the required key figures despite this inhomogeneity, Code Ocean utilizes framework-specific adapters that extract the required information from testing frameworks' textual output. Such an adapter has to be provided for every testing framework or family of related frameworks to be used with Code Ocean.

A testing framework adapter is a lean Ruby class that extends the generic `TestingFrameworkAdapter` class and implements the `parse_output` method in a framework-specific manner. Based on the testing framework configured for an execution environment, a specific testing framework adapter is instantiated as part of the test execution workflow. In the workflow's last step, the testing framework adapter is used to extract the key figures named above from the testing framework's textual output. For this purpose, the output is passed to the adapter's `parse_output` method in order to be parsed. A testing framework adapter is expected to return a hash[18] including at least two of the three keys *count*, *failed*, and *passed*. Unless included, the third key can be inferred from the two given ones.

```ruby
class RspecAdapter < TestingFrameworkAdapter
  REGEXP = /(\d+) examples?, (\d+) failures?/

  def self.framework_name
    'RSpec 3'
  end

  def parse_output(output)
    captures = REGEXP.match(output[:stdout]).captures.map(&:to_i)
    {count: captures.first, failed: captures.second}
  end
end
```

Listing 4.8: Testing Framework Adapter for RSpec

```
...

Finished in 0.95839 seconds (files took 3.1 seconds to load)
3 examples, 0 failures
```

Listing 4.9: Output of an Exemplary RSpec Invocation, Using the Default Formatter

[18] `http://www.ruby-doc.org/core-2.1.5/Hash.html`

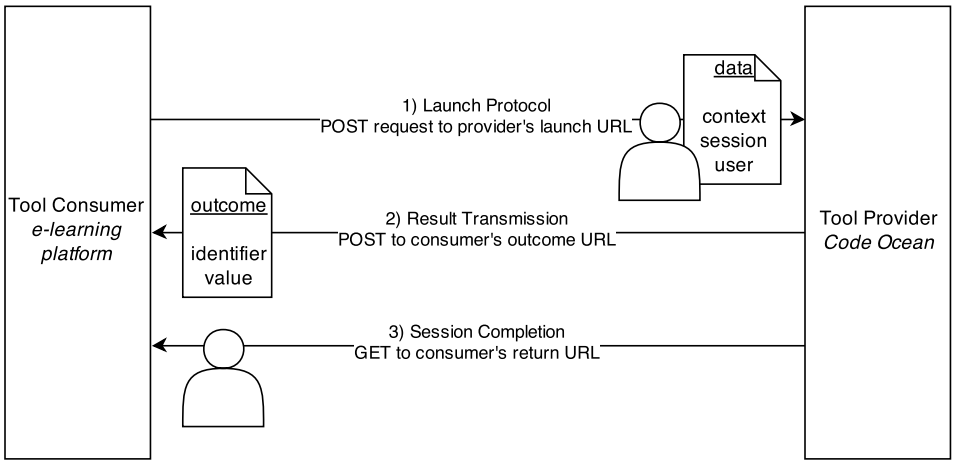

Figure 4.9: LTI Messages Exchanged in the Course of a Programming Session

Listing 4.8 depicts Code Ocean's testing framework adapter for RSpec. The single responsibility of the class is to parse RSpec-specific test output, as exemplarily depicted in Listing 4.9. According to the convention, the adapter extends the `TestingFrameworkAdapter` class and implements the `parse_output` method. It extracts the required key figures from the test output by means of a single regular expression that matches the number of executed tests and the number of failed tests.

As the example shows, testing framework adapters can be provided without much effort. Therefore, the set of testing frameworks supported by Code Ocean is easily extensible.

4.7 Interoperability

As introduced in Section 3.6, Code Ocean provides interoperability with existing e-learning applications based on the LTI standard. This section describes the concrete interoperability workflow.

As depicted in Figure 4.9, a single programming session involves three different messages between a tool consumer and Code Ocean. Initially, the tool consumer performs a launch request to start the programming session. Eventually, when the learner has finished her work, Code Ocean transmits the learner's result to the consumer. Finally, the learner returns to the consumer application.

4.7.1 Message Signing

LTI messages that are exchanged between a tool consumer and a tool provider are protected by means of OAuth 1.0[19] message signing. This allows both parties to validate the integrity of received messages. Since message signing requires the exchange of credentials, consumer applications have to be registered with Code Ocean and have to be provided with them. This way, only trusted consumer applications

[19] `http://oauth.net/core/1.0`

are able to profit from the services offered by Code Ocean. The OAuth credentials comprise a client key and a shared secret. The client key uniquely identifies a consumer and is transmitted in plain text with every request. The shared secret is used for generating message signatures.

When a tool provider receives a signed HTTP request, it looks up the initiating tool consumer by means of the request's client key. Using the shared secret associated to this consumer, the provider can calculate the message signature in order to verify the message's integrity.

4.7.2 Launch Protocol

The launch protocol serves the purpose of sending a user from an e-learning application to a third-party application, such as a special-purpose tool. The protocol is based on a regular HTTP POST request sent from a tool consumer to a tool provider, which is either issued in a server-to-server fashion or, more commonly, through submitting an HTML form on client side.

Workflow

In order to send a learner to Code Ocean, a consumer application must perform a POST request towards Code Ocean's LTI endpoint URL. Since LTI providers usually have a single endpoint, session-specific information is transmitted in the request's payload. To direct a learner to a specific exercise, Code Ocean requires the exercise token (see Section 4.2) to be provided as a so-called custom parameter. Furthermore, an additional *locale* parameter, which controls the UI's language, is supported.

If an incoming LTI launch request is validly signed, Code Ocean creates an external user based on the provided information, signs her in, and redirects her to the specified exercise. In case of a prior visit by the same learner, no user is created, but the existing one is retrieved from the database.

Launch Data

Besides request metadata and OAuth attributes, the payload of a launch request contains a set of parameters that provide information specific to the guest user and the context from which the tool provider is accessed.

The LTI specification defines a double-digit number of supported parameters for the launch protocol. However, very few of these parameters are mandatory. Therefore, information provided by tool consumers can be greatly limited, for instance in case of strict privacy settings. The LTI standard specifies a set of recommended parameters and encourages tool consumers to provide as much data as possible. However, it also advises tool providers to tolerate missing information.

Table 4.1 presents a set of typical parameters for an exemplary launch request sent to Code Ocean. Resource-specific parameters provide information regarding the link to access Code Ocean from within the consumer application. User-specific information includes the student's email address, ID, name, and roles. The provided ID typically does not correspond to the consumer application's real user ID, but it is usually a uniquely generated key. The *launch_presentation_return_url* parameter represents the URL to redirect learners to as soon as the programming session is completed. The *lis_outcome_service_url* and *lis_result_sourcedid* parameters indicate

Name	Type	Value
custom_locale	*C*	de
custom_token	*C*	ff79903c
launch_presentation_return_url	*R*	http://example.org/lti/return
lis_outcome_service_url	*O*	http://example.org/lti/outcome
lis_person_contact_email_primary	*R*	turing@example.org
lis_person_name_family	*R*	Turing
lis_person_name_full	*R*	Alan Mathison Turing
lis_person_name_given	*R*	Alan
lis_result_sourcedid	*O*	c9a68d73f61bb832
resource_link_id	*M*	42
resource_link_title	*R*	Fibonacci Sequence
roles	*R*	Learner
user_id	*R*	1912

Table 4.1: Exemplary LTI Launch Parameters

C: custom parameter, *M*: mandatory parameter, *O*: optional parameter, *R*: recommended parameter

that the tool consumer accepts results and specify how to provide them. Finally, as mentioned above, the launch data contains custom parameters that are used to reference a specific exercise and to declare the learner's preferred locale.

The LTI standard specifies several user roles, including *Administrator*, *Instructor*, *Learner*, and *Staff*. However, since Code Ocean uses LTI solely for providing learners access to the application, all external users are considered as such, regardless of the specified role.

4.7.3 Result Transmission

As covered in Section 4.6, when the student has finished her programming session and submits her code for terminal assessment, the final score is determined by executing the exercise's tests against the learner's solution.

If the tool consumer supports receiving results, the exercise score is normalized to a value between 0 and 1, as stipulated by the LTI standard, and sent to the outcome URL specified by the consumer. As discussed in Section 3.5, the responsibility to incorporate the practical assignment's score resides with the consumer application. In order to grant an appropriate score for finishing the assignment, the normalized score is usually multiplied by an assignment-dependent factor or used for a binary decision whether to award an adequate number of points.

Finally, the learner is signed out and redirected to a success page. This page contains her final score, a link to return to the consumer application, as well as some statistics regarding how well the learner performed in relation to her peers. Providing statistical information enables students to compare their scores with fellow students' ones and might be beneficial for long-term motivation [68].

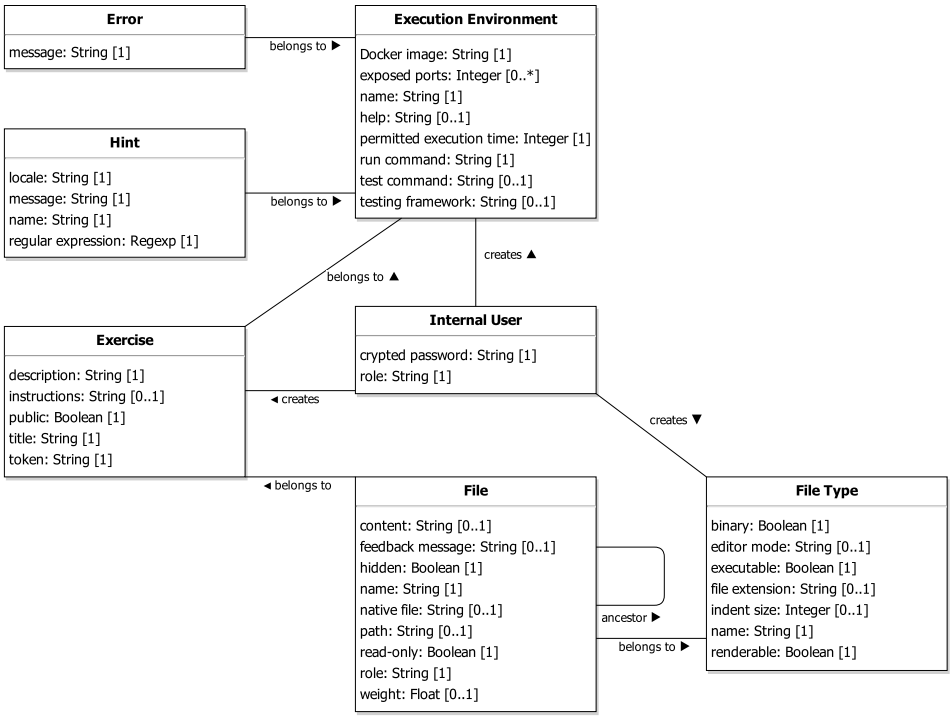

Figure 4.10: UML Class Diagram Presenting an Excerpt from the Application's Domain Model

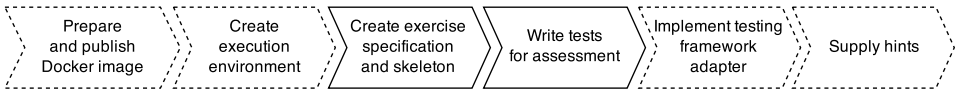

Figure 4.11: Steps of the Content Creation Process

4.8 Content Creation

This section describes the usage of Code Ocean from a teacher's perspective. It illustrates the steps that have to be taken in order to create programming assignments that fit the needs of a certain course or course module.

Figure 4.10 depicts an excerpt from Code Ocean's domain model, as presented in Section 4.2, which focuses on entities that are relevant for content creation.

In order to create content, teachers have to sign in to Code Ocean's administration back-end (see Figure 4.12) using their email address and password. Usually, their content creation process involves the steps presented in Figure 4.11. Steps depicted using solid lines are mandatory. Steps depicted using dashed lines are optional or apply only once per execution environment.

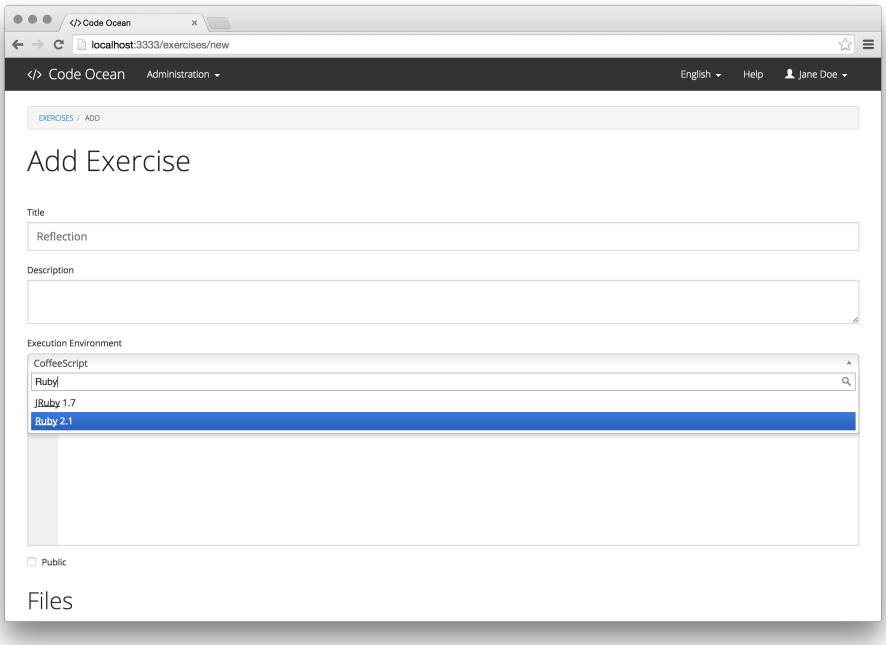

Figure 4.12: Administration Back-End

4.8.1 Execution Environment

The first step to create content for a new course is to create a qualified execution environment.

The core of an execution environment is its Docker image. In order to create a new execution environment, an instructor can either choose to reuse a Docker image that is already known to Code Ocean through an existing execution environment or can set up a new one. A new Docker image is specified using its identifier, such as *hklement/ubuntu-ruby:2.1.5*, which is composed of the repository name and an optional tag. Images are expected to be publicly available at Docker Hub, so that Code Ocean can download them. An instructor may pick an arbitrary Docker image that is published at Docker Hub or prepare a custom one by hand.

As introduced in Section 3.3, Dockerfiles provide a useful means for the reproducible creation of Docker images. Since Dockerfiles can build up on existing Docker images, the instructor may decide to reuse an existent image as a foundation. In order to foster the reusability of execution environments that Code Ocean already provides, their corresponding Docker images and Dockerfiles are publicly available. Docker images are hosted at Docker Hub[20], whereas Dockerfiles can be found at GitHub[21].

Listing 4.10 depicts a Dockerfile describing a JRuby[22] environment. Since JRuby requires a Java virtual machine (JVM), the Dockerfile is based on Code Ocean's Java Docker image that provides the required dependencies.

[20] `https://hub.docker.com/u/hklement/`
[21] `https://github.com/hklement/dockerfiles`
[22] `http://jruby.org/`

```
FROM hklement/ubuntu-java
MAINTAINER Hauke Klement <dev@hklement.de>
RUN wget -O /tmp/jruby.tar.gz \
  https://s3.amazonaws.com/jruby.org/.../jruby-bin-1.7.17.tar.gz
RUN mkdir /usr/jruby
RUN tar xfz /tmp/jruby.tar.gz -C /usr/jruby
ENV PATH $PATH:/usr/jruby/jruby-1.7.17/bin
RUN gem install bundler
```

Listing 4.10: Dockerfile Describing a JRuby Environment

Apart from a Docker image, the teacher has to specify a few other attributes when creating a new execution environment. These attributes include the permitted execution time for students' code submissions, a testing framework, as well as the commands for running the student's main program and the associated tests. An extensive list of execution environments' properties is presented in Section 4.2.

```
rspec %{filename} --format documentation
```

Listing 4.11: RSpec-specific Test Command

The commands for running student code and for executing tests usually expect a filename. For that reason, both commands can contain a placeholder that is replaced by an actual filename before invocation. For instance, the command depicted in Listing 4.11 is appropriate for assessing a student's submission using RSpec. To run a test, the command's *filename* placeholder is replaced by the actual filename of the test file.

In order to inspect her newly created execution environment, the creator can either utilize a Docker container on her local machine or use a simple shell provided by Code Ocean. This way, tests and sanity checks can be performed without being limited to the application's conventional development workflow.

4.8.2 Exercise

A new programming exercise can either be created from scratch or by cloning and adjusting an existing assignment. The second option allows to reduce the considerable effort for implementing automatically gradable exercises [68] by reusing parts of similar ones.

Consider the example of the Fibonacci sequence [45], a programming problem that is commonly used to teach the basics of recursion. The definition of the Fibonacci sequence is depicted in Figure 4.13. In order to create an exercise based on the Fibonacci sequence, an instructor has to specify title, description, and instructions, and has to create a number of files.

As emphasized in Section 2.5, exercise instructions should be provided in a detailed, unambiguous fashion. In this regard, Vihavainen et al. [85] recommend to make instructions as informative as possible and to include exemplary inputs with their expected outputs since this helps learners to confirm that they are proceeding into the right direction.

$$fibonacci(n) = \begin{cases} n & \text{if } n \leq 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{otherwise} \end{cases} \quad (4.1)$$

Figure 4.13: Definition of the Fibonacci Sequence

To enable a continuous workflow, all files associated to an exercise can be created within the exercise form. For an efficient workflow, instructors should prepare and test their exercises using their favorite tools on a local machine or in a Docker container. Afterwards, the completed exercise files can be uploaded. Besides a file's content, several other properties, including name, file type, and role, have to be defined. A complete list of properties can be found in Section 4.2.

```ruby
def fibonacci(n)
  # Return the correct Fibonacci number.
end
```

Listing 4.12: Exemplary Exercise Skeleton

Code Ocean does not dictate a certain exercise structure. However, a main file that constitutes the starting point for students' implementation should be provided. Based on her learners' level of knowledge, an instructor may choose to provide a program foundation ranging from a blank file to an extensive code skeleton, annotated with comments. A reasonable code skeleton for a Ruby-based Fibonacci sequence exercise is presented in Listing 4.12.

In order to create the basis for automated assessment, one or more test files containing test cases based on the exercise's execution environment's testing framework have to be defined. As discussed in Section 3.5, in order to increase the value of automated assessment for novice programmers, a helpful feedback message, which is presented to the learner when tests fail, has to be provided for every test file. Such a message might explain what went wrong and could provide hints on how to adjust the code.

By assigning different weights to test files, teachers can control the individual scores awarded for passing tests. Depending on their learners' capabilities, weighting might be used to increase or decrease the significance of challenging problem aspects for the final grade.

Tests should be defined in a manner that enables students to progress in small steps. In terms of the Fibonacci sequence example, the most basic test case might validate the existence of a `fibonacci` method that has an arity of one. Further test cases might validate that the method returns the expected values, behaves correctly in edge cases, and works recursively.

RSpec-based tests corresponding to two of these test cases are depicted in Listings 4.13 and 4.14. The first test uses the metaprogramming capabilities of the `method` method, defined by Ruby's `Kernel`[23] module, to check the existence and arity of the `fibonacci` method. Testing the method's recursiveness works by observing the number of times it is invoked during test execution.

[23] `http://www.ruby-doc.org/core-2.1.5/Kernel.html`

```
require './exercise'

describe '#fibonacci' do
  it 'is defined' do
    expect { method(:fibonacci) }.not_to raise_error
  end

  it 'has the correct arity' do
    expect(method(:fibonacci).arity).to eq(1)
  end
end
```

Listing 4.13: RSpec Test Validating the `fibonacci` Method's Existence and Arity

```
require './exercise'

describe '#fibonacci' do
  it 'works recursively' do
    n = 4
    expect(self).to receive(:fibonacci).and_call_original
      .at_least(n).times
    fibonacci(n)
  end
end
```

Listing 4.14: RSpec Test Validating the `fibonacci` Method's Recursiveness

Please note that tests like these are enabled by the ability to use a Ruby-specific testing framework. In contrast, a more universal yet more limited assessment approach, such as I/O-based assessment, would not permit to obtain the insights required for these tests.

4.8.3 Testing Framework Adapter

The instructor is flexible in her choice of a testing framework to use for the specification of automatically assessable assignments. However, if the chosen testing framework is not supported by Code Ocean yet, a testing framework adapter has to be provided that can extract key figures from the framework's textual output (see Section 4.6).

In order to keep the effort for preparing such an adapter at a minimum, Code Ocean supplies a generator for testing framework adapters, which integrates into Rails' built-in generator infrastructure.

```
rails g testing_framework_adapter Jasmine
```

Listing 4.15: Exemplary Generator Invocation for Generating a Testing Framework Adapter for the Jasmine Testing Framework

The said generator can be invoked as depicted in Listing 4.15 using the example of the testing framework Jasmine[24]. Besides a skeleton for a testing framework adapter, a second one for a corresponding unit test is generated. By completing both skeletons in a way that meaningful tests are defined and that these tests pass, the instructor can ensure to contribute a working piece of software.

After a new testing framework adapter has been prepared and integrated into the application, a newly supported testing framework appears in Code Ocean's administration interface.

4.8.4 Hints

As addressed in Section 3.4, teachers can supply their students with helpful hints regarding mistakes they make.

If an instructor has created a new execution environment, it has no associated hints yet. In order to change this, hints can either be provided in advance, based on the instructor's experience and expectations, or on demand, based on errors that occurred most during the execution of learners' code.

Just as the other resources, hints are created using the administration back-end. As discussed in Section 4.5, the teacher has to provide a regular expression and a helpful message for every hint.

[24] `http://jasmine.github.io/`

Chapter 5

Evaluation

This chapter presents an evaluation of Code Ocean in terms of scalability and versatility. Section 5.1 discusses the application's viability for the use in large-scale e-learning environments by analyzing its scalability, especially with regard to code execution and assessment. Section 5.2 reasons about the application's qualification to fit the needs of various programming courses by supporting an adequate amount of programming languages, application domains, and assessment techniques.

5.1 Scalability

In order to evaluate the scalability of our application, we prepared a production environment that is appropriate for handling the number of concurrent users to be expected in a MOOC. We simulated a corresponding load using Apache JMeter[1].

5.1.1 Test Environment

The production server is equipped with two Intel Xeon E5-2680 v2[2] central processing units (CPUs), supplying 40 logical CPU cores in total, and 64 GB of memory.

The server hosts all major components of Code Ocean, which are the web application, a web server, the database, and Docker. We use Ubuntu 14.04.1 LTS (64-bit), Docker 1.3.1, and PostgreSQL 9.3.5. As depicted in Figure 5.1, the web application is served by Puma[3] 2.10.1, a web server built for speed and concurrency, using Nginx[4] 1.6.2 as a reverse proxy.

In order to make best use of the parallelism provided by the server's many-core CPU, we decided not to use Ruby's standard interpreter, which is Matz's Ruby Interpreter (MRI), but to rely on JRuby, its JVM-based equivalent. While MRI's Global Interpreter Lock (GIL) limits the multi-thread performance of a single interpreter process by allowing only one thread to execute at a time [63], JRuby offers thread-level parallelism by mapping Ruby threads to Java threads, which in turn are mapped to native OS threads by most JVMs.

[1] `http://jmeter.apache.org/`
[2] `http://ark.intel.com/products/75277`
[3] `http://puma.io/`
[4] `http://nginx.org/`

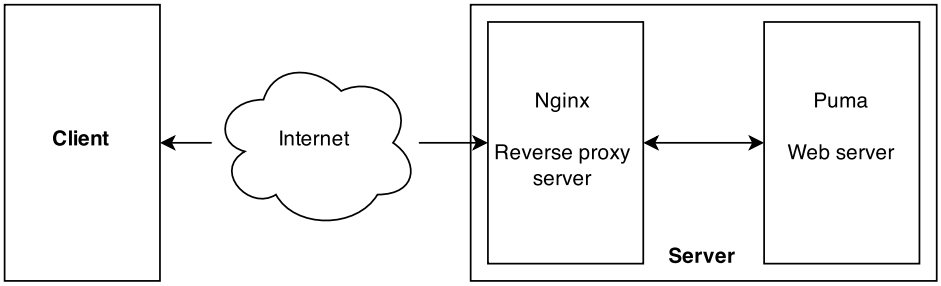

Figure 5.1: Web Server Setup

In order to prepare the production environment for the planned load, we set the maximum numbers of database connections allowed by PostgreSQL and allocatable by Active Record's connection pool[5] to 1024. Moreover, Puma has been configured to use up to 64 threads.

5.1.2 Test Plan

The JMeter-based load test simulates 500 learners who use Code Ocean in parallel to solve a practical assignment.

Each simulated student's programming session starts with an LTI launch request, as described in Section 4.7. After that, students perform an iterative development process, as discussed in Section 4.3. Each iteration comprises two requests between client and server. The first request sends the learner's code to the server for creating a code snapshot. The second request triggers either a code run or the execution of tests.

The requests associated to a single simulated student are issued at intervals of five seconds, which mimic the student's thinking time.

5.1.3 Test Results

Figure 5.2 depicts the response times of requests for starting the programming session and sending code revisions to the server. An LTI launch request, which involves validating the request signature, redirecting the learner to the specified exercise, and rendering the development environment, takes about 270 ms on average. The simpler request for creating a code submission, which does not render a view but yields a JSON response, takes about 50 ms on average.

Short and hardly varying response times throughout the entire load test indicate that Code Ocean is able to handle the simulated load for the regarded requests without problems. Moreover, the application should be able to handle a higher number of concurrent users when given a proportionate amount of resources.

Unfortunately, an entirely different picture emerges when taking code execution and assessment into account. Figure 5.3 depicts the response times of a subset of the requests regarded in Figure 5.2 as well as requests corresponding to code execution. As the graph in Figure 5.3 shows, response times for code execution increase with the number of concurrent requests. In a massively parallel usage

[5] `http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/ConnectionPool.html`

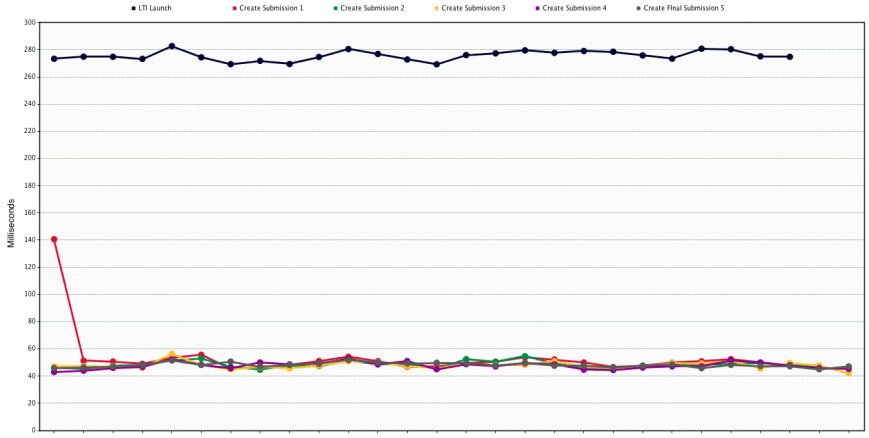

Figure 5.2: Response Time Graph Depicting Requests for LTI Launch and Snapshot Creation

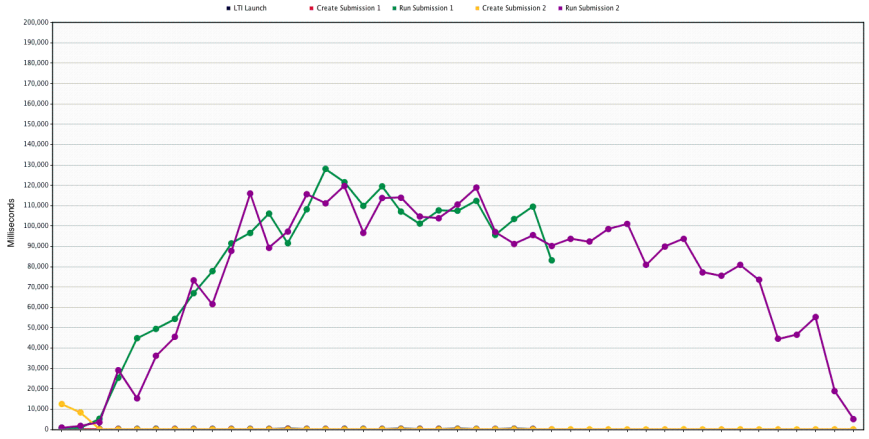

Figure 5.3: Response Time Graph Depicting Requests for LTI Launch, Snapshot Creation, and Code Execution

scenario, as simulated by the load test, response times rapidly reach a level at which students cannot be provided with feedback in a timely manner anymore. While concurrent requests increase the response times for code execution to tens of seconds, the response times of all other requests remain at similar levels as depicted in Figure 5.2. However, due to the increased range of response times, these requests are barely perceptible in Figure 5.3.

The fact that only Docker-related requests' response times escalate, while other requests are handled with ease, suggests that there is no general resource shortage but rather a problem concerning Docker. Figure 5.4 shows the server's workload during the load test as determined using the process viewer htop[6]. In fact, the image does not indicate resource shortage, but it shows that the server's available CPUs and memory are hardly used. Therefore, we believe that the poor scalability

[6] `http://hisham.hm/htop/`

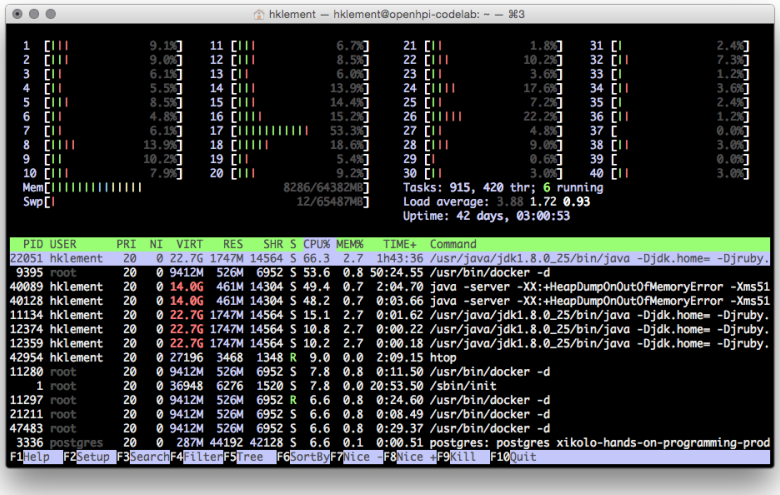

Figure 5.4: Server Workload During the Load Test

of code execution is attributable to problems with parallelizing server-to-server requests between the web application and Docker's API endpoint.

The production server's many-core CPU should easily support running a sizable number of Docker containers in parallel. In order to eliminate the possibility of a general parallelization problem, we conducted an experiment investigating the parallelizability of concurrent Docker executions. In contrast to short-running processes, which are usual for students' code submissions and which are represented in the load test, we regarded the behavior of long-running processes executed in Docker containers. In order to minimize the differences between the load test and the experiment, containers were started from within Code Ocean's web application. Figure 5.5 depicts the server's workload during the experiment. The image shows that all logical CPU cores are fully occupied by running the Docker containers. Hence, parallelization of concurrently running Docker containers is possible on the regarded infrastructure.

Based on our observations, we deduce that Docker can provide sandboxed execution of multiple learners' code submissions in parallel. While this should theoretically provide the scalability needed for a large-scale usage of Code Ocean, scalable code execution is not achieved in practice. We suspect that the increasing response times for concurrent code execution requests are caused by a problem with parallelizing the allocation of new Docker containers. In order to enable the usage of Code Ocean in a MOOC, this issue has yet to be eliminated or evaded.

5.2 Versatility

An important requirement of Code Ocean is to support a wide range of application domains, programming languages, and testing frameworks in order to fit the needs of various programming courses. This section presents a number of assignment domains that have been implemented prototypically. While Code Ocean's possibilities

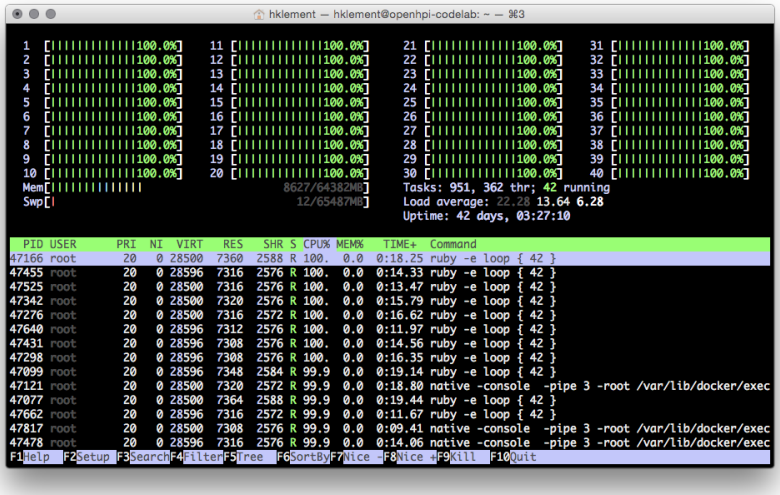

Figure 5.5: Server Workload During the Parallelizability Experiment

are far from being exhausted, we believe that the chosen examples underline the application's versatility.

Interpreted Languages A popular choice for teaching programming concepts to novices is using dynamic, high-level programming languages. Such languages typically feature a concise syntax that allows learners to express working programs within few lines of code and without dealing with low-level implementation issues.

In this regard, Code Ocean currently supports Ruby, Python, JavaScript, and CoffeeScript[7], the latter two based on Node.js[8]. Further execution environments for scripting languages can easily be defined since usually only an interpreter has to be installed. For each of the languages, assessment is based on a testing framework that is commonly used with the particular language. Namely, we use RSpec for Ruby, PyUnit[9] for Python, and Jasmine for JavaScript and CoffeeScript.

Compiled Languages Besides novice-friendly scripting languages, a handful of compiled languages play an important role in education. Java has been the dominant introductory teaching language over the past decade [32]. Also, C and C++ are commonly used as introductory languages [4, 87] due to their dominant role in OSs, embedded systems, and real-time rendering.

In contrast to interpreted languages, whose programs can usually be executed using a single command, compiled languages require multi-command workflows for execution. In the example of Java, the execution of an application comprises setting up the class path, compiling source files to bytecode class files, and finally launching the application by specifying the class whose main method to invoke. The workflow's complexity grows if third-party packages are involved.

As presented in Section 4.2, Code Ocean's design provides that execution environments use a single command for running student submissions as well as another

[7] `http://coffeescript.org/`
[8] `http://nodejs.org/`
[9] `http://pyunit.sourceforge.net/`

command for invoking the tests used for assessment. In order to make more complex execution workflows fit into this concept, we recommend using build utilities, such as Ant[10] and Make[11]. Depending on their teaching objectives, instructors can provide learners with prepared build recipes, such as Makefiles, or require students to write the build files themselves.

Web Development Web development skills are attractive to learners and demanded by the market [14], which is why web programming gains a stronger position in CS curricula [38]. To support this trend, Code Ocean offers means for building client-side to full-stack web applications.

Simple websites and web applications that only use client-side technologies such as HTML, JavaScript, and CSS do not require Docker-based execution. Instead, they are rendered in a separate window in the learner's web browser. By integrating powerful JavaScript frameworks, such as AngularJS[12], Ember.js[13], and D3.js[14], impressive applications can be realized solely using client-side capabilities. Assessment for client-side web applications can be provided by testing the JavaScript portion, using a web testing framework, or inspecting the HTML markup for validity and presence of expected elements.

Not only client-side web programming is supported by Code Ocean but also the development of full-stack web applications. An exemplary execution environment is based on Sinatra[15], a lightweight Ruby web application framework. To run such a web application, the execution environment's run command starts a process that serves the application from within the container. By exposing the port that the web server is running on, learners are able to connect to their application and try it out.

Databases Besides general-purpose programming courses, MOOCs dealing with database topics, such as Coursera's popular "Introduction to Databases"[16] and openHPI's "In-Memory Data Management"[17], can benefit from practical exercises. Therefore, we prepared two execution environments for the purpose of database assignments.

The first database-related execution environment is dedicated to the lightweight SQLite[18] database. Students work with a local database that grants unlimited privileges for data definition and data manipulation. Since a new Docker container is provided for every query execution, potentially undesirable effects of earlier queries cannot cause problems.

The second execution environment for database assignments is based on SAP's in-memory database HANA[19]. All students' queries are executed against a single shared database located on a remote server. Since destructive database operations would affect all learners, access rights to the database should be limited to

[10] `http://ant.apache.org/`
[11] `http://www.gnu.org/software/make/`
[12] `https://angularjs.org/`
[13] `http://emberjs.com/`
[14] `http://d3js.org/`
[15] `http://www.sinatrarb.com/`
[16] `https://www.coursera.org/course/db`
[17] `https://open.hpi.de/courses/imdb2014`
[18] `http://www.sqlite.org/`
[19] `http://www.saphana.com/`

user-specific schemas or restricted to reading data and calling read-only stored procedures.

There are two options for the execution of database queries. Students either write plain SQL statements, which are transparently executed in a manner prepared by the instructor, or they write programs that make use of high-level database abstractions, such as provided by Active Record and JDBC[20].

In line with these approaches, automated assessment can be realized by comparing a learner's query's result set against that of a reference query or by verifying correctness using an appropriate testing framework.

Context-specific hints can be provided for common beginners' mistakes related to SQL, such as misnamed fields and relations, incorrect clause order, and incomplete statements.

[20] `http://www.oracle.com/technetwork/java/javase/jdbc/index.html`

Chapter 6

Future Work

This chapter proposes future work. While Code Ocean essentially fulfills the requirements stated in Section 3.1, we have identified several areas for enhancement.

Scalability Although our application follows scalable approaches towards code execution and assessment, the scalability evaluation (see Section 5.1) revealed that Docker-based code execution does not scale within the scope of Code Ocean. In order to enable the usage of the application in a large-scale e-learning environment, such as a MOOC, this limitation has to be addressed.

Elasticity The application's current implementation is restricted to the usage of a single Docker host. Depending on the computational resources that are required to handle all learners' requests, this might be a stand-alone server or the same one that the web application is running on. As a consequence, in order to minimize the amount of idle resources, the server's dimensions have to be tailored to an average workload.

However, due to the fact that students tend to hand in their submissions just in time to meet deadlines, MOOCs have to face the technical challenge of significant load peaks shortly before those deadlines [35, 80, 85, 92]. Consequently, third-party tools to be used in MOOCs, such as Code Ocean, have to consider high-demand periods, too.

As suggested by Neuhaus et al. [62], a hybrid Cloud setup can provide an appropriate infrastructure for the varying workloads in a MOOC. Private Cloud resources are provided for handling an average workload. In addition, public Cloud resources can be utilized to handle workload peaks. In order to meet the elasticity demands of MOOCs, Code Ocean might incorporate this approach.

Usability Evaluation Within the scope of this work, we were not able to evaluate the usability of our application, which has been designed with the needs of novice programmers in mind. While we believe that Code Ocean provides a simple but sound programming environment, which supports beginners in their effort to learn programming and whose capabilities equal or exceed those of comparable programming tools, we see a usability evaluation involving beginning programmers as an important next step.

Assignment Variation Since automatic evaluation systems such as Code Ocean automate execution and assessment of programming assignments, automated variation of assignments is a natural extension. Assignment variation can provide each learner with a different version of a programming task for the purpose of discouraging plagiarism and encouraging natural cooperation in solving problems [51, 93]. Furthermore, variable assignments might be used to enable parameterized exercises that are slightly different after every submission attempt and therefore cannot be solved using mindless trial-and-error approaches [38].

Since Code Ocean does not impose a certain exercise structure but provides flexibility in terms of code execution and assessment, variable code skeletons, tests, and instructions might be realized with moderate effort.

Program Input At this moment in time, Code Ocean does neither provide means for supplying command-line arguments to a student-written program nor for writing to its standard input stream. However, user-defined program input might be required for certain programming assignments. Examples include applications that accept command-line options, applications that are controlled by keyboard, and applications that require CLI input.

In order to enable programming assignments like these, Code Ocean should be extended to support supplying programs with command-line arguments before execution and with user input during execution. For this purpose, the currently unidirectional output stream between a Docker container and the web-based development environment might be replaced by a bidirectional WebSocket [30] connection.

Collaborative Programming On-campus programming courses typically involve collaborative group projects. Besides teaching cooperation, collaborative programming practices can benefit the learning process. Students may understand concepts better when explained by a peer. At the same time, the explaining learner can foster her own understanding of the topic. Not only fellow students who are enrolled into the same course can cooperate, but also former course participants might be willing to volunteer as mentors [81].

Pair programming is an established software development practice that involves two programmers working together at a single workstation. Compared to developers working solo, pair programming is known to yield higher quality software in less time [95]. Likewise, student pair programmers are reported to work more autonomous and to perform better than their fellow students working alone [60].

Code Ocean does not support collaboration yet. However, we think that enabling collaborative programming is a meaningful objective. Remote cooperation on programming assignments could be provided by means of universal collaboration solutions, such as TogetherJS[1], Ace-specific collaboration solutions, such as Firepad[2], or real-time communication features, for instance based on WebRTC[3].

Auxiliary Resources In order to compensate for the missing face-to-face instructor involvement that is inherent in MOOCs, Code Ocean provides help resources that assist students in using the platform and in understanding the charac-

[1] `https://togetherjs.com/`
[2] `http://www.firepad.io/`
[3] `http://www.webrtc.org/`

teristics of specific execution environments. Moreover, assessment and hints provide learners with feedback regarding their problem-solving strategies.

The auxiliary resources provided by Code Ocean could be extended by integrating documentation for programming languages and frameworks into the development environment. Furthermore, e-learning applications usually offer groups or forums where learners can exchange their experiences. Such a dedicated group or forum might be made accessible from Code Ocean by providing its URL in the form of a custom LTI launch parameter.

Hybrid Assessment Currently, Code Ocean's scalable assessment approach is solely based on evaluating a submission's completeness and correctness according to the exercise specification. However, a combination of multiple assessment techniques is known to increase the quality of assessment [68]. Since Code Ocean facilitates the implementation of different assessment strategies, a combination of multiple approaches should be investigated.

As discussed in Section 2.5.1, exposing students to TDD at an early stage can teach them to appreciate its value and help them to become better programmers. Code Ocean already supports the execution of tests using various testing frameworks, both implicitly as part of assessment and explicitly on behalf of learners. Based on the present capabilities, strategies for involving testing into the grading process should be examined.

Section 2.5 presented and compared scalable assessment strategies. Automated assessment can provide instant feedback and enable an iterative development process. In contrast, peer assessment is typically applicable only after assignment completion, but it can provide feedback in greater depth. Moreover, it allows students to practice code reviews and gain insights into their peers' problem-solving strategies. In order to bring both assessment approaches' advantages together, it might be worthwhile to combine the current grading method with peer assessment.

Error Aggregation As discussed in Section 4.5, errors that occur during the execution of students' code are logged to the database in order to provide teachers with a guideline to identify the most common mistakes that their students make. Based on their insights, teachers can provide specific hints for these mistakes.

Currently, similar errors are aggregated based on identical error messages. This approach can provide some insights with simple methods, but it is limited to aggregating errors messages that contain the same context-specific identifiers, such as filenames, variable names, and types. In order to allow instructors to identify common misconceptions regardless of insignificant textual differences, more sophisticated aggregation techniques, for instance based on text similarity or keyword occurrence, should be implemented in the future.

Chapter 7

Conclusion

This work addressed the question how MOOCs can integrate practical programming assignments in a way that meets the demands of novice programmers and satisfies the scalability requirements of large-scale e-learning environments.

As a solution to this question, we presented Code Ocean, a web-based platform that provides hands-on programming exercises for MOOCs and that essentially fulfills the requirements identified in Section 3.1.

Versatility Code Ocean provides a versatile training platform that fits the needs of various programming courses. By using Docker for running students' code submissions, the application is open to a wide variety of programming languages and program domains. Tailor-made execution environments are easily configurable by instructors and enable the use of third-party applications, libraries, and custom tools. Since Code Ocean does not restrain exercise complexity, practical assignments can range from short beginners' programs to much more elaborate tasks. The ability to work with multiple files and to create new ones enables students to practice program composition and exceeds the usual functionality of web-based development environments used in MOOCs that we regarded.

Assessment of code submissions is based on automatic execution of tests but does not dictate a certain strategy. The approach provides instructors with the flexibility to use a testing framework of their choice. Therefore, teachers can employ the tools and techniques that they are most familiar with, that fit the assignment best, or that provide the deepest insights.

Novice-Friendliness We believe that Code Ocean offers a novice-friendly platform for practicing programming. Its web-based development environment provides learners with a homogeneous and installation-free training environment, is easy to use, and presents a low barrier for starting programming. While Code Ocean's UI is simple, the application offers decent development capabilities. On-demand code execution, easily understandable hints for errors, and natural-language feedback for failing tests facilitate students' problem-solving processes and enable an iterative development cycle.

Scalability While the chosen approaches for code execution and assessment theoretically offer the scalability for a large-scale usage of Code Ocean, our evaluation

revealed practical limitations in terms of code execution (see Section 5.1). Further efforts have to be made in order to provide the necessary scalability for using the application in a large-scale e-learning context, such as a MOOC.

Security Secure execution of arbitrary student-written programs is enabled by OS-level virtualization based on Docker. Besides offering lightweight virtualization, Docker provides abstractions and tools that facilitate the creation of custom execution environments and permit instructors to perform every aspect of the content creation process by themselves.

Interoperability Instead of extending a single MOOC platform for supporting practical programming assignments, we decided to develop a stand-alone web application whose services can benefit different e-learning systems, such as LMSs and MOOC platforms. By offering compliance with LTI, an established interoperability standard for e-learning applications, Code Ocean is usable as a special-purpose extension for a wide range of existing e-learning systems.

Bibliography

[1] Rob Abel. LTI 2: The future is here again, just a lot easier than once envisioned, 2012. [Online; accessed 14-December-2014]. URL: `http://www.imsglobal.org/blog/?p=184`.

[2] Timo Aho, Adnan Ashraf, Marc Englund, Joni Katajamäki, Johannes Koskinen, Janne Lautamäki, Antti Nieminen, Ivan Porres, and Ilkka Turunen. Designing IDE as a Service. *Communications of Cloud Software*, 1(1), 2011.

[3] Kirsti Ala-Mutka. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2):83–102, 2005.

[4] Kirsti Ala-Mutka, Toni Uimonen, Hannu-Matti Järvinen, and Linda Knight. Supporting Students in C++ Programming Courses with Automatic Program Style Assessment. *Journal of Information Technology Education*, 3, 2004.

[5] Sarah AlHumoud, Hend S. Al-Khalifa, Muna Al-Razgan, and Auhood Alfaries. Using App Inventor and LEGO mindstorm NXT in a Summer Camp to attract High School Girls to Computing Fields. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 173–177. IEEE, 2014.

[6] Scott W. Ambler. Mapping objects to relational databases: What you need to know and why, 2000. [Online; accessed 14-December-2014]. URL: `http://www.ibm.com/developerworks/library/ws-mapping-to-rdb/`.

[7] Kent Beck. Simple Smalltalk Testing: With Patterns. *The Smalltalk Report*, 4(2):16–18, 1994.

[8] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.

[9] Mordechai Moti Ben-Ari. MOOCs on Introductory Programming: A Travelogue. *ACM Inroads*, 4(2):58–61, 2013.

[10] Steve Benford, Edmund K. Burke, Eric Foxley, and Christopher A. Higgins. The Ceilidh System for the Automatic Grading of Students on Programming Courses. In *Proceedings of the 33rd Annual on Southeast Regional Conference*, ACM-SE 33, pages 176–182, New York, NY, USA, 1995. ACM.

[11] Douglas Blank. Robots Make Computer Science Personal. *Communications of the ACM*, 49(12):25–27, 2006.

[12] Maura Cerioli and Pierpaolo Cinelli. GRASP: Grading and Rating ASsistant Professor. In *Proceedings of the ACM-IFIP IEEIII 2008 Informatics Education Europe III Conference. Venice, Italy*. Citeseer, 2008.

[13] Amit Chauhan. Massive Open Online Courses (MOOCS): Emerging Trends in Assessment and Accreditation. *Digital Education Review*, 25:7–17, 2014.

[14] European Commission. Support Services to Foster Web Talent in Europe by Encouraging the Use of MOOCs Focused on Web Talent, 2014.

[15] Christopher Douce, David Livingstone, and James Orwell. Automatic Test-Based Assessment of Programming: A Review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.

[16] Stéphane Ducasse. SUnit Explained. Technical report, University of Berne, Institute of Computer Science, 2003.

[17] Stephen H. Edwards. Improving Student Performance by Evaluating How Well Students Test Their Own Programs, 2003.

[18] John English. Automated Assessment of GUI Programs using JEWL. *ACM SIGCSE Bulletin*, 36(3):137–141, 2004.

[19] Todd J. Feldman and Julie D. Zelenski. The Quest for Excellence in Designing CS1/CS2 Assignments. *ACM SIGCSE Bulletin*, 28(1):319–323, 1996.

[20] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. Technical report, IBM, Austin, TX, USA, 2014.

[21] Sally Fincher, Stephen Cooper, Michael Kölling, and John Maloney. Comparing Alice, Greenfoot & Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pages 192–193. ACM, 2010.

[22] Michal Forišek. Security of Programming Contest Systems. *Informatics in Secondary Schools, Evolution and Perspectives*, 2006.

[23] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[24] Armando Fox and David Patterson. Crossing the Software Education Chasm. *Communications of the ACM*, 55(5):44–49, 2012.

[25] Armando Fox, David Patterson, Richard Ilson, Samuel Joseph, Kristen Walcott-Justice, and Rose Williams. Software Engineering Curriculum Technology Transfer: Lessons learned from MOOCs and SPOCs. Technical report, EECS Department, University of California, Berkeley, 2014.

[26] Nicolas Fricke. Raising Completion Rates in xMOOCs through Social Engagement. Master's thesis, Hasso Plattner Institute, 2014.

[27] David Geer. Will Software Developers Ride Ruby on Rails to Success? *Computer*, 39(2):18–20, 2006.

[28] Daniela Giordano and Francesco Maiorana. Use of Cutting Edge Educational Tools for an Initial Programming Course. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 556–563. IEEE, 2014.

[29] Max Goldman, Greg Little, and Robert C. Miller. Real-Time Collaborative Coding in a Web IDE. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 155–164. ACM, 2011.

[30] Ilya Grigorik. *High Performance Browser Networking: What Every Web Developer Should Know About Networking and Browser Performance*. O'Reilly Media, Inc., 2013.

[31] Franka Grünewald, Christoph Meinel, Michael Totschnig, and Christian Willems. Designing MOOCs for the Support of Multiple Learning Styles. In *Scaling up Learning for Sustained Impact*, pages 371–382. Springer, 2013.

[32] Philip Guo. Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities, 2014. [Online; accessed 14-December-2014]. URL: `http://cacm.acm.org/blogs/blog-cacm/176450`.

[33] Maurice Halstead. *Elements of Software Science*. Elsevier Science Inc., 1977.

[34] Harri Hamalainen, Jukka Tarkkonen, Kari Heikkinen, Jouni Ikonen, and Jari Porras. Use of Peer-Review System for Enhancing Learning of Programming. In *Advanced Learning Technologies, 2009. ICALT 2009. Ninth IEEE International Conference on*, pages 658–660. IEEE, 2009.

[35] Colin Higgins, Tarek Hegazy, Pavlos Symeonidis, and Athanasios Tsintsifas. The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies*, 8(3):287–304, 2003.

[36] Jack Hollingsworth. Automatic Graders for Programming Classes. *Communications of the ACM*, 3(10):528–529, 1960.

[37] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. *ACM SIGCSE Bulletin*, 35(1):153–156, 2003.

[38] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93. ACM, 2010.

[39] David Jackson and Michelle Usher. Grading Student Programs using ASSYST. *ACM SIGCSE Bulletin*, 29(1):335–339, 1997.

[40] Katy Jordan. Initial Trends in Enrolment and Completion of Massive Open Online Courses. *The International Review of Research in Open and Distance Learning*, 15(1), 2014.

[41] Mike Joy, Nathan Griffiths, and Russell Boyatt. The BOSS Online Submission and Assessment System. *Journal on Educational Resources in Computing (JERIC)*, 5(3):2, 2005.

[42] Alan Kay. Squeak Etoys, Children & Learning, 2005. [Online; accessed 14-December-2014]. URL: `http://www.vpri.org/pdf/rn2005001_learning.pdf`.

[43] Jennifer S. Kay and Tom McKlin. The Challenges of Using a MOOC to Introduce "Absolute Beginners" to Programming on Specialized Hardware. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 211–212. ACM, 2014.

[44] Michael Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. The BlueJ System and its Pedagogy. *Computer Science Education*, 13(4):249–268, 2003.

[45] Thomas Koshy. *Fibonacci and Lucas Numbers with Applications*. John Wiley & Sons, 2011.

[46] Glenn E. Krasner and Stephen T. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[47] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A Study of the Difficulties of Novice Programmers. *ACM SIGCSE Bulletin*, 37(3):14–18, 2005.

[48] Winnie W. Y. Lau, Grace Ngai, Stephen C. F. Chan, and Joey C. Y. Cheung. Learning Programming through Fashion and Design: A Pilot Summer Course in Wearable Computing for Middle School Students. *ACM SIGCSE Bulletin*, 41(1):504–508, 2009.

[49] Tharindu Liyanagunawardena, Karsten Lundqvist, Luke Micallef, and Shirley Williams. Teaching Programming to Beginners in a Massive Open Online Course. In *Proceedings of OER14: building communities of open practice*, 2014.

[50] Marianne Lykke, Mayela Coto, Sonia Mora, Niels Vandel, and Christian Jantzen. Motivating Programming Students by Problem Based Learning and LEGO Robots. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 544–555. IEEE, 2014.

[51] Lauri Malmi, Ari Korhonen, and Riku Saikkonen. Experiences in Automatic Assessment on Mass Courses and Issues for Designing Virtual Courses. *ACM SIGCSE Bulletin*, 34(3):55–59, 2002.

[52] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. Programming by Choice: Urban Youth Learning Programming with Scratch. *ACM SIGCSE Bulletin*, 40(1):367–371, 2008.

[53] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, 2003.

[54] Thomas J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[55] Christoph Meinel, Michael Totschnig, and Christian Willems. openHPI: Evolution of a MOOC platform from LMS to SOA. In *Proceedings of the 5th International Conference on Computer Supported Education (CSEDU), INSTICC, Aachen, Germany*, volume 5, 2013.

[56] Christoph Meinel and Christian Willems. *openHPI: The MOOC Offer at Hasso Plattner Institute.* Number 80 in Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam. Universitätsverlag Potsdam, 2013.

[57] Dirk Merkel. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*, 2014(239), 2014.

[58] Christoph Müller. Security and Scalability of Programming Evaluation Systems in MOOCs. Master's thesis, Hasso Plattner Institute, 2014.

[59] Wolfgang Müller and Ulrich Kortenkamp. Learning Programming With Ruby. In *Proceedings of the IFIP Workshop "New Developments in ICT and Informatics Education". IFIP TC3*, 2010.

[60] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. Improving the CS1 Experience with Pair Programming. *ACM SIGCSE Bulletin*, 35(1):359–362, 2003.

[61] Felix Naumann, Maximilian Jenders, and Thorsten Papenbrock. Ein Datenbankkurs mit 6000 Teilnehmern. *Informatik-Spektrum*, 37(4):333–340, 2014.

[62] Christian Neuhaus, Frank Feinbube, and Andreas Polze. A Platform for Interactive Software Experiments in Massive Open Online Courses. *Journal of Integrated Design and Process Science*, 18(1):69–87, 2014.

[63] Rei Odaira, Jose G. Castanos, and Hisanobu Tomari. Eliminating Global Interpreter Locks in Ruby through Hardware Transactional Memory. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 131–142. ACM, 2014.

[64] Laura Pappano. The Year of the MOOC. *The New York Times*, 2012.

[65] Marian Petre and Blaine Price. Using Robotics to Motivate 'Back Door' Learning. *Education and Information Technologies*, 9(2):147–158, 2004.

[66] Dominic Petrick. Designing and Implementing a Peer Assessment Workflow for a MOOC Platform. Master's thesis, Hasso Plattner Institute, 2014.

[67] Chris Piech, Jonathan Huang, Zhenghao Chen, Chuong Do, Andrew Ng, and Daphne Koller. Tuned Models of Peer Assessment in MOOCs. In *Proceedings of the 6th International Conference on Educational Data Mining (EDM 2013)*, 2013.

[68] Vreda Pieterse. Automated Assessment of Programming Assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, pages 45–56, 2013.

[69] Hasso Plattner. *A Course in In-Memory Data Management.* Springer, 2013.

[70] Ken Reily, Pam Ludford Finnerty, and Loren Terveen. Two Peers are Better Than One: Aggregating Peer Reviews for Computing Assignments is Surprisingly Accurate. In *Proceedings of the ACM 2009 International Conference on Supporting Group Work*, pages 115–124. ACM, 2009.

[71] Jan Renz, Thomas Staubitz, Christian Willems, Hauke Klement, and Christoph Meinel. Handling Re-grading of Automatically Graded Assignments in MOOCs. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 408–415. IEEE, 2014.

[72] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2):137–172, 2003.

[73] Stephanie Rogers, Steven Tang, and John Canny. ACCE: Automatic Coding Composition Evaluator. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 191–192. ACM, 2014.

[74] Christian Safran. Collaborative Feedback: Code Peer Review in Higher Education. In *Proceeding of ICL2008, Villach, Austria*, 2008.

[75] Ken Schwaber. Scrum Development Process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

[76] Charles Severance, Ted Hanss, and Joseph Hardin. IMS Learning Tools Interoperability: Enabling a Mash-up Approach to Teaching and Learning Tools. *Technology, Instruction, Cognition and Learning*, 7(3-4):245–262, 2010.

[77] Nihar B. Shah, Joseph Bradley, Sivaraman Balakrishnan, Abhay Parekh, Kannan Ramchandran, and Martin J. Wainwright. Some Scaling Laws for MOOC Assessments, 2014.

[78] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated Feedback Generation for Introductory Programming Assignments. In *ACM SIGPLAN Notices*, volume 48, pages 15–26. ACM, 2013.

[79] Thomas Staubitz. Mini Exercises for Java, Based on JavaBat, 2009.

[80] Thomas Staubitz, Jan Renz, Christian Willems, Johannes Jasper, and Christoph Meinel. Lightweight Ad Hoc Assessment of Practical Programming Skills at Scale. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 475–483. IEEE, 2014.

[81] Thomas Staubitz, Jan Renz, Christian Willems, and Christoph Meinel. Supporting Social Interaction and Collaboration on an xMOOC Platform. In *EDULEARN14 Proceedings*, pages 6667–6677, 2014.

[82] Matthew Thornton, Stephen H. Edwards, Roy P. Tan, and Manuel A. Pérez-Quiñones. Supporting Student-Written Tests of GUI Programs. *ACM SIGCSE Bulletin*, 40(1):537–541, 2008.

[83] Nikolai Tillmann, Jonathan de Halleux, Tao Xie, and Judith Bishop. Code Hunt: Gamifying Teaching and Learning of Computer Science at Scale. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 221–222. ACM, 2014.

[84] Nghi Truong, Peter Bancroft, and Paul Roe. Learning to Program Through the Web. *ACM SIGCSE Bulletin*, 37(3):9–13, 2005.

[85] Arto Vihavainen, Matti Luukkainen, and Jaakko Kurhila. Multi-faceted Support for MOOC in Programming. In *Proceedings of the 13th Annual Conference on Information Technology Education*, pages 171–176. ACM, 2012.

[86] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. Scaffolding Students' Learning using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, pages 117–122. ACM, 2013.

[87] Birgit Vogel-Heuser, Sebastian Rehberger, Timo Frank, and Thomas Aicher. Quality Despite Quantity - Teaching Large Heterogenous Classes in C Programming and Fundamentals in Computer Science. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 367–372. IEEE, 2014.

[88] Martin von Löwis, Thomas Staubitz, Ralf Teusner, Jan Renz, Susanne Tannert, and Christoph Meinel. Scaling Youth Development Training in IT Using an xMOOC Platform. Unpublished, 2014.

[89] Milena Vujošević-Janičić, Mladen Nikolić, Dušan Tošić, and Viktor Kuncak. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. *Information and Software Technology*, 55(6):1004–1016, 2013.

[90] Qianxiang Wang, Wenxin Li, and Tao Xie. Educational Programming Systems for Learning at Scale. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 177–178. ACM, 2014.

[91] Joe Warren, Scott Rixner, John Greiner, and Stephen Wong. Facilitating Human Interaction in an Online Programming Course. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 665–670. ACM, 2014.

[92] Christian Willems, Nicolas Fricke, Sebastian Meier, Richard Meißner, Kai-Adrian Rollmann, Simon Völcker, Sebastian Woinar, and Christoph Meinel. Motivating the Masses - Gamified Massive Open Online Courses on openHPI. In *EDULEARN14 Proceedings*, pages 4042–4052, 2014.

[93] Christian Willems, Johannes Jasper, and Christoph Meinel. Introducing Hands-On Experience to a Massive Open Online Course on openHPI. In *IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE2013)*, pages 307–313. IEEE, 2013.

[94] Christian Willems, Jan Renz, Thomas Staubitz, and Christoph Meinel. Reflections on Enrollment Numbers and Success Rates at the openHPI MOOC Platform. In *Proceedings of the European MOOC Stakeholder Summit 2014*, pages 101–106, 2014.

[95] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the Case for Pair-Programming. *IEEE Software*, 17(4):19–25, 2000.

[96] Brenda Cantwell Wilson and Sharon Shrock. Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. *ACM SIGCSE Bulletin*, 33(1):184–188, 2001.

[97] Ling Wu, Guangtai Liang, Shi Kui, and Qianxiang Wang. CEclipse: An Online IDE for Programing in the Cloud. In *IEEE World Congress on Services (SERVICES) 2011*, pages 45–52. IEEE, 2011.

[98] David Xiao and Robert C. Miller. A Multiplayer Online Game for Teaching Software Engineering Practices. In *Proceedings of the First ACM Conference on Learning @ Scale*, pages 159–160. ACM, 2014.

[99] Joseph L. Zachary and Peter A. Jensen. Exploiting Value-Added Content in an Online Course: Introducing Programming Concepts via HTML and JavaScript. *ACM SIGCSE Bulletin*, 35(1):396–400, 2003.

List of Acronyms

AJAX	Asynchronous JavaScript and XML
API	application programming interface
BDD	behavior-driven development
CLI	command-line interface
CPU	central processing unit
CS	computer science
CSS	Cascading Style Sheets
DOM	Document Object Model
DSL	domain-specific language
GIF	Graphics Interchange Format
GIL	Global Interpreter Lock
GUI	graphical user interface
HPI	Hasso-Plattner-Institut
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I/O	input/output
ID	identifier
IDE	integrated development environment
IT	information technology
JSON	JavaScript Object Notation
JVM	Java virtual machine
LMS	learning management system
LTI	Learning Tools Interoperability
LXC	Linux Containers
MOOC	massive open online course
MRI	Matz's Ruby Interpreter
MVC	model-view-controller
ORM	object-relational mapping
OS	operating system
PaaS	Platform as a Service
PC	personal computer
SaaS	Software as a Service
SDK	software development kit

SPOC	small private online course
SQL	Structured Query Language
SSE	server-sent events
SVG	Scalable Vector Graphics
TDD	test-driven development
UI	user interface
UML	Unified Modeling Language
URL	uniform resource locator
UX	user experience
VM	virtual machine
WWW	World Wide Web
XP	Extreme Programming

List of Figures

List of Listings

List of Tables

Declaration of Originality

I hereby declare that I have prepared this master's thesis myself and without inadmissible assistance. I certify that all citations and contributions by other authors used in this thesis or that led to the ideas behind this thesis have been properly identified and referenced in written form. I also warrant that the same applies to the implementation of the project.

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne unzulässige Hilfe Anderer angefertigt habe. Ich bestätige, dass alle Zitate und Gedankengänge anderer Autoren, welche in dieser Arbeit verwendet wurden oder zu Ideen in ihr führten, ordnungsgemäß gekennzeichnet und mit schriftlichen Verweisen versehen wurden. Ich erkläre weiterhin die Gültigkeit dieser Aussage für die Implementierung des Projekts.

Potsdam, December 19$^{\text{th}}$, 2014

Hauke Klement