CodeOcean and CodeHarbor: Auto-Grader and Code Repository

Sebastian Serth
Hasso Plattner Institute
Potsdam, Germany
sebastian.serth@hpi.de

Thomas Staubitz
Hasso Plattner Institute
Potsdam, Germany
thomas.staubitz@hpi.de

Ralf Teusner
Hasso Plattner Institute
Potsdam, Germany
ralf.teusner@hpi.de

Christoph Meinel
Hasso Plattner Institute
Potsdam, Germany
christoph.meinel@hpi.de

ABSTRACT

The Hasso Plattner Institute (HPI) successfully operates a MOOC (Massive Open Online Course) platform since 2012. Since 2013, global enterprises, international organizations, governments, and research projects funded by the German ministry of education are partnering with us to operate their own instances of the platform. The focus of our platform instance is on IT topics, which includes programming courses in different programming languages. An important element of these courses are graded hands-on programming assignments. MOOCs, even more than traditional classroom situations, depend on automated solutions to assess programming exercises. Manual evaluation is not an option due to the massive amount of users that participate in these courses. The paper at hand presents two of the tools developed in this context at the HPI: CodeOcean—an auto-grader for a variety of programming languages, and CodeHarbor, a tool to share auto-gradable programming exercises between various online platforms.

Author Keywords

MOOC; Scalability; Programming; Auto-Grader; Code Repository; Sharing

CCS Concepts

•**Applied computing** → **Collaborative learning;** *Learning management systems; Distance learning; E-learning;* •**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

INTRODUCTION

Future generations should learn the skills to participate in digital life as part of modern society. These skills are often coined as fundamental digital knowledge, and most people agree that this includes programming skills. For that reason, educational measures to teach programming "to the masses" are both of huge interest as well as vastly required, given the fact that neither enough teachers with the necessary skill set exist nor will everybody be able to go back to school. But even when having a shared understanding of the necessity of programming education, the follow-up questions, how far that education should go, and what content it should include, brings up vivid discussions again. From our point of view, profound knowledge of syntax and control structures of at least one major fourth-generation programming language is just the basis of solid digital knowledge.

Seventh SPLICE Workshop at SIGCSE 2021 "CS Education Infrastructure for All III: From Ideas to Practice", SPLICE'21, March 15-16, 2021, Virtual Event

Being an informed digital citizen requires also the ability to understand other peoples' code as well as to express one's own thoughts in a formal way and being able to communicate and discuss decisions when implementing program logic. Learning not just to implement a given logic, but to develop own solutions, alter them and defend them against critic, should therefore be learned in a collaborative way. Several schools have already taken up that challenge. Despite being lead by committed individuals, these efforts often face essential problems. Most schools simply cannot invest sufficient time and money in solid infrastructure to support practical programming assignments. Ensuring a reliable environment for 30+ pupils per class is already a challenging task during initial setup and requires know-how in virtualization or remote setup for desktop PCs. In the long run, the maintenance effort for such approaches will further diminish the outcome that teachers can achieve, even when spending additional efforts in their spare time.

Another issue is the availability of tested and trusted educational content. For traditional subjects, schools often rely on textbooks supplied by educational publishers. For computer science, as of now, most teachers, however, rely on their own material or material they got from their colleagues. The creation of (particularly auto-gradable) programming assignments tends to be very time-consuming and is requiring a profound knowledge of the employed programming language as well as suitable testing approaches. The body of available exercises is therefore limited. A different issue is the absence of local fellow learners. Some form of digital communication tool, therefore, also promises to be beneficial. While being able to consume online content, further mastery of the acquired knowledge as well as the practical experience of discussing one's ideas would be prevented, when not having access to a school or a group currently learning the content at hand together. We present prototypical solutions for the challenges mentioned above and propose further ideas for discussion to leverage learning outcomes and digital maturity.

EARLY EXPERIMENTS

Back in 2013, when we started our first experiments with JavaScript programming exercises in the course *Web-Technologies*[1], implementing a customized solution—automated or peer-review-based—was out of question in terms of effort and timing. Therefore, we evaluated several third-party, web-based coding tools as a quick and cheap alternative and finally decided to use JSFiddle[2]. To assess the students' solutions, we employed a methodology, which we called STEAP (Solution Through Execution Assessment Pattern). The basic idea was to prepare a programming problem in a publicly available online tool, along with a piece of code that is able to evaluate the participant's solution. The evaluation code returns a password if the participant's solution provided the correct results. The participant then had to copy this password and paste it in a free-text-question in a standard quiz on the openHPI MOOC platform [9].

The next step towards auto-graded programming exercises on our MOOC platform was the course *Spielend Programmieren lernen!*[3]. It was the first pure programming course on our platform and intended to teach the Python programming language to school kids. To provide the participants with the possibility to write and execute code in their browser, *WebPython* was developed. The tool supported two modes, console mode for text and number-based exercises, and turtle mode for graphical output. Turtle graphics is a programming model developed by Seymour Papert in the 1970s and a core part of the Python standard library [4]. Python programming literature for children typically leverages this support [1, 3]. *WebPython* transmitted the participants' code for execution and evaluation to our servers [12].

CODEOCEAN

WebPython had several issues and, therefore, inspired us to implement a different solution:

- It required the hardware of the FutureSoc-Lab[4], which we cannot use on a regular basis.
- It only supported the Python programming language.
- Its implementation was prototypical and not really designed for long-term productive use.

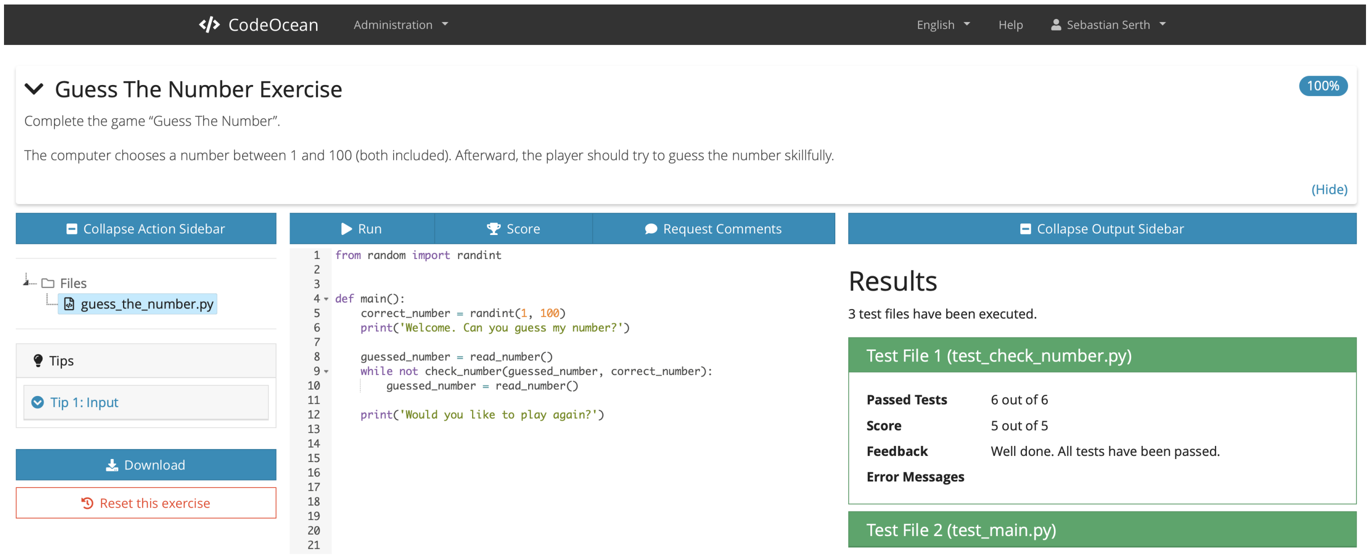
Therefore, *CodeOcean* was developed as an open-source tool[5]. Before starting the development, an intensive literature review was conducted to examine the state-of-the-art of such tools [7]. *CodeOcean* supports, at least in theory, every programming language that can be executed on a Linux operating system. In practice, we made use of *CodeOcean* for Java, Python, Ruby, JavaScript, and R exercises, in several courses by now. These courses range from offline seminars at our institute to full-scale MOOCs on our MOOC platforms with tens of thousands of registered participants. We have also employed *CodeOcean* to conduct a course on test-driven development with JUnit. The participants had to write test cases and an implementation for each exercise. While we were not able to guarantee that they have actually been working test-driven, *CodeOcean* at least allowed us to evaluate if they had written test-cases that covered the requirements of the exercises [10].

Enabling additional programming languages only requires writing an adapter for the output of the testing framework that is to be employed and generating a Docker image that contains all required components. For each programming language, we created a dedicated set of a testing adapter suitable for the employed testing framework and a custom Docker image containing language-specific tools. This allows an exercise author to use the standard testing frameworks of each language, e.g., PyUnit for Python, JUnit for Java, and RSpec for Ruby. The respective testing adapter for each framework extracts meta information (such as the number of failed tests) and test-specific error messages. So far, an additional adapter was written for C++ but has not been used yet in a course. We further created an adapter to parse the output of PyLint[6] as a static code analysis tool. In addition to the test output, it provides suggestions on improving the code style based on predefined rules. It enabled us to employ a new set of exercises focusing on different aspects of software engineering (such as understanding and refactoring of existing code).

Similar to *WebPython*, *CodeOcean* is a web-based development environment composed of a client-side code editor and a server-side component for code execution. An advantage of this approach is that it allows us to provide learners with a homogeneous and novice-friendly programming environment. It simplifies the support of multiple programming languages and third-party libraries while providing a consistent workflow for both code execution and assessment. And very importantly, it allows us to collect insights into learners' problem-solving strategies by analyzing their code submissions. *CodeOcean* promotes the concept of files within exercises. Multiple files can be editable per exercise, while the instructors have every possibility to individually restrict read and write access to each file. *CodeOcean* does not restrict the number of executions before an exercise has to be submitted. Its development environment is based on widespread web standards that are natively supported by current web browsers. Hence, learners can interact with their program during execution and provide input for the command line or turtle graphics. Figure 1 depicts the user interface for learners when solving an exercise. Most of the time, learners can focus on the implementation with the code editor. The right pane which is expanded in Figure 1 is hidden per default and only shown if needed to reduce distraction for novices.

Through its Learning Tools Interoperability (LTI) interface, *CodeOcean* can be attached to any Learning Management

[1]https://open.hpi.de/courses/www
[2]https://jsfiddle.net
[3]https://open.hpi.de/courses/pythonjunior2014
[4]https://hpi.de/en/research/future-soc-lab.html
[5]https://github.com/openHPI/codeocean
[6]https://www.pylint.org



Guess The Number Exercise

Complete the game “Guess The Number”.

The computer chooses a number between 1 and 100 (both included). Afterward, the player should try to guess the number skillfully.

```
from random import randint


def main():
    correct_number = randint(1, 100)
    print('Welcome. Can you guess my number?')

    guessed_number = read_number()
    while not check_number(guessed_number, correct_number):
        guessed_number = read_number()

    print('Would you like to play again?')
```

Results

3 test files have been executed.

Test File 1 (test_check_number.py)

Passed Tests	6 out of 6
Score	5 out of 5
Feedback	Well done. All tests have been passed.
Error Messages	

Test File 2 (test_main.py)

Figure 1. Successfully evaluated programming assignment in *CodeOcean*. Below the exercise description (top), a three-pane view is available to learners. The left sidebar is used to switch between different files or get support through tips, the center contains a code editor for working on the assignment and the right sidebar is shown on request with test output.

System (LMS) that supports this standard. Next to openHPI, these are e.g. Moodle[7], Sakai[8], Blackboard[9], or openEdX[10], to name just a few. See Staubitz et al. [8] for further implementation details. The tool had a rough start going from a prototypical implementation directly into a real course with 10,000 enrolled participants. While the testing adapters and overall web server worked as expected, the large number of parallel users in the first course showed a performance bottleneck of the single-threaded Docker service on our machine. As a result, the startup time of containers was too long for our interactive use case and necessitated pre-warming of idle containers for immediate use. By now, *CodeOcean* runs smooth and doesn't require much troubleshooting. During a recent course, it provided about 2.5 million code executions per month.

An increasingly expressed interest by school and university teachers motivated us to identify and tackle their needs regarding a stable programming environment for use in class scenarios. Hence, most recently, *CodeOcean* gained support for various configuration options through LTI allowing teachers to customize the experience learners have with the tool at any time. As of now, we added support for 14 independent options according to requests by teachers (i.e., to adapt the features offered by *CodeOcean* to comply with regulations for graded exercises in university courses). We further use LTI to mirror course participants to *CodeOcean* and automatically create so-called study groups with the respective users. The formation of study groups supports high-school teachers and university instructors to monitor the progress of their students and provide them with individual assistance whenever needed.

In distance education with a large number of users, individual help by course instructors is almost impossible due to the sheer volume of requests. Therefore, we designed *CodeOcean* to minimize upcoming questions and support the help-seeking behavior of learners. Thus, *CodeOcean* provides three different feedback mechanisms to learners: (1) Annotations for uncaught runtime exceptions, (2) exercise-specific hints, and (3) the possibility to ask fellow students questions regarding their implementation. Depending on the situation, these assistance features allow students to get tailored support. For example, the annotations for runtime exceptions are automatically shown when a learner's program terminates unexpectedly with an exception. Annotations are intended to rephrase the error message and thereby offer students another approach to identify the root cause of their issue.

If at any time, learners are unsure how to proceed with an exercise, they can access teacher-defined hints tailored for the current exercise. Depending on the assignment, these tips may contain a reminder of core programming concepts or exercise-specific hints. Multiple tips allow teachers to define incremental support steps for learners. Third, *CodeOcean* enables students to share their current implementation with fellow students and ask for comments regarding their questions. Targeted for beginners, this feature ensures that volunteers dedicating resources to help fellow learners have all information required to get a full picture of the question and provide code-level support.

Throughout the past years, *CodeOcean* has evolved to an educational development environment with support for various learning scenarios. New exercise types allow instructors enhanced flexibility in teaching relevant concepts and no longer limit them on grading through unit tests. For more advanced use cases, *CodeOcean* can be used in combination with traditional integrated development environments (IDEs). A plugin

[7] https://docs.moodle.org/310/en/LTI_and_Moodle

[8] https://www.sakailms.org/feature-details

[9] https://help.blackboard.com/Learn/Administrator/SaaS/Integrations/Learning_Tools_Interoperability

[10] https://open.edx.org/blog/open-edx-lti-tool-provider/

for Eclipse[11], a popular Java IDE, and a package for R enables teachers to transition from *CodeOcean* to a local environment with their students. The interface allows instructors to preserve the ability to use server-side assessment and grading capabilities. Learners can then transmit the score gained through the locally edited source code using LTI to the LMS.

CODEHARBOR

We define auto-graders as software tools that help instructors to grade programming assignments of their students according to some pre-defined criteria. We distinguish between dynamic testing approaches and static testing approaches including style-checkers. While dynamic approaches check the functionality of the handed-in assignments according to the requirements of the given exercise, static approaches check the code for possible flaws in the implementation or coding style issues. Some tools allow a combination of both methods.

The effort that is required to create auto-graded exercises is one of the major obstacles that we identified why these are often not provided in a sufficient quantity. This is particularly true for exercises that rely mostly on dynamic evaluation. Particularly, for computer science education in schools, MOOCs that come with a sufficient amount of programming exercises bear a great potential [2]. Using a form of a digital worksheet, they can resemble traditional materials and combine them with interactive content from MOOCs [6].

An experiment that we conducted in 2016—with two teachers and thirty pupils of a local school— signaled that with certain adjustments, the MOOC format can fit the special requirements of schools pretty well. The results of a similar experiment that we conducted in 2017 at a larger scale—invited were all MINT-EC schools and about 1000 pupils participated—promise to confirm the findings of the first experiment. One of the findings of several workshops that we conducted 2015 and 2016 with teachers both at a principal meeting of German schools engaging in the area of STEM[12] and a state meeting of Berlin-Brandenburg computer science teachers[13] was that many computer science teachers in schools lack either time or the ability or both to create such exercises. A survey that we conducted among the teachers that participated in the above-mentioned larger experiment, confirms this finding.

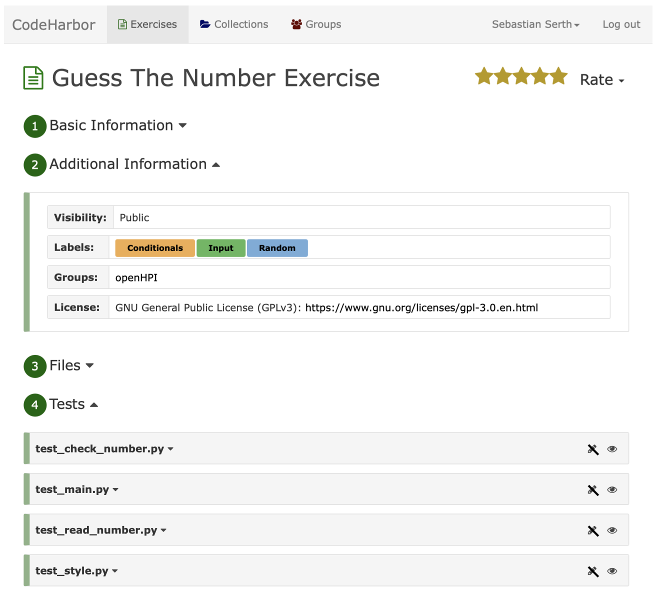
We, therefore, came up with the idea to provide a simple platform that allows sharing auto-gradable coding exercises: *CodeHarbor*. It allows sharing these exercises between auto-graders such as *CodeOcean* or others, e.g. *Praktomat*, by employing the ProFormA data format [11] that has been developed by the eCult-project[14]. The ProFormA specification is used as a foundation for the internal representation of exercises in *CodeHarbor* and the data exchange between *CodeOcean* and *CodeHarbor*. Interoperability with other ProFormA-compliant tools has been tested successfully. In addition to the standard compliance, *CodeHarbor* offers the most requested features of

[11]https://www.eclipse.org

[12]https://www.mint-ec.de/angebote/angebote-fuer-schulleitungen/mint-ec-schulleitertagung/

[13]https://ibbb.cses.informatik.hu-berlin.de/events/15-gi-tagung-zur-schulinformatik-in-berlin-und-brandenburg

[14]http://www.ecult-niedersachsen.de/


CodeHarbor Exercises Collections Groups Sebastian Serth Log out

Guess The Number Exercise Rate

1 Basic Information

2 Additional Information

Visibility:	Public
Labels:	Conditionals Input Random
Groups:	openHPI
License:	GNU General Public License (GPLv3): https://www.gnu.org/licenses/gpl-3.0.en.html

3 Files

4 Tests

test_check_number.py

test_main.py

test_read_number.py

test_style.py

Figure 2. The same exercise as shown in Figure 1 with additional information is available on *CodeHarbor*. Teachers can use additional labels to classify the exercise and add a license information. Further, the test files used for assessment are available so that other instructors can re-use the assignment in their context.

an exercise repository: It enables course instructors to create and share exercises among each other, tracks changes with built-in versioning support, and offers collections to group similar exercises. As shown in Figure 2, the shared exercises include test files and additional metadata.

CodeHarbor[15] is currently entering a public beta phase. It is designed as an open-source project and available on GitHub[16].

FUTURE WORK

The next steps on the agenda for our toolset are to deepen the integration of *CodeOcean* and *CodeHarbor* by moving the editing features closer to *CodeHarbor*, to ensure better interoperability of both tools with other systems, and to promote *CodeHarbor* so that a lively community will evolve. Particularly, *CodeHarbor* is dependent on such a community by definition. To just make reusing our own exercises for upcoming courses easier, it would have been easier to introduce the concept of an exercise repository directly to *CodeOcean*. *CodeHarbor* follows a more general strategy, however. The ability to share exercises between different auto-graders, to fork exercises, and to review exercises within a larger community will go way further than just reusing one's own exercises. To promote the idea we present the toolset at conferences such as this one and are continuously communicating with teachers and other projects, such as the MERLOT project[17], eCult[14], or X5gon[18]. In order to further ease the discoverability of programming exercises stored in *CodeHarbor*, we plan to automatically list available assignments in educational

[15]https://codeharbor.openhpi.de

[16]https://github.com/openHPI/codeharbor

[17]https://www.merlot.org

[18]http://www.k4all.org/project/x5gon

catalogs, such as EduSharing instances[19]. By supporting alternative data exchange formats such as the Programming Exercise Markup Language (PEML)[20], the interoperability of *CodeHarbor* could be increased.

In *CodeOcean*, we further plan to improve support for more sophisticated programming exercises in order to teach advanced topics in software engineering. Therefore, we aim to investigate how a static code analysis tool could be integrated comprehensively to provide added value for students. While these tools are commonly used in professional software development, only a few courses for novices apply these techniques. Another aspect found across software engineers is to use pair programming techniques for complex tasks. Previous research suggested a positive impact of using pair programming with beginners but rarely dealt with the required adaption of the concept to distance education [5]. This will not only teach the basic principles of pair programming but will also foster collaboration among participants. To enable programming courses with ECTS points, we're planning to add support for proctored programming assignments to *CodeOcean*. This would seamlessly supplement the face recognition solution, which we are already using for our quiz-based assignments in openHPI. On a technical level, we will evaluate which prerequisites machine learning courses imply on *CodeOcean* and how to further improve the scalability of our services.

CONCLUSION

We have come a long way from our first steps with the STEAP exercises in 2014 to our current possibilities concerning auto-graded exercises in our MOOCs using *CodeOcean*. Programming assignments in *CodeOcean* may contain unit tests and style checkers to provide instant feedback to learners. When questions arise while solving a task, students can use the built-in assistance features of the platform to get individual support. The toolset we developed for our platform covers most of our current needs. In order to strengthen the efforts towards auto-gradable exercises as open educational resources, we encourage an open discussion of educators' needs towards a coding exercise repository. We propose our first version of *CodeHarbor* as a starting point towards such a discussion.

REFERENCES

[1] Jason R. Briggs. 2012. *Python for Kids: A Playful Introduction to Programming*. No Starch Press, San Francisco, CA. 348 pages.

[2] Catrina Tamara Grella, Thomas Staubitz, Ralf Teusner, and Christoph Meinel. 2017. *Can MOOCs Support Secondary Education in Computer Science?* Springer International Publishing, Cham, 478–493. DOI: `http://dx.doi.org/10.1007/978-3-319-50337-0_45`

[3] Gregor Lingl. 2010. *Python für Kids* (4 ed.). bhv, Heidelberg, Germany. 432 pages.

[4] Seymour A. Papert and Cynthia Solomon. 1971. Twenty Things To Do With A Computer. *MIT Artificial Intelligence Memo* 248 (01 June 1971), 41. `http://hdl.handle.net/1721.1/5836`

[19] https://edu-sharing.com

[20] https://cssplice.github.io/peml

[5] Sebastian Serth. 2019. Integrating Professional Tools in Programming Education with MOOCs. In *2019 IEEE Frontiers in Education Conference (FIE)*. 1–2. DOI: `http://dx.doi.org/10.1109/FIE43999.2019.9028643`

[6] Sebastian Serth, Ralf Teusner, Jan Renz, and Matthias Uflacker. 2019. Evaluating Digital Worksheets with Interactive Programming Exercises for K-12 Education. In *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE Press, 1–9. DOI: `http://dx.doi.org/10.1109/FIE43999.2019.9028680`

[7] Thomas Staubitz, Hauke Klement, Jan Renz, Ralf Teusner, and Christoph Meinel. 2015. Towards practical programming exercises and automated assessment in Massive Open Online Courses. In *IEEE International Conference on Teaching, Assessment, and Learning for Engineering, TALE 2015, Zhuhai, China, December 10-12, 2015*. IEEE, 23–30. DOI: `http://dx.doi.org/10.1109/TALE.2015.7386010`

[8] Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz, and Christoph Meinel. 2016. CodeOcean - A versatile platform for practical programming excercises in online environments. In *2016 IEEE Global Engineering Education Conference, EDUCON 2016, Abu Dhabi, United Arab Emirates, April 10-13, 2016*. IEEE, 314–323. DOI: `http://dx.doi.org/10.1109/EDUCON.2016.7474573`

[9] Thomas Staubitz, Jan Renz, Christian Willems, Johannes Jasper, and Christoph Meinel. 2014. Lightweight ad hoc assessment of practical programming skills at scale. In *2014 IEEE Global Engineering Education Conference, EDUCON 2014, Istanbul, Turkey, April 3-5, 2014*. IEEE, 475–483. DOI: `http://dx.doi.org/10.1109/EDUCON.2014.6826135`

[10] Thomas Staubitz, Ralf Teusner, Christoph Meinel, and Nishanth Prakash. 2016. Cellular Automata as basis for programming exercises in a MOOC on Test Driven Development. In *IEEE International Conference on Teaching, Assessment, and Learning for Engineering, TALE 2016, Bangkok, Thailand, December 7-9, 2016*. IEEE, 374–380. DOI: `http://dx.doi.org/10.1109/TALE.2016.7851824`

[11] Sven Strickroth, Michael Striewe, Oliver Müller, Uta Priss, Sebastian Becker, Oliver Rod, Robert Garmann, Oliver J. Bott, and Niels Pinkwart. 2015. ProFormA: An XML-based exchange format for programming tasks. *eleed* 11, 1 (2015). `http://nbn-resolving.de/urn:nbn:de:0009-5-41389`

[12] Martin von Löwis, Thomas Staubitz, Ralf Teusner, Jan Renz, Christoph Meinel, and Susanne Tannert. 2015. Scaling youth development training in IT using an xMOOC platform. In *2015 IEEE Frontiers in Education Conference, FIE 2015, El Paso, TX, USA, October 21-24, 2015*. IEEE Computer Society, 1–9. DOI: `http://dx.doi.org/10.1109/FIE.2015.7344145`