openHPI’s Coding Tool Family: CodeOcean, CodeHarbor, CodePilot

Thomas Staubitz[1], Ralf Teusner[2], Christoph Meinel[3]

Abstract: The Hasso Plattner Institute successfully runs a self-developed Massive Open Online Course (MOOC) platform—openHPI—since 2012. MOOCs, even more than classic 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 maps the landscape of tools that are used on openHPI in the context of automated grading of programming exercises. Furthermore, it provides a sneak preview to new features that will be integrated ion the near future. Particularly, we will introduce CodeHarbor, our platform to share auto-gradeable exercises between various online code execution platforms.

Keywords: MOOC; Scalability; Programming; Autograder; Code Repository; Sharing

1 Introduction

Academia, industry as well as politics nowadays agree that future generations should learn the skills to take part in digital life as part of a 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 skillset 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, a profound knowledge of syntax and control structures of at least one major 4th generation programming language is just the basis of a solid digital knowledge. 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

[1] University of Potsdam, Faculty of Digital Engineering, Prof.-Dr.-Helmert-Str. 2-3, Germany, thomas.staubitz@hpi.de
[2] University of Potsdam, Faculty of Digital Engineering, Prof.-Dr.-Helmert-Str. 2-3, Germany, ralf.teusner@hpi.de
[3] University of Potsdam, Faculty of Digital Engineering, Prof.-Dr.-Helmert-Str. 2-3, Germany, christoph.meinel@hpi.de

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 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 ones 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.

2 Early Experiments

Back in 2013, when we started our first experiments with Javascript programming exercises in the course *Web-Technologies*[4], 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[5]. 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. The idea was quite successful, but had strong limitations. It only worked for few programming languages, and the evaluation was done on the client-side, which created possibilities for cheating. Back then about 40% out of 3,172 active participants submitted a solution for at least one of the practical bonus exercises. About 70% out of those who started, submitted solutions for all eight bonus tasks. The workload to complete these exercises was way higher than the workload required to complete a multiple-choice homework, so these numbers were convincing enough to take further steps [St14].

[4] https://open.hpi.de/courses/www
[5] https://jsfiddle.net

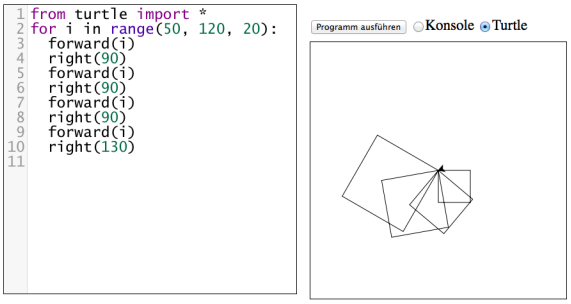

Abb. 1: WebPython in Turtle mode

The next step towards auto-graded programming exercises on our MOOC platform, was the course *Spielend Programmieren lernen!*[6]. It was the first pure programming course on our platform and intended to teach the Python programming language to school kids. The targeted age group ranged from 12 - 18 years. To provide the participants' with the possibility to write and execute code in their browser, *WebPython* was developed by Prof. Martin v. Löwis, who also conducted the course. The tool supported two modes, console mode for text and number based exercises, and turtle mode for graphical output (see Fig.1). Turtle graphics [PS71] is a programming model developed by Seymour Papert in the 1970s and a core part of the Python standard library. Python programming literature for children typically leverages this support [Br12] [Li10]. *WebPython* transmitted the participants' code for execution and evaluation to our servers. While this step made more difficult for the participants to cheat, compared to our previous solution with client-side evaluation, it introduced a scalablity problem. *WebPython* approached this problem by making use of the hardware capacities of the HPI's FutureSoc-Lab[7]. Even those machines found their masters' in an armada of programming novices writing their first *while-loops* [Lö15].

3 CodeOcean

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

- It required the hardware of the FutureSoc-Lab, which we only can borrow at certain points of time and not on a regular basis.
- It only supported the Python programming language.
- It's implementation was prototypical and not really fit for a long-term productive use.

[6] https://open.hpi.de/courses/pythonjunior2014
[7] https://hpi.de/en/research/future-soc-lab.html

Therefore, *CodeOcean* was developed and made available open source on GitHub[8]. Before the tool was developed, an intensive literature review was conducted to examine the state-of-the-art of such tools [St15a]. 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, and JavaScript exercises, in a number of courses by now. These courses range from offline seminars at our institute to full-scale MOOCs on openHPI with 10,000 registered participants. We have also employed *CodeOcean* to conduct a course on test-driven development. 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[St16a]. Enabling additional programming languages only requires to write an adapter for the output of the testing framework that is to be employed and to generate a docker image that contains all required components.

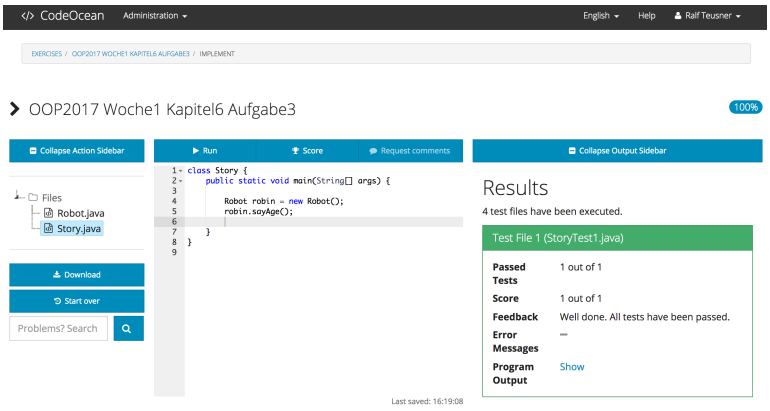

Abb. 2: Successfully evaluated coding assignment.

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 amount 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. Via it's Learning Tools Interoperability (LTI) interface, *CodeOcean* can be

[8] https://github.com/openHPI/codeocean

attached to any Learning Management System (LMS) that supports this standard. Next to openHPI, these are e.g. Moodle[9], Sakai[10], Blackboard[11], or openEdX[12], to name just a few. See Staubitz et al [St16b], for further implementation details. The tool had a rough start going from a prototypical implementation directly into a real course of 10.000 enrolled participants. By now it runs smooth and doesn't require much troubleshooting during the courses.

4 CodeHarbor

We define auto-graders as software tools that help instructors to grade handed-in programming assignments of their students according to some pre-defined criteria. Basically, 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 for 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 an sufficient amount of programming exercises bear a great potential [Gr17]. An experiment that we conducted in 2016—with two teachers and thirty pupils of a school in Riesa—signalled 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 the MINT-EC's principal meeting[13] and the meeting of Berlin-Brandenburg computer science teachers[14], was that many computer science teachers in schools lack either time or 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 to share auto-gradable coding exercises: *CodeHarbor*. *CodeHarbor* will allow to share these exercises between auto-graders such as *CodeOcean* or *Praktomat* by employing the Pro-Form-A data format [St15b] that has been

[9] https://docs.moodle.org/33/en/LTI_and_Moodle
[10] https://sakaiproject.org/extending-sakai
[11] https://help.blackboard.com/Learn/Administrator/SaaS/Integrations/Learning_Tools_Interoperability
[12] https://open.edx.org/learning-tools-interoperability
[13] https://www.mint-ec.de/angebote/angebote-fuer-schulleitungen/mint-ec-schulleitertagung/
[14] https://ibbb.cses.informatik.hu-berlin.de/events/15-gi-tagung-zur-schulinformatik-in-berlin-und-brandenburg

developed by the eCult-project[15]. *CodeHarbor*[16] is currently entering a public alpha phase. It is designed as an open source project and available on GitHub[17].

5 CodePilot

CodePilot is a video conferencing solution that we integrated into CodeOcean for tutoring purposes. Participants are notified when a member of the teaching team is online and ready to provide feedback. If they desire they can establish a direct video-chat with the teaching team member on tutoring duty. The current state of their solution, as well as all modifications during the session and the grading results are mirrored to the tutor. The tutor is not able to modify the code of the participants, in order to ensure that misunderstood concepts are resolved and the participants are enabled to fix their code themselves.

Concerning the desirability of this feature, we ran a small live test as well as two surveys. Overall, we got mixed results. In the course *Java Workshop - Einführung in die Testgetrieben Entwicklung mit JUnit (2016)*, more than 75% of all participants that have answered the survey deemed the possibility to get direct contact to a teaching team member as valuable or even very valuable (options 4 and 5 of a 5-point Likert scale). Also, the live test yielded positive feedback from participants that gave the feature a try. However, it became apparent that often technical issues on participants side, such as missing microphones or cameras, complicated the sessions. In the course *Objektorientierte Programmierung in Java (2017)*, we went for more detailed feedback in a survey with 1638 participants. 38% of them expressed concerns with regard to their privacy and stated that they would not use the feature. 21% would use the feature to contact the teaching team, 19% stated that they do not have the technical requirements to use the feature at all. 12% would primarily use the feature to collaborate with their teammates, and 10% would even be willing to contribute in some way to get access to that feature.

Given these answers, we will mostly rely on simpler collaboration measures first, and focus on further experiments with regards to user acceptance before placing CodePilot more prominently in the coding environment.

6 FutureWork

The next steps on the agenda for our toolset are to promote CodeHarbor, so that a lively community will evolve. This particular tool 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 question banks directly to CodeOcean. CodeHarbor

[15] http://www.ecult-niedersachsen.de/
[16] https://tools.openhpi.de/codeharbor
[17] https://github.com/openHPI/codeharbor

follows a more general strategy however. The ability to share exercises between different auto-graders, to fork exercises, 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[18], eCult[19], or X5gon[20]. CodePilot, although many participants have privacy issues with using their webcam, also has its use cases, maybe these are to be found in another direction than we initially had planned. Pair programming tasks would be one of the first things that come to mind. Also audio only chats would be an option. To enable programming courses with ECTS points, we're planning to add a keyboard pattern based user identification to CodeOcean. This would seamlessly supplement the face recognition solution, which we are already using for our quiz based assignments. Finally, we will add static evaluation options for the grading mechanisms in CodeOcean.

7 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. The toolset we developed for our platform covers most of our current needs. In order to strengthen the efforts towards auto-gradeable 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.

Literaturverzeichnis

[Br12] Briggs, Jason R.: Python for Kids. No Starch Press, San Francisco, 2012.

[Gr17] Grella, Catrina Tamara; Staubitz, Thomas; Teusner, Ralf; Meinel, Christoph: Can MOOCs Support Secondary Education in Computer Science? In (Auer, Michael E.; Guralnick, David; Uhomoibhi, James, Hrsg.): Interactive Collaborative Learning: Proceedings of the 19th ICL Conference - Volume 1. Springer International Publishing, Cham, S. 478–493, 2017.

[Li10] Lingl, Gregor: Python für Kids. bhv, Heidelberg, 4. Auflage, 2010.

[Lö15] Löwis, M.; Staubitz, T.; Teusner, R.; Renz, J.; Meinel, C.; Tannert, S.: Scaling youth development training in IT using an xMOOC platform. In: 2015 IEEE Frontiers in Education Conference (FIE). S. 1–9, Oct 2015.

[PS71] Papert, Seymour; Solomon, Cynthia: Twenty Things To Do With A Computer. MIT Artificial Intelligence Memo, (248), 1971.

[St14] Staubitz, Thomas; Renz, Jan; Willems, Christian; Jasper, Johannes; Meinel, Christoph: Lightweight Ad Hoc Assessment of Practical Programming Skills at Scale. 10.1109/EDUCON.2014.6826135. IEEE, S. 475–483, 2014.

[18] https://www.merlot.org/merlot/index.htm

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

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

[St15a] Staubitz, T.; Klement, H.; Renz, J.; Teusner, R.; Meinel, C.: Towards practical programming exercises and automated assessment in Massive Open Online Courses. In: 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE). S. 23–30, Dec 2015.

[St15b] Strickroth, Sven; Striewe, Michael; Müller, Oliver; Priss, Uta; Becker, Sebastian; Rod, Oliver; Garmann, Robert; Bott, Oliver J.; Pinkwart, Niels: ProFormA: An XML-based exchange format for programming tasks. eleed, 11(1), 2015.

[St16a] Staubitz, T.; Teusner, R.; Meinel, C.; Prakash, N.: Cellular Automata as basis for programming exercises in a MOOC on Test Driven Development. In: 2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE). S. 374–380, Dec 2016.

[St16b] Staubitz, Thomas; Klement, Hauke; Teusner, Ralf; Renz, Jan; Meinel, Christoph: CodeOcean - A versatile platform for practical programming excercises in online environments. In: 2016 IEEE Global Engineering Education Conference (EDUCON). S. 314–323, April 2016.