Towards a Repository for Open Auto-Gradable Programming Exercises

Thomas Staubitz, Ralf Teusner, Christoph Meinel
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
{firstname.lastname}@hpi.de

***Abstract*—Auto-gradable hands-on programming exercises are a key element for scalable programming courses. A variety of auto-graders already exist, however, creating suitable high-quality exercises in a sufficient amount is a very time-consuming and tedious task. One way to approach this problem is to enable sharing auto-gradable exercises between several interested parties. School-teachers, MOOC[1] instructors, workshop providers, and university level teachers need programming exercises to provide their students with hands-on experience. Auto-gradability of these exercises is an important requirement. The paper at hand introduces a tool that enables the sharing of such exercises and addresses the various needs and requirements of the different stakeholders.**

Keywords—Open Educational Resources; MOOC; Massive Open Online Courses; Auto-Grader; Assignment; Assessment

I. INTRODUCTION

Auto-gradable programming exercises are an essential feature of programming courses. In MOOCs as well as in universities or schools, manual grading of these exercises is often not an option. In MOOCs, it is impossible due to the sheer amount of submissions. In schools, teachers often lack the time or the skills to create and grade such exercises. Universities often do not pay for the time that instructors spent with grading[2].

So-called auto-graders are abundantly available by now. The development of these tools has started back in the 1960s and they have been constantly developed further in several generations since then [1]. In general, we can define auto-graders as software tools that help instructors to grade programming assignments of their students according to some pre-defined criteria. Most common are dynamic and static testing approaches. 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.

Our team also has developed an auto-grader (*CodeOcean*[3]) for the purposes of our MOOC platform, which offers the ability to write and execute code within a browser [2]. Given that the instructors have provided suitable unit tests, *CodeOcean* also offers the ability to automatically grade the submitted code. Static testing approaches are in the pipeline but have not been implemented yet. *CodeOcean* is only loosely coupled with our MOOC platform via the Learning Tools Interoperability (LTI)[4] interface and can, therefore, also be used with other MOOC platforms or Learning Management Systems, which support this standard: *Moodle*[5] or *Open edX*[6], just to name a few. *CodeOcean* is open source and freely available for usage and contribution.

While static testing and checks for code-style often make use of existing libraries and frameworks, dynamic testing requires the instructors to implement tests, customized for each exercise. This requires a certain amount of expertise and time on the side of the instructors, which in many cases is missing.

Particularly in schools, this is a serious issue. Computer science education in the German school system leaves a lot to improve. Only in three of the sixteen federal states it is mandatory, in the others it often hardly exists. Qualified teachers for computer science are rare. Already in 2014, we have offered a programming MOOC (*PythonJunior*[7]), which particularly targeted school children [20]. A second iteration of this course was conducted in 2015. To identify the reasons why less teachers participated in these MOOCs with their classes than we expected, we conducted workshops with teachers on the usage of MOOCs in class during the MINT-EC[8] headmasters conference[9] and at a networking day[10], which was

1 Massive Open Online Course

2 Particularly, if the instructors are part-time lecturers instead of regular faculty

3 https://github.com/openHPI/codeocean

4 http://www.imsglobal.org/activity/learning-tools-interoperability

5 https://moodle.org/

6 https://open.edx.org/

7 The original title of the course was "Spielend Programmieren lernen!" throughout the rest of this paper we will use the shorter "PythonJunior".

8 https://www.mint-ec.de/

9 Würzburg, Germany, 2015

10 Potsdam, Germany, 2016

organized by the Gesellschaft für Informatik(GI)[11]. As one of the results of these workshops, we conducted a modified version of *PythonJunior* in collaboration with MINT-EC in 2016 and 2017.

One of the most appreciated features of our MOOC platform in these courses was the possibility to write and execute code in the browser. Many of the teachers we have interviewed during the workshops and the courses, appreciated the high-quality programming exercises. One of the top items on their wish-list, was the option to access the exercises outside of the context of the MOOC. As *CodeOcean* can be integrated with any LTI consumer, this is basically not a problem. However, the amount of available exercises on *CodeOcean* also is limited. To increase the amount of available exercises, a repository that allows to share these exercises as open educational resources between a wide variety of auto-graders is a viable solution. As a further result of these workshops, we have, therefore, decided to address this issue and have started to implement such a repository: *CodeHarbor*. Sharing, cloning discussing and rating the exercises are some of the features provided by this repository. The option to share such exercises is not only important for teachers but is also well suited to reduce the overall workload for instructors in MOOCs [3] and universities.

Section 2 provides a closer look at some selected auto-graders and presents some repositories in the context of open educational resources. It also discusses candidates for a data-exchange format to share these exercises between various auto-graders. Section 3 evaluates the requirements for such an exercise repository in different scenarios. Section 5 presents a high-level overview of the proposed platform and finally, Sections 6 and 7 detail our future work and conclude our findings.

II. Related Work

The topic of the paper at hand is basically related to three research areas.

- Auto-graders
- Learning object repositories
- Data-exchange formats

A. Auto-graders

The automatic evaluation of code submitted by students is a well-researched topic. Many institutions that are teaching computer science and programming have developed solutions to reduce the time instructors need to evaluate the source code submitted by their students. Hollingsworth proposed an auto-grader back in 1960 [4], Arnow and Barshay presented a web-based auto-grader (*WebToTeach*) in 1999 [5], Higgins et al. presented a tool to automatically assess Java programming exercises (*CourseMaker*) in 2005 [6], Venables and Haywood [7] in 2003 and Truong et al. [8] in 2005 worked on immediate feedback for programming students, Almajali came up with an auto-grader for advanced programing tasks in 2012 [9], El Balaa describes a programming language agnostic tool (*EMSEL*), which does not only grade exercises but supports students in finding exercises and instructors in creating such exercises in 2016 [10]. Efforts in cataloguing auto-graders have been made by Ala-Mutka in 2005 [11], Douce et al. also in 2005 [12], Ihantola et al. in 2010 [13], or Caiza and Ramiro in 2013 [14]. This list just mentions a few approaches and is far from being comprehensive.

Apart from syntactical correctness, which is validated "for free" by the compiler or interpreter, the code can be examined for style, possible flaws, and for correct functionality. Dynamic and static approaches are possible candidates to determine the grade. While static approaches examine either the source code or the compiled code, dynamic approaches usually run the code against test cases that check if the code acts according to the given specification and delivers the correct results. Static approaches provide information about software metrics, code quality, and possible flaws that might lead to misbehavior of the program. Dynamic approaches feed the submitted program with several inputs and check if the resulting output matches the defined specification. Static approaches have the advantage that libraries and frameworks exist, which check for the most common issues in many programming languages. Dynamic approaches have the advantage that they execute the code and, therefore, can tell if it provides the correct results. They have the disadvantage that they need to execute the code and, therefore, require special security arrangements. Furthermore, it is often not easy to employ dynamic approaches in very basic low level exercises, e.g. to test if students have defined a variable or not. A further disadvantage is that they, usually, require a customized solution for each exercise and, therefore, produce a high workload for the instructors.

In the following we will compare three open source auto-graders.

- *INGInious* – an auto-grader developed at the Université Catholique de Louvain (UCL) in Belgium
- *Praktomat* – an auto-grader developed at the Karlsruhe Institute of Technology (KIT) in Germany
- *CodeOcean* – the auto-grader developed at the Hasso Plattner Institute (HPI) in Germany

Further auto-graders exist, but have not been considered for this evaluation. We focused on these three for now as their source code is available on GitHub and they are sufficient to show that these tools follow similar schemas and that it, therefore, should be possible to develop a tool that allows the sharing of exercises between them.

INGInious[12][13] is implemented in *Python*[14] and uses *Docker*[15] containers to isolate the programs of students from

[11] https://en.gi.de/startpage.html

[12] http://www.inginious.org/

[13] https://github.com/UCL-INGI/INGInious

[14] https://www.python.org/

each other and the hosting environment. The pluggable architecture with *Docker* allows *INGInious* to support arbitrary programming languages. Students are enabled to write and edit their code from within the browser. For execution and evaluation, their code is submitted to *INGInious'* servers. *INGInious* mainly follows a dynamic testing approach [15], more recently a static code analysis has been added for *Oz*[16] [16], a multi-paradigm programming language, which has been developed at the UCL for educational purposes.

Praktomat is also implemented in *Python* and makes use of the *Django*[17] web framework [17]. According to Breitner et al. the programs of the students can either be executed as a separate user or within *Docker* containers [17]. *Praktomat* supports the languages Java, C/C++, Fortran, Haskell, R and Isabelle. Dynamic grading with unit tests is only supported for Java and Haskell. The students upload files containing their code to the *Praktomat* server, where they are compiled and evaluated. *Praktomat* supports both visible and hidden checks. In the visible checks the way how the code is evaluated is shown to the students, while for the hidden checks only the result is shown.

CodeOcean[18] is a *Ruby on Rails*[19] application and uses *Docker* containers for code isolation. Theoretically, it can execute and grade source code in any programming language. In practice, it requires a *Docker* container that is prepared to support this language and an adapter for the testing framework that is intended to be used. Currently, containers for *Python*, *Java*, *Ruby* and *Node* are maintained, adapters have been implemented for *PyUnit*[20], *JUnit*[21], *RSpec*[22], and *Mocha*[23], thus providing the possibility to use common native testing frameworks for each of the supported programming languages. Static evaluation is not supported yet, but a concept exists and is highly prioritized on the to-do list. First experiments introducing a static code checking tool have already been conducted.

B. Repositories and Sharing

The ability to share information at low cost with anyone anywhere is one of the internet's great contributions. We list some exemplary repositories in the educational area in this section. These examples show the feasibility of such efforts, some of them might even serve as potential partners for future cooperation.

MERLOT[24] is a curated collection of open educational resources (OER). Its main purpose is to collect meta-information about these resources and to increase their visibility. *4teachers*[25] allows (German) teachers to share their teaching materials with their colleagues. It provides a collection of teaching materials as OER. Furthermore, the site offers a discussion forum. Both systems might be possible candidates for future cooperation.

Moodle[26], Canvas[27], Sakai[28], and Blackboard[29] are learning management systems (LMS). These systems support the learning tools interoperability (LTI) interface, so that they can integrate exercises as provided by auto-graders, such as *CodeOcean*. Many of them support similar concepts as the one that we propose with *CodeHarbor* for quizzes instead of programming exercises. So-called question-banks allow instructors to collect, create, reuse, and share quiz questions throughout each platform. The difference to the idea that we promote with *CodeHarbor*, however, is that these question-banks are generally locked within the borders of one instance of these systems. Hence, instructors within one institution might share these quiz questions, but they do not allow to share them beyond the borders of the institution.

The Australian School of Audio Engineering (SAE)[30] with campuses in many major cities worldwide, runs a proprietary system[31] that allows its instructors to reuse quiz questions that have been created by instructors at any of those campuses. To access the questions of other instructors, each instructor must share some questions herself, to make sure that the repository is continuously growing and up to date with the courses' learning objectives.

Lon-Capa[32] is a repository for sharing educational content, including interactive exercises, among academic institutions. It is a distributed system and provides access to a substantial amount of exercises for a wide variety of subjects. So, finally, here we can see that the concept of sharing questions and exercises is possible and has been realized. However, as it is a very generalized tool, its options are very complex and it is not easy to use. In a meeting[33] with the eCULT[34] project, some of

---

15 Docker is a popular software that builds on top of Linux containers and allows to run isolated processes with less overhead than virtual machines or even separate hardware machines. Docker's original purposes are running several apps side by side on a single piece of hardware or to optimize the delivery pipelines. Many auto-graders make use of it to provide separated low-cost environments for the execution of the students' code. https://www.docker.com/

16 http://mozart.github.io/

17 https://www.djangoproject.com/

18 https://github.com/openHPI/codeocean

19 http://rubyonrails.org/

20 http://pyunit.sourceforge.net/

21 http://junit.org/junit4/

22 http://rspec.info/

23 https://mochajs.org/

24 https://www.merlot.org/merlot/index.htm

25 http://www.4teachers.de

26 https://moodle.org/

27 https://sakaiproject.org/

28 https://www.canvaslms.com/?lead_source_description=instructure.com_

29 http://www.blackboard.com/

30 http://www.sae.edu/?global

31 One of the authors knows this system from personal experience.

32 http://www.lon-capa.org/whatis.html

33 October 26, 2016 Skype Call

34 The eCULT project (eCompetences and Utilities for Learners and Teachers) is developing an exchange format for auto-gradable programming exercises. Several German universities are contributing to this project. We are not part of this project, but we are using the standard that they are developing

the participants expressed the desire for a more lightweight system for certain use cases.

C. Exchange Formats and Standards

To enable sharing programming exercises between a variety of auto-graders, a common definition of "exercise" is required. We examined the data-models of the selected auto-graders and evaluated existing standards and data exchange formats as possible candidates to be used in *CodeHarbor*.

We skipped SCORM[35] and CommonCartridge[36] rather early in our evaluation process. Although they are well-known formats in the e-learning context, they are far too heavy-weight and general for our purpose as they are targeted to move complete courses from one LMS to another. The *IMS Question & Test Interoperability® Specification (QTI)*[37] was the next candidate on our list. However, this standard was developed to exchange (multiple choice, etc.) quizzes between LMS systems and does not fit the special requirements of auto-gradable programming exercises. IEEE's *Learning Object Metadata (LOM)*[38] standard also does not fit. It will, however, add an additional value to the exercises in the future, e.g. for a potential cooperation with meta-repositories such as MERLOT.

Finally, we encountered the *ProFormA*[39] format, which is being developed by the eCULT project for a very similar purpose to ours. In their case, *Lon-Capa* takes the role of a shared repository for auto-graded programming exercises. The format mainly targets the auto-graders *Praktomat, JACK*[40], and a few others [18]. As a part of their research for the *ProFormA* standard, Strickroth et al. [19] have analyzed many different grading tools to determine a common superset of the different requirements. Their results confirm our analysis of the auto-graders' exercise structures. *ProFormA* is the best fit for our purposes so far, and has been chosen to serve as *CodeHarbor's* data exchange format.

III. Requirements Evaluation

An exercise sharing platform must address the needs of a variety of possible stakeholders and use cases. The requirements of these stakeholders differ with the size and structure of the audience and the learning situation in general. We focus on the following possible scenarios. While they have many requirements in common, there are also some differences:

- MOOCs
- Regular on-campus seminars
- SPOCs as educational resources in schools
- Workshops and CoderDojos

A. MOOCs

We can build on our own experience here as we have conducted about ten MOOCs during the last three years that provided scalable programming exercises and assignments to the participants to a varying extent. In four of these courses, the auto-graded assignments contributed most of the points to be gained by the participants. Furthermore, we have started to contact other groups that are providing MOOCs with auto-gradable programming exercises on an international level to discuss further requirements and promote the idea of sharing these exercises. One of the most important requirements that we identified here, is that the exercises need to be completely self-contained and self-explanatory as it is rather difficult to provide additional information on an individual level in this context. The heterogeneity of the participants' backgrounds requires exercises of various complexity levels.

B. Regular On-campus Seminars

Compared to MOOCs, the user base is more homogenous. Exams and exercises, generally, can be more demanding and time consuming. A suggested time constraint could be added to the exercises. Although we have some experience in this area ourselves, our focus is biased towards the MOOC scenario. We, therefore, asked our colleagues from the eCULT project for their requirements towards such a tool. Here, we will highlight just a few of their points. The tool itself needs to be open-source, the exercises ideally should be published under a Creative Commons[41] license. A well-defined exercise access model was listed as a high priority. The possibility to rate and comment on exercises was also highly prioritized. An important non-technical issue that was raised here, was the question who will host and maintain this repository in the long run. It needs to be highly available, reliable, and sustainable.

C. SPOCs in Schools as Additional Educational Resources

In 2016, we ran *PythonJunior* in an extended version as a SPOC (Small Private Online Course) for a school. Extended means that we took the original 4-weeks course and extended its runtime so that it fit with the school semester (12 weeks). We didn't change any content, we just allowed the participants more time. About 20 pupils from this school participated in the SPOC.

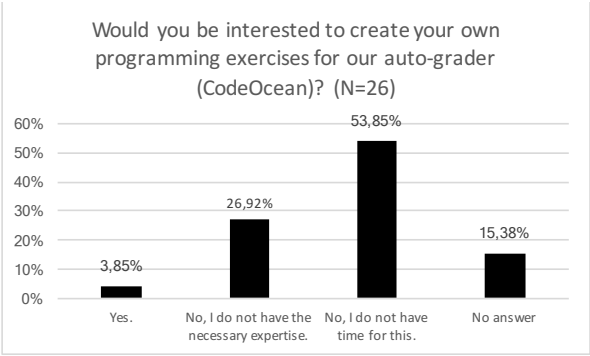

Fig. 1. Teachers lack expertise and time to create exercises.

in *CodeHarbor*. We are in contact with this project and we hope to deepen our cooperation in the future.

35 https://scorm.com/scorm-explained/

36 https://www.imsglobal.org/activity/common-cartridge

37 https://www.imsglobal.org/question/index.html

38 https://standards.ieee.org/findstds/standard/1484.12.1-2002.html

39 https://github.com/ProFormA

40 http://www.s3.uni-duisburg-essen.de/en/jack/

41 https://creativecommons.org/

We cooperated closely with two teachers from that school who accompanied their class during the course. The original *PythonJunior* MOOC was designed to teach Python programming to kids [20]. Before we ran it in the school setting, it has been run in two iterations as a regular 4-weeks course. In 2017, we repeated the experiment in a larger setting. This time we had more than 1000 participants from about 30[42] schools in the course, 41% of the pupils received a Record of Achievement at the end of the course. The no-show rate—users who register but never show up in the course—was particularly low: only 3%.

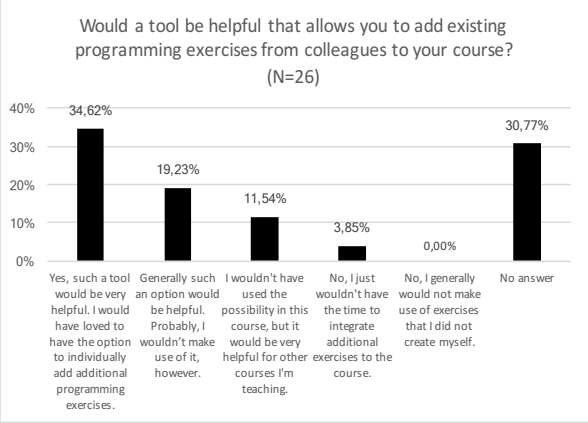

Fig. 2. Teachers see the usefulness of having such an exercise repository

In this context, we conducted a survey among the invited MINT-EC teachers. 32 teachers participated in that survey, 18 of them were using our course in different settings: from a regular class to unattended extracurricular activities[43]. In this survey, we also particularly asked for the teachers' interest in creating programming exercises of their own or in the possibility to select additional exercises from a larger pool.

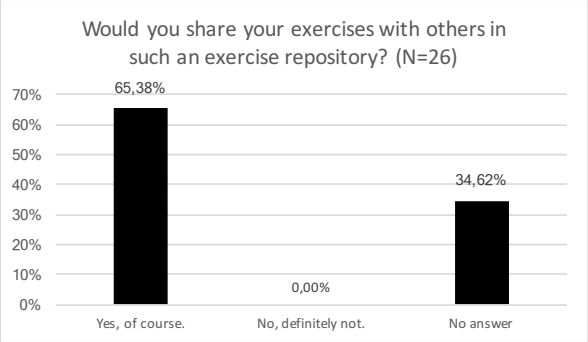

Fig. 3. Teachers are willing to share their material

More than 50% of the interviewed teachers answered that they do not have enough time for such activities. Another 30% stated that they lack the expertise to create such exercises (see Fig. 1). On the other hand, a substantial number of teachers stated that such an exercise repository and the possibility to make use of an auto-grader, such as *CodeOcean*, would be very helpful for their teaching, in classes based on our MOOC as well as in other, traditionally taught classes (see Fig. 2). Interestingly, although German teachers are often described as being reluctant to the idea of sharing [22], more than 60% of the survey's participants stated that they would happily share their materials, 30% did not answer the question but none said that they would explicitly not be willing to share (see Fig. 3).

Furthermore, we asked the teachers which would be the most important features for such an exercise repository. We provided three pre-defined answers, multiple selections were possible, and added an option for the teachers to add their own ideas. None of the teachers made use of this option.

The three features that we had asked them to rate were:

- A rating system for the exercises
- Well-defined access control (to make sure the pupils do not get hold of graded exercises)
- Collections of exercises that can be reused

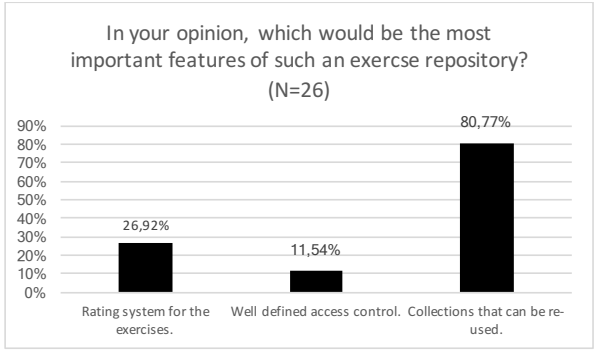

Fig. 4. Re-usable collections were the most popular feature

Surprisingly for us, a well-defined access control was not too much of a concern for the teachers that participated in our survey. This might be explained by the fact that more than 60% of those that have used the MOOC, did this in the context of extracurricular activities. The rating system for exercises received less attraction than expected as well. The most dominant feature was the option to create persistent collections that can be reused repeatedly (see Fig. 4).

D. Workshops and CoderDojos

Unfortunately, our sources in this area are rather anecdotal. We have not conducted a survey among this group yet. However, we have talked to some organizers of such workshops. Grading is not so much of an issue here, exercises will mostly serve as the basis for experiments or be inspiration for other implementations. Therefore, access control is of less importance, if not an obstacle. We have reason to expect that this group is a potential candidate to contribute additional

[42] This number has been estimated from the amount of learning groups in the course which have been labeled with the name of a certain school. Considered that the average class sizes in Germany are around 20-30 pupils and that some of the teachers used the course with even smaller groups in an extracurricular setting, this number probably was a little higher.

[43] A more detailed evaluation of these experiments and the complete survey will be discussed in a separate paper.

exercises. The needs of this group, therefore, have to be investigated more thoroughly in the future.

IV. CODEHARBOR

Figure 5 shows a schematic overview of the relations between exercise repository, auto-grader, and LMS or MOOC platform.

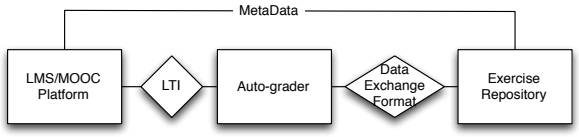

Fig. 5. Schematic view of relation between LMS, auto-grader and exercise repository

Building on this, Figure 6 shows a possible landscape of LMS systems, MOOC platforms, and auto-graders. MERLOT has been added here to demonstrate a possible integration with such a meta-system.

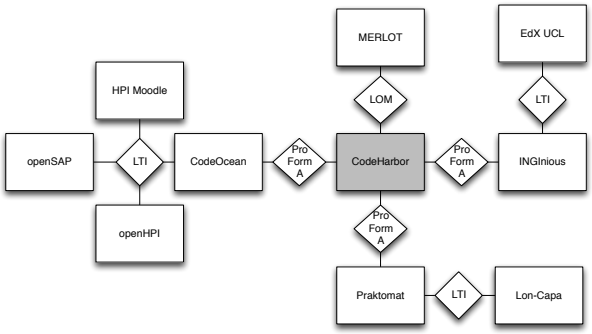

Fig. 6. Possible location of CodeHarbor in a landscape of LMS', MOOC platforms and auto-graders.

A demo version of *CodeHarbor* can be found here[44]. The following features are already implemented:

- create and modify exercises
- search for/browse exercises
- export exercises to auto-grader
- multiple exercise languages
- rate and comment exercises
- shopping cart and collections
- fork/clone exercises
- user groups
- exercise privacy

In the following we will discuss some of *CodeHarbor's* features in more detail.

A. Exercise Data Model

Figure 7 shows a reduced class diagram of the most relevant classes of *CodeHarbor* with a focus on the *Exercise* class.

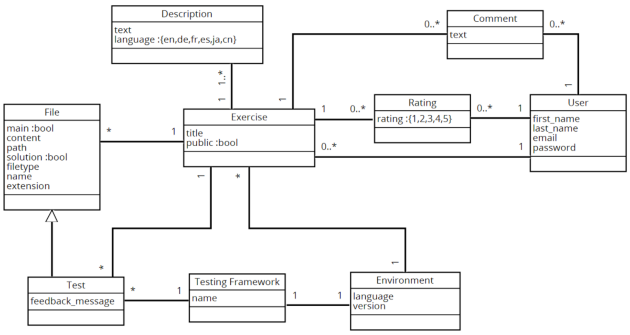

Fig. 7. Most relevant classes in CodeHarbor and their relations.

Exercises consist of one to many *files*. These files can either be skeleton implementations for the students to complete, predefined program elements, such as interfaces, or abstract classes to be implemented, or additional code snippets to take some workload off the participants' shoulders such as additional classes that already provide a piece of the functionality to be implemented. Further possible file types are *tests* for grading and user-defined tests. According to the requirements of the exercise, any of these files can be *read-only* or *hidden*. Finally, there might be additional files such as images, archives, and build directives, such as ANT files or make files to complete the setting.

Each auto-grader has a different data model for programming exercises, therefore a bidirectional, ideally lossless, exchange to the respective *CodeHarbor* classes is required. As our transfer format, we use the already mentioned ProFormA-XML standard.

The eCULT project already has developed an open source editor[45] to export auto-gradable programming exercises into ProFormA-XML, zips, or Lon-Capa compatible formats. As the tool is published under a Creative Commons 3.0 share alike license it can be integrated into *CodeHarbor* to provide simple integration with this project. To integrate *CodeHarbor* with our own auto-grader *CodeOcean*, we are currently developing a solution that allows a more direct interaction instead of downloading the data from the repository and uploading it to the auto-grader. A direct web-API based connection will allow more comfortable access and is our suggested solution for the other systems as well.

B. Rating and Commenting

CodeHarbor allows its users to rate exercises for expressing their satisfaction with the quality of an exercise and recommending it to others. It allows users to filter for content of proven quality. We are working with a 5-star rating system like the one that is used by Amazon. There is an ongoing discussion whether this system is too detailed or not. Other

[44] https://tools.openhpi.de/codeharbor

[45] https://github.com/ProFormA/formatEditor

popular models are a simple binary "thumbs up, thumbs down" as it is used on YouTube [21], or a traffic light approach—green: good to go, yellow: slight improvements required, red: do not use without major rework—, which is used e.g. by SAE's quiz repository (see Section 2). Users that mark a quiz with yellow or red must provide an additional verbose comment. In addition to the rating system, we encourage the users to discuss the perceived flaws and to provide suggestions for improvement. This allows to establish a communication between authors and consumers and to improve exercises in a collaborative effort.

C. Shopping Carts and Collections

CodeHarbor supports a "shopping cart" as a tool to collect an amount of exercises and export it to an auto-grader in one sweep. A more persistent way of combining multiple exercises are so called "collections". *Shopping carts* can be turned into *collections* and *collections* can be added to a *shopping cart* for export. *Collections* remain after they are added to a *shopping cart*, while the *shopping cart* will be emptied after the export. Users can have several *collections* to persist different packages for different purposes, e.g. a set of exercises to train the concept of polymorphism or a set of exercises to form a Python course.

D. Access Rights for Programming Exercises

Different stakeholders have different requirements towards the options to access and share the details of the exercises. While school teachers, university faculty or MOOC instructors require some restrictions so that the exercises can be used for exam purposes, e.g. workshop organizers will be interested in less restrictive settings so that they can share not only the exercises but also the solutions or at least the test cases.

In interviews, many instructors have expressed their concern to control the visibility and availability of the exercises. Particularly in exam situations, full control over the visibility of grading related files, such as unit tests, hints, or sample solutions is required. In other contexts, such as workshops or in self-directed learning settings, the (partial) seclusion of the material can hinder the participants in learning or reduce the usability of the material as an educational resource.

The initial approach to address this problem in *CodeHarbor* was to implement a simple role system with *user*, *teacher*, and *admin* roles. Only *teachers* and *admins* had the rights to see the solutions and tests of the exercises. Furthermore, only *teachers* and *admins* had the right to create and export exercises. The needs of stakeholders in less formal settings were not represented very well. Additionally, the solution also would have required a high administrative effort. Users would have had to apply for the *teacher* role, proving somehow that they are school teachers, university faculty or MOOC instructors.

We have, therefore, replaced the role system with a more flexible solution of *groups* and the concept of *private* and *public* exercises. *Public* exercises are fully accessible by any registered user. *Private* exercises will only reveal their title and description to users without more specific access rights. Hints, solutions, and tests, per default are only visible to the exercise's author. Every user, however, has the option to allow certain *groups* of users to access these exercises. Any user can create *groups* and add other users to that *group*. The author of a private exercise can then allow a given group full access to this exercise. Adding co-authors to an exercise can be done the same way. Thus, the whole process becomes more democratic and transparent, as not a central institution decides who can access which content item, but each participant can do this herself in a very fine-grained way.

E. Exercise Versions, Forks, and Clones

Versioning, forking, and cloning are key features for any kind of repository. *CodeHarbor* allows to define relationships between exercises, e.g. *Ex.2* translates *Ex.1* to another natural language or *Ex.3* ports *Ex.1* from *C++* to *Java*. *Ex.4* is an improved version of *Ex.2*, etc. Modelling these relationships enhances the probability to quickly find the right exercises for the given purpose.

It will also become important to provide the possibility to trace the revision history of an exercise, reset it to an older revision, compare different revisions, and to track and merge changes once the users start to work on the material in a collaborative fashion.

V. Future Work

The idea behind *CodeHarbor* has many supporters. However, whether *CodeHarbor* will be successful in the sense that the software will receive (long-term) development efforts and whether it will find users populating its exercise pool is unknown. Currently, there is little awareness among lecturers and teachers that such a system exists. To become a sustainable well-maintained open source project, the tool's documentation and feature set will have to be improved. The main issue, however, is to promote the tool among exercise providers and consumers. At conferences and similar events, we have received mostly positive feedback from practitioners, instructors and researchers. We are reaching out to similar initiatives, such as the X5gon[46] project to receive further visibility. Our next steps will be to fill the repository with our existing exercises and encourage others to add their exercises as well. Then, we will have to evaluate many of the features that have been implemented in terms of their everyday usefulness and usability.

VI. Conclusion

Programming exercises are time consuming to grade and create. Therefore, the concepts of automatic grading and sharing programming exercises can help teachers teach computer science and programming. A technical solution, such as *CodeHarbor* can only be a step in the right direction. Next to the availability, usability, and accessibility of the platform its success depends on how well it is accepted and filled with life by its users. The plain existence of projects such as *eCULT* or *MERLOT* proves that the interest in a platform for sharing programming exercises exists. The teachers we have interviewed during several workshops, expressed a desire to share teaching materials with other teachers. A survey among teachers, however, showed that their confidence in both, their time resources and their skills to create such exercises is rather

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

low. Other stakeholders such as Workshop providers and MOOC instructors are more likely to fill the repository with exercises.

ACKNOWLEDGMENT

Many thanks to Kirstin Heidler, Marcel Jankrift, Leo Selig, Adrian Steppat, and Theresa Zobel for their contribution to CodeHarbor. Further thanks to Oliver Rod from the eCULT project and all the teachers and school children who participated in the mentioned MOOCs and surveys.

REFERENCES

[1] T. Staubitz, H. Klement, J. Renz, R. Teusner, and C. Meinel, "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)*, Dec. 2015, pp. 23–30.

[2] T. Staubitz, H. Klement, R. Teusner, J. Renz, and C. Meinel, "Codeocean - a versatile platform for practical programming excercises in online environments," in *2016 IEEE Global Engineering Education Conference (EDUCON)*, Apr. 2016, pp. 314–323.

[3] T. Staubitz, R. Teusner, C. Meinel, and N. Prakash, "Cellular automata as an example for advanced beginners' level coding exercises in a mooc on test driven development," *International Journal of Engineering Pedagogy (iJEP)*, vol. 7, no. 2, pp. 125–141, 2017.

[4] J. Hollingsworth, "Automatic graders for programming classes," *Commun. ACM*, vol. 3, no. 10, pp. 528–529, Oct. 1960.

[5] D. Arnow and O. Barshay, "Webtoteach: An interactive focused programming exercise system," in *Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, vol. 1, Nov. 1999, 12A9/39–12A9/44 vol.1.

[6] C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas, "Automated assessment and experiences of teaching programming," *J. Educ. Resour. Comput.*, vol. 5, no. 3, Sep. 2005.

[7] A. Venables and L. Haywood, "Programming students need instant feedback!" In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20*, ser. ACE '03, Adelaide, Australia: Australian Computer Society, Inc., 2003, pp. 267–272.

[8] N. Truong, P. Roe, and P. Bancroft, "Automated feedback for "fill in the gap" programming exercises," in *Proceedings of the 7th Australasian Conference on Computing Education - Volume 42*, ser. ACE '05, Newcastle, New South Wales, Australia: Australian Computer Society, Inc., 2005, pp. 117–126.

[9] S. Almajali, Computer-based tool for assessing advanced computer programming skills, DOI: 10.1109/ICeLeTE.2012.6333420.

[10] Z. E. Balaa, "Developing an exercise management system for e-learning," in *2016 Sixth International Conference on Digital Information Processing and Communications (ICDIPC)*, Apr. 2016, pp. 93–96.

[11] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005.

[12] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, no. 3, Sep. 2005.

[13] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '10, Koli, Finland: ACM, 2010, pp. 86–93.

[14] J. C. Caiza and J. M. del Álamo Ramiro, "Programming assignments automatic grading: Review of tools and implementations," in *7th International Technology, Education and Development Conference (INTED2013)*, 2013, pp. 5691–5700.

[15] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. V. Roy, "Automatic grading of programming exercises in a mooc using the inginious platform," in *European MOOC Stakeholder Conference (EMOOCS)*, (Mons, Belgium), P.A.U. Education, 2016, pp. 86–91.

[16] N. Magrofuoco and A. Paquot, "Correctoz – recognizing common mistakes in the programming exercises of a computer science mooc," Master's thesis, Ecole Polytechnique de Louvain (EPL), Louvain, 2016. [Online]. Available: https://dial.uclouvain.be/memoire/ucl/fr/object/thesis:4587/datastream/PDF_01/view.

[17] J. Breitner, M. Hecker, and G. Snelting, *Der grader praktomat*, O. J. Bott, P. Fricke, U. Priss, and M. Striewe, Eds., https://pp.ipd.kit.edu/uploads/publikationen/praktomat16.pdf, Online; accessed 08-July-2016, 2016.

[18] ProFormA group, *An XML exchange format for (programming) tasks*, https://github.com/ProFormA/taskxml/blob/master/whitepaper.md, Online; accessed 08-July-2016, 2016.

[19] S. Strickroth, M. Striewe, O. Müller, U. Priss, S. Becker, O. Rod, R. Garmann, O. J. Bott, and N. Pinkwart, "Proforma: An XML-based exchange format for programming tasks," *Eleed*, vol. 11, no. 1, 2015, ISSN: 1860-7470. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0009-5-41389.

[20] M. Löwis, T. Staubitz, R. Teusner, J. Renz, C. Meinel, and S. Tannert, "Scaling youth development training in it using an xmooc platform," in *2015 IEEE Frontiers in Education Conference (FIE)*, Oct. 2015, pp. 1–9. DOI: 10.1109/FIE. 2015.7344145.

[21] S. Rajaraman, *Five stars dominate ratings*, https://youtube.googleblog.com/2009/09/five-stars-dominate-ratings.html, Online; accessed 08-July-2016, 2009.

[22] W. Bos, B. Eickelmann, J. Gerick, F. Goldhammer, H. Schaumburg, K. Schwippert, M. Senkbeil, R. Schulz-Zander, and H. Wendt, Eds., *Computer- und informations- bezogene Kompetenzen von Schülerinnen und Schülern in der 8. Jahrgangsstufe im internationalen Vergleich*. Münster; New York: Waxmann, 2014, ISBN: 978-3-8309-3131-7