Enhancing a Theory-Focused Course Through the Introduction of Automatically Assessed Programming Exercises – Lessons Learned

Marcus Soll [1][0000−0002−6845−9825], Louis Kobras[1][0000−0003−4855−2878], Melf Johannsen[2], and Chris Biemann[3]

[1] Universität Hamburg, Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
{2soll,4kobras}@informatik.uni-hamburg.de
https://www.uni-hamburg.de
[2] Universität Hamburg, Center for Optical Quantum Technologies, Luruper Chaussee 149, 22761 Hamburg, Germany
mjohanns@physnet.uni-hamburg.de
http://photon.physnet.uni-hamburg.de/en/zoq/
[3] Universität Hamburg, Language Technology Group, Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
biemann@informatik.uni-hamburg.de
https://www.inf.uni-hamburg.de/en/inst/ab/lt/home.html

Abstract. In this paper, we describe our lessons learned during the introduction of automatically assessed programming exercises to a Bachelor's level course on algorithms and data structures in the Winter semester 2019/2020, which is yearly taken by around 300 students. The course used to mostly focus on theoretical and formal aspects of selected algorithms and data structures. While still maintaining the primary focus of a theoretical computer science course, we introduce a secondary objective of enhancing programming competence by giving practical programming exercises based on select topics from the course. With these assignments, the students should improve their understanding of the theoretical aspects as well as their programming skills. The programming assignments were given in regular intervals during lecture period with a thematic alignment between assignments and lectures. To compensate for the new set of tasks, the workload of assignments on theoretical aspect was reduced. We describe the different experiences and lessons learned through the introduction and conduction of these exercises. A user study with 44 participants shows that the introduction was perceived well by the students, although improvements are still possible, especially in the area of feedback to the students.

Keywords: Automatic Assessment of Programming Exercises · CodeRunner · Lessons Learned · Moodle

1 Introduction

One of the key competences a student of computer science should possess at the end of his or her study should be the competence to write computer programs.

To support students in learning this important skills, many tools for automatically assessed programming exercises have been developed over the last years [2,14]. To help the students improve their programming skills, new automatically assessed programming exercises were introduced in the course *Algorithmen und Datenstrukturen* (algorithms and data structures) at the Universität Hamburg, taken by around 300 students in the Winter semester 2019/2020. In total, six blocks of exercises were created, in which the students had to participate. In this paper, we share our experiences and lessons learned when implementing these programming exercises in practice.

2 Related Work

There are many publications about the details of different tools for automatic assessment of programming tasks (e.g. see the reviews [1,2,7,14]). All of those reviews have a slightly different focus on the topic of automatically assessed programming exercises. While Caiza and Del Alamo [2] present a list of assessment tools, Ihantola et al. [7] discusses the technical features found in different assessment software. Both Ala-Mutka [1] and Souza et al. [14] include methodological aspects (e.g. testing for different quality measures like efficiency or test coverage in [1] or specialisation of tools like quizzes or contests in [14]) in their analysis.

In comparison, literature of the actual experience of introducing these tools into regular classes seems to be relatively sparse. However, there are publications describing the experiences of introducing automatically assessed (programming) tasks in regard to exercise design [3], plagiarism [3], resource usage [3], resubmission policies [12] (although they do not describe programming tasks, their tasks assess the understanding of algorithms on a concept level), and even redesigning of whole courses [8,11] including exams [13].

In our work, we use the CodeRunner tool for automatic assessment. The tool was developed by Lobb and Harlow [10]. Croft and England [4] described their experiences of introducing Coderunner, however, their publication focuses on the technical details and less on their actual experiences in deploying and using CodeRunner.

3 Context and Prior State

Currently, e-learning at Universität Hamburg is mainly used for the distribution of files (like lecture notes or exercise sheets) and for communication, to the best knowledge of the authors. There are only few cases where the potential of blended learning [5] is used. One example of such a project is the *CaTS* project [6], in which our department participated. In that project, online self assessment tests were developed for the class *Formale Grundlagen der Informatik I und II* (theoretical foundations of computer science, level 1 and level 2).

The goal of the Bachelor's level course *Algorithmen und Datenstrukturen* (algorithms and data structures) is to teach the students the principles of efficient

algorithms, both in theoretical and practical terms. Each year, around 300 students participate in the course. Prior to the introduction of the programming exercises, the main focus of the module was set on the theoretical and formal aspects of selected algorithms and data structures. Because programming skills were mainly taught in different modules, the practical aspects (sample applications, programming tasks) were not discussed. With this development, our goal is to blur the distinction of theoretical and practical courses, thereby allowing students to implement theoretical concepts from scratch.

In the course, the available e-learning platform Moodle[4] was previously used for sharing documents and for communication through a forum. In addition, students were able to check the progress on their course achievements, these, however, had to be input manually by the instructors. The introduction of online tests in the form of automatically assessed programming exercises is a novelty for the course. The implementation of these new exercises was done using the CodeRunner plugin [10] for Moodle.

4 Design and Deployment

By developing these programming exercises, we wanted to allow the students to deepen their understanding of the algorithms and data structures discussed in the lecture. This was done by letting the students implement different algorithms and sometimes let them use the algorithms to solve different tasks. One welcomed side effect was to improve the programming skills of the students through these exercises. To compensate for the additional work caused by the new exercises, the workload of assignments in the area of theory had to be reduced.

All exercises were created based on the topics of the lecture. We created the different tasks by first defining the requirements of the tasks. Based on these, we chose suitable algorithms and data structures for the programming tasks. Those were transformed into the actual task, the test cases and an example solution. The same procedure was used to create example programs for the actual lecture itself.

It was required for the students to pass the exercises in order to complete the course. As such, the students were externally motivated to complete the programming exercises. One example of such a task from the point of view of the students (including short explanations of all important user interface elements) can be seen in Fig. 1. In total, 10 tasks were created, which were combined into 6 blocks. The students could choose whether they wanted to use Java or Python, for each block the better result was counted. For each block, the students were given two weeks to complete the tasks. For each task, the students had 10 tries to develop a correct solution that passes all test cases, however, they could also test their solution on a smaller set of pre-test cases. The test cases were composed of corner cases (e.g. empty input, maximum input value), normal cases and random tests. The random tests prevented the students from hard-coding the test results

[4] `https://moodle.org/`

into their programs. All test cases were restricted in execution time and memory usage, however, the provided limitations were more than enough to pass all test cases even with inefficient solutions. Feedback to the students was only send through the result of the test cases, since manual feedback would have put a lot of additional work on the instructors and thus was not feasible. In addition, a sample solution was provided for each task. The average length of the provided sample solution, including source code comments, amounted to 24.5 and 19.7 lines of code for Java and Python, respectively, with both peaking at 41. All tasks were perceived as easy by all instructors.

Fig. 1. Example of programming the least common denominator (LCD) in Java. Realised using the CodeRunner plugin [10] for Moodle. All important user interface elements are explained.

To facilitate communication with the students, multiple ways of communication were offered, both for announcements as well as for questions. These include a mailing list, Moodle-based communication (a forum as well as announcements) and special tutorials, which could be joined by students at will.

5 Lessons Learned

While the technical setup did not pose notable issues and overall the students were able to use the system and achieve their learning goals, we encountered some issues. These are described below.

5.1 Heterogeneity of Student Knowledge

Due to the curricular structure of the Universität Hamburg as well as possible extracurricular activity, the knowledge of the students when starting the course is highly diverse. Firstly, the students are enrolled in different study programs. Because of this, there is not one single programming language everyone is trained in. As a consequence, we had to develop the tasks in different programming languages (Java and Python), which significantly increased the efforts, as this does not only imply creating the tasks twice, but also requires modelling this on the side of automatic score reporting. In addition, the students greatly diverged in programming skill levels. While some perceived the tasks as quite difficult, there was also a smaller group who found the tasks to be extremely easy.

5.2 Abstraction of CodeRunner

CodeRunner adds an extra layer of abstraction between the student and the system on which the code is actually run. This extra layer caused many problems for the students.

Once, it is not directly visible what exactly the underlying system is doing, and especially what effect the different user interface elements have on the system. To reduce difficulties, we used different counter-measures: a live demonstration at the beginning of the semester, as well as a user manual that students could consult on any questions. Still, the students had problems with the user interface in the first weeks.

In addition, errors in students' solutions are not easy to debug. Although any compiler errors or failed test cases are shown, the execution of the source code could not directly be analysed by standard tools like a debugger. It has been proven helpful to provide special source code files, which allowed the students to develop solutions on their own computer by emulating the behaviour of the system.

Finally, students were quick to blame the system for any error instead of searching them in their own solution. For example, one student has blamed the system for not allowing enough execution time for his solution although he programmed an infinite loop in his solution. Because of this and similar cases, we had a high demand of support (see below).

5.3 Students' Creativity

We could observe the problems of many students to apply the knowledge they gained in the lecture to the programming exercises. As a result, many students

tried to use their self-developed, *creative*, algorithms instead of using the algorithms presented during the lecture. This was also the case for tasks like 'Implement algorithm X'. Often, these algorithms had many problems in different cases (especially in corner cases), which causes malfunction in both normal execution and our test cases. Because of this, it is proven to be especially important to cover each possible cause of errors with its own respective test case, some of which were hard to anticipate. This way, students could analyse each failed test case individually and easily find their errors. Whenever there was a cause of error we did not anticipate (and therefore did not have a test case) we could observe the students having more problems.

In addition, there were cases where a student had problems with the random test cases while at the same time they passed all other test cases. This shows two things: There might be some hidden problems in the student's solution, and there were some test cases we had missing. While we plan to improve our test cases by collecting these issues, it is not always possible to avoid such problems since tasks might have to be changed each year in order to ensure unseenness.

5.4 High Demand of Support

Although there were no major problems with the programming exercises and the systems ran stable, there was still a high demand of support from the students in the form of questions and support requests whenever they were not able to solve an issue on their own. This includes for example questions concerning the interpretation of the given task, technical difficulties, issues with their solution, and organizational questions. Since most of the aforementioned problems and resolutions were very specific to the students' solutions, the support was highly individual and therefore caused high time and effort demands.

6 Evaluation

To evaluate the acceptance of the programming exercises by the students, we conducted a user study. The study is carried out following Kreidl [9], who defined multiple variables (grouped into 4 categories) that contribute to the acceptance of e-learning systems by students. For the evaluation, we used a modified version of his questionnaire (which was originally in German, and we conducted the user study in German). The scale used in the survey is inverted compared to the original publication by Kreidl. The variables *voluntariness and incentives* (the participation was mandatory to pass the course) and *exam preparation* (the programming exercises were not relevant for the exam) were not tested for the given reasons.

Out of the 300 students that partook in the course, 44 students additionaly participated in the user study on a voluntary basis. As can be seen in Tab. 1, the programming exercises were accepted well by the students in general (all values are around 2). However, improvements can be made especially in the area of feedback to the students (variables *availability of tasks and learning causes* and

Table 1. Average of measured variables by the user study of the acceptance of the programming exercises. The scale is 1 (worst) to 5 (best), with 3 being the middle. (n=44)

Category	Variable	Average	σ
didactics	understandable content	4.0	1.0
	availability of tasks and learning causes	3.4	1.0
	feedback to the students	3.6	1.2
	communication and cooperation	4.0	0.9
	overall quality of system	4.1	0.9
organisation	supporting measures	4.1	0.9
	technical realisation	4.3	0.8
incentive	usage of platform	3.7	1.2
	motivation for learning	3.8	1.1
	satisfaction	3.9	1.0
usage	intensity of usage	2.7	1.3

feedback to the students). The value *intensity of usage* is low in comparison to the others, however, this is expected, since it was intended that the students do the programming exercises only a single time.

We also evaluated the amount and difficulty of the tasks. The students could rate the difficulty on a scale of 1 (too easy) to 5 (too difficult), with 3 equal to being adequate. The amount of tasks was evaluated on a similar scale, from 1 (too many) to 5 (too few), with 3 equal to being a good number of tasks. The tasks received a difficulty of 3.0 ($\sigma = 0.7$) whereas the amount received a rating of 2.7 ($\sigma = 0.7$). This shows that our tasks where perceived as having the right number and difficulty for the course.

7 Conclusion

In this paper, we described our experiences of introducing automatically assessed programming exercises in a Bachelor's level course focusing on algorithms and data structures in computer science with around 300 yearly participants. The course is mostly focused on theoretical and formal aspects. Overall, the introduction of the programming exercises was successful, although we experienced some difficulties in the area of the mixed prior knowledge of participants, students' creativity, the extra abstraction layer of CodeRunner, and a high demand of support. A user study shows that the programming exercises were accepted by the students, although there is still room for improvement especially in the area of feedback to the students concerning the specific issues of their solutions.

Currently, it is planned to continue the programming exercises in next year's course. Improvements are especially planned for including better feedback. Since manual feedback by instructors is not feasible for the course, it is planned to improve feedback by both improving the test cases and the feedback included in the test cases (e.g. purpose of the test case and common mistakes).

Acknowledgements. This research was supported by MINTFIT Hamburg. MINTFIT Hamburg is a joint project of Hamburg University of Applied Sciences (HAW), HafenCity University Hamburg (HCU), Hamburg University of Technology (TUHH), University Medical Center Hamburg-Eppendorf (UKE) as well as Universität Hamburg (UHH) and is funded by the Hamburg Authority for Science, Research and Gender Equality.

References

1. Ala-Mutka, K.M.: A Survey of Automated Assessment Approaches for Programming Assignments. Computer Science Education **15**(2), 83–102 (2005). https://doi.org/10.1080/08993400500150747
2. Caiza, J.C., Del Alamo, J.M.: Programming assignments automatic grading: review of tools and implementations. In: 7th International Technology, Education and Development Conference (INTED2013). pp. 5691–5700 (2013)
3. Cheang, B., Kurnia, A., Lim, A., Oon, W.C.: On automated grading of programming assignments in an academic institution. Computers & Education **41**(2), 121–131 (2003). https://doi.org/10.1016/S0360-1315(03)00030-7
4. Croft, D., England, M.: Computing with CodeRunner at Coventry University: Automated Summative Assessment of Python and C++ Code. In: Proceedings of the 4th Conference on Computing Education Practice 2020. CEP 2020 (2020). https://doi.org/10.1145/3372356.3372357
5. Friesen, N.: Report: Defining Blended Learning. Received from `https://www.normfriesen.info/papers/Defining_Blended_Learning_NF.pdf` on Apr 3rd 2020 (2012)
6. Goethe-Universität - Computerbasiertes adaptives Testen im Studium. https://www.studiumdigitale.uni-frankfurt.de/66776844/CaTS, last accessed: 26.02.2020
7. Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O.: Review of Recent Systems for Automatic Assessment of Programming Assignments. In: Proceedings of the 10th Koli Calling International Conference on Computing Education Research. pp. 86–93. Koli Calling '10 (2010). https://doi.org/10.1145/1930464.1930480
8. Kaila, E., Kurvinen, E., Lokkila, E., Laakso, M.J.: Redesigning an Object-Oriented Programming Course. ACM Transactions on Computing Education **16**(4) (2016). https://doi.org/10.1145/2906362
9. Kreidl, C.: Akzeptanz und Nutzung von E-Learning-Elementen an Hochschulen. Gründe für die Einführung und Kriterien der Anwendung von E-Learning. Waxmann (2011), `http://nbn-resolving.org/urn:nbn:de:0111-opus-82880`
10. Lobb, R., Harlow, J.: Coderunner: A Tool for Assessing Computer Programming Skills. ACM Inroads **7**(1), 47–51 (2016). https://doi.org/10.1145/2810041
11. Lokkila, E., Kaila, E., Karavirta, V., Salakoski, T., Laakso, M.: Redesigning Introductory Computer Science Courses to Use Tutorial-Based Learning. In: EDULEARN16 Proceedings. pp. 8415–8420. 8th International Conference on Education and New Learning Technologies (2016). https://doi.org/10.21125/edulearn.2016.0837
12. Malmi, L., Karavirta, V., Korhonen, A., Nikander, J.: Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmission Policies. Journal on Educational Resources in Computing **5**(3), 7:1–7:23 (2005). https://doi.org/10.1145/1163405.1163412

13. Rajala, T., Kaila, E., Lindén, R., Kurvinen, E., Lokkila, E., Laakso, M.J., Salakoski, T.: Automatically Assessed Electronic Exams in Programming Courses. In: Proceedings of the Australasian Computer Science Week Multiconference. pp. 11:1–11:8. ACSW '16 (2016). https://doi.org/10.1145/2843043.2843062
14. Souza, D.M., Felizardo, K.R., Barbosa, E.F.: A Systematic Literature Review of Assessment Tools for Programming Assignments. In: 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET). pp. 147–156 (2016). https://doi.org/10.1109/CSEET.2016.48