Teaching during the Covid-19 Pandemic — Online Programming Education

Sandro Speth[1], Niklas Krieger[1], Georg Reißner[1], Steffen Becker[1]

Abstract: Teaching programming skills is an essential part of the first-semester computer science curriculum. During the Covid-19, it became particularly challenging as, in particular, the essential exercise lessons had to be implemented virtually. Before the pandemic, we implemented didactic concepts like objects-first and gamification with the hamster simulator. They had to be transferred to the virtual setting as good as possible. In this paper, we report on the implementation of our virtual course programming and software development. In particular, we report on the lessons we learnt from our course implementation, which, overall, was relatively successful.

Keywords: Digital Lectures; Programming Education; Intelligent and Automated Tutoring System

1 Introduction

In every computer science curriculum, students have to gain programming skills in the first semester. At the University of Stuttgart, we usually teach programming in a course with 180 minutes lecture time and 90 minutes exercise time per week. On average, about 750 students participate in the course. The lecture takes place in a large lecture hall, and the exercises are split into approximately 35 exercise groups taught by paid students as tutors. However, when the Covid-19 pandemic started, we had to move all activities in a virtual setting.

Virtualising such a large and complex setting is a challenging task. We faced several issues we had to address. How to deliver the lecture to the students while still keeping interactive elements such as live discussion breaks during the lecture and allowing for questions? How to compensate for the lack of peer programming in the exercise? How to offer the same amount of tutor time while facing the fact that every activity takes longer in a virtual setting than a physical meeting? How to replace the mock exam the students are required to pass before they are allowed to participate in the actual exam? How to deal with students who had to use our computer lab because of a lack of dedicated hardware? Moreover, how to keep in contact with the students and monitor their learning progress?

In our course, we already had implemented a couple of modern didactic methods. We taught objects first, as characterised by B. Meyer in his book “Touch of Class” [Me09]. We used

[1] University of Stuttgart, Institute of Software Engineering, Universitätsstraße 38, 70569 Stuttgart, Germany
{sandro.speth,niklas.krieger,georg.reissner,steffen.becker}@iste.uni-stuttgart.de

doi:10.18420/SEUH2022_09

our variant of the hamster simulator[2], a mini programming world to implement the objects first approach in Java, including BlueJ at the beginning of the course.

In this paper, we report how we faced the characterised challenges. We outline in detail our implemented measures to countermeasure each issue. The implemented actions include the use of (1) a dedicated interactive and video-based e-Learning book for the lecture, (2) virtual exercises in Microsoft Teams with breakout rooms including the use of CodeTogether, (3) the use of the ArTEMiS exercise management system [KS18] which we extended towards becoming an intelligent tutoring system, and (4) a YouTube live stream to keep the connection to the students alive. We described the concrete hardware and software setup with costs in a GitHub wiki[3]. Our lessons learned are many-fold. Several implemented countermeasures worked surprising well. Others did not succeed as planned. In the end, we deem that many lessons learnt can be reused for hybrid or blended learning setups.

The remainder of the paper details our implemented countermeasures (cf. Section 2). In Section 3 we outline our main lessons learnt before we conclude the paper in Section 4.

2 Transformations

Transformation I: Lecture Due to the Covid-19 pandemic and the resulting need for students to attend lectures from home, traditional lectures had to be converted into digital formats. Consequently, many lecturers decided to conduct their lectures virtually via conferencing tools such as Zoom or Microsoft Teams. However, it is challenging for many students to attend an entire lecture concentrated on a video call on the screen without losing attention and missing essential details. Therefore, other lecturers tended to pre-record their lectures and make the videos available to students for self-study from home. Nevertheless, the quality of the recordings can vary greatly. Additionally, students have difficulties finding a specific topic in a long video. For this reason, we have broken down the content of our lectures into short videos of around 5–10 minutes in length. To ensure that the amount of videos is available to the students in a structured way and that they can quickly find the content they are looking for, we use the *learning module* plugin of the ILIAS e-learning platform. In an ILIAS learning module, it is possible to create chapters and pages similar to a book. A lecturer can fill the pages with content such as text, images or videos. Furthermore, the learning module shows the progress through colour coding, making it easier for the student to identify which pages have not yet been viewed. Moreover, there is a page at the end of each chapter for checking the learning status with quizzes which, if passed, unlocks the next chapter and can be repeated as often as desired. We built a setup with a green screen, 4k video camera, external screen, podcast microphone, and various lighting in a soundproof basement for the video recording. The videos were recorded with Open Broadcast System (OBS) and then post-edited with Adobe Premiere and Adobe After Effects to improve the image and sound quality.

[2] `https://github.com/SQAHamster/plain-java-hamster`

[3] `https://github.com/spethso/Remote-Teaching-Setup/wiki`

Transformation II: Central Lecture Exercise In addition to the lecture, students could voluntarily attend the weekly lecture exercise before the pandemic, in which we explained selected topics from the lecture in detail. Another significant part of the lecture exercise consisted of shorter tasks to discuss the respective topics and the opportunity to ask questions centrally. For the content parts, we created videos and an ILIAS learning module similar to the lecture. However, the question and discussion part, in particular, depends on the active participation of the students, which is why an asynchronous alternative was needed. Since the overall number of students in a lecture exercise usually exceeds the maximum number of participants of Microsoft Teams, Cisco Webex, and Zoom Calls, we decided to stream the synchronous part of the lecture exercise via YouTube for one lecture hour per week. The streaming was done with OBS from the video room, and the students could ask anonymous questions via chat which were discussed on the iPad in a digital whiteboard by us. Moreover, several tutors managed the active stream of questions in the chat and answered some pending questions. Since YouTube automatically records live streams, students could also watch them at any time later.

Transformation III: Introduction Lecture A significant disadvantage of digital teaching is that first-year students can no longer see their lecturers in person and get to know them inside the auditorium. However, this is important for most students so that the lecturer no longer remains the unknown magical being behind the lessons. Furthermore, students usually ask many organisational questions in the first lecture. In a purely asynchronous lecture setting, these can only be asked via a forum or email. To overcome both challenges, we streamed two lectures live via YouTube at the beginning of the semester. The first live stream served to explain the organisational framework and clarify questions. In the second live stream, "Meet-your-Prof", students were able to ask us various questions, including funny ones, to get to know us better. The event thus served as an ice breaker so that students could experience university life in the online setting during the pandemic and motivate them to ask questions.

Transformation IV: Exercises The most important part of learning programming is practice. However, traditional on-site group exercises supervised by a tutor are not possible in a digital setting. For this reason, we conducted our synchronous live exercise sessions virtually via Microsoft Teams, including breakout groups with two students per team, through which the tutor could iterate. We decided on Teams as it is free for universities and schools. Both students in a breakout group work on the programming tasks together through pair programming. Since participants could not switch between groups independently in the breakout feature of Microsoft Teams, which we required for various discussion activities, we decided to simulate breakout rooms artificially in Microsoft Teams via a team and several channels. Additionally, as the students do not sit together in front of a computer as they would in a face-to-face exercise, they could use the collaboration tool CodeTogether[4]. Using

[4] `https://www.codetogether.com/`

CodeTogether, one team member hosted an IDE session, and the other member joined via a shared link either in their IDE using a plugin or the browser. CodeTogether allowed both students to work on the same project simultaneously. One student transferred his IDE as a screen share to enable the tutor to identify functional and stylistic errors, thus, enabling the tutor to view the current work without interference. Furthermore, the students could check their tasks on their own using ArTEMiS as a virtual tutor, which is explained in Sect. 2.
To successfully be allowed to participate in the course's final exam, the students were required to attend and pass the exercises and succeed in four quizzes on ILIAS as a replacement for a mock exam written before the end of the semester to prepare students for the actual exam. They were released regularly over the semester and checked that all students understood the topics from the lectures. The students had a window of two hours for each of the quizzes on the day they could work on it. During these two hours, they could start the quiz exactly once, from which point they had 30 minutes to complete and submit it. To pass a quiz, the students had to achieve at least 50% of the available points, which between ~87–100% of those who attempted the quiz did over the four quizzes.
To also provide the computer science teaching qualification students with valuable reflective pedagogical practice [BBF20], we implemented the described extra tasks and separate tutorial groups for all pre-service teacher students in the course.

Transformation V: ArTEMiS as Virtual Tutor Live tutor feedback is one of the critical aspects of every exercise. Despite all efforts spent on online tutoring, there was less interaction between students and tutors than offline exercises. Therefore, we used ArTEMiS[5], an automatic assessment management system developed at the Technical University Munich [KS18], to give additional feedback on exercises. Our setup consists of the main system, a GitLab plugin to store the student repositories, and a Jenkins plugin to automatically evaluate the students' code. After the students registered an account and joined our course on our ArTEMiS website, they could see the list of already existing exercise sheets. When starting an exercise sheet, the system creates a git repository, which the student can clone to work on the exercises. After each push to the remote repository, ArTEMiS automatically executes tests, and the student can see how many tests succeeded for each exercise. If tests failed, we also provided feedback on the cause of the failure. In order to provide an additional incentive to use ArTEMiS, we decided to provide tests both for non-graded and some graded exercises. This allows students to get automatic feedback on their exercises before submitting the final solution to ILIAS without a tutor manually reviewing the code. However, we made sure that the students were not able to see the source of the tests by using the Ares library[6], a JUnit extension. This library also prevented the students from executing several types of malicious code on our test servers by i.a., preventing network or file system access and adding hard time limits to test cases. Additionally, the test results helped us identify the student's open fields of learning.

[5] `https://github.com/ls1intum/ArTEMiS`
[6] `https://github.com/ls1intum/Ares`

Transformation VI: Online IDE In regular (presence) semesters, students, who did not own powerful enough computing hardware or any computing hardware which is compatible with a JDK or one of the used IDEs, had the opportunity to participate in those exercises held in a computer pool at the university and use the provided hardware. As this was not a possibility when the university's buildings were not accessible to students, an alternative way of providing compatibility for lower-spec devices was needed. The original plan to use "Microsoft Visual Studio Codespaces", a VS Code instance hosted in an Azure Cloud usable with any Microsoft Account, was not realizable due to Microsoft renaming the product to "GitHub Codespaces" and moving it into an (at that point in time) closed Beta. As a replacement, we created a similar application called "SQA CodeOnline" which we now host for free for all course students. It is based on the open-source product "code-server"[7] which hosts a VS Code Server and provides access to the GUI through the browser. After a student logs in on the welcome page, an individual docker container, based on a custom image in which code-server and all needed tools are installed, is created and assigned a virtual filesystem of 150MB for the student's files. Within those constraints, the students can use their containers for regular development. As the hamster simulator which we use in the course initially relies on a JavaFX GUI and the container the students are constrained to do not provide a window server, an alternative user interface was added. An extension for VS Code, which is pre-installed in the docker image, connects to a local HTTP server which the hamster simulator automatically opens if it detects that it is running on SQA CodeOnline [8]. The GUI in the extension is mainly a web view displaying a browser version of the GUI (including all controls). This gives the student the full ability to work on the exercises from any device (even mobile ones), just like others who work on VS Code locally. They can install extensions etc., to customize their editing experience. Because it stores the files on a remote server, they are synchronized between all used devices.

3 Lessons Learned

As demonstrated in the sections above, it is very well possible to transform even a more extensive lecture course into a purely online course without losing too much of the value for the students. Some formats even have a better reception by students than the traditional presence course. For one, the short videos, structured in the ILIAS e-Learning Module, were rated more positively by the students. Even contrary to many experiences in other online courses, the interactivity was (for some of the formats) even higher than what was experienced in presence. The most interaction occurred on the YouTube streams of the central lecture exercises where students were asking questions and interacting more freely than what they would in a lecture hall in their presence. Less interaction and challenges how to help the students arose, however, in the tutor exercises. Learning programming needs social interaction. However, students often join the meetings without saying or writing much. Of course, all the added tools and support cannot replace that and engage all those

[7] `https://github.com/cdr/code-server`

[8] `https://github.com/SQAHamster/VSCodeHamster-Mirror`

involved in teaching. While ArTEMiS provided valuable feedback, several students had difficulties using git. That is why some students, esp. those needing feedback the most, decided not to use the system. However, this effect can be mitigated by providing a better IDE integration to make using ArTEMiS easier. Therefore, we are integrating Orion[9], an IntelliJ IDEA extension for ArTEMiS, into the lecture for winter semester 2021/2022.

4 Conclusion and Outlook

In this paper, we present the implementation of our virtual introductory course to programming. We outline how we tackle the various challenges of the virtual setting. Finally, we report the lessons we have learnt during our journey.

The lessons we learned can help course instructors implement hybrid programming courses in a blended learning approach. In doing so, instructors can combine the best of the virtual teaching environment with the physical setup.

In the future, we will try to reorganize our programming course and move it closer towards a sound blended learning setup. Furthermore, we work on the implementation of our intelligent tutoring system. This intelligent tutoring system (named IT-REX) should better track the gathered skills of the students, adapt the exercised to the progress and give better hints to typical programming mistakes.

References

[BBF20] Becker, S.; Bescherer, C.; Fest, A.: Reflective Pedagogical Practice on and in Introduction to Programming and Software Engineering. In: Tagungsband des 17. Workshops “Software Engineering im Unterricht der Hochschulen” 2020, Innsbruck, Österreich, 26. - 27.02.2020. Vol. 2531, CEUR-WS.org, pp. 7–10, 2020.

[KS18] Krusche, S.; Seitz, A.: ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. SIGCSE ’18, Association for Computing Machinery, pp. 284–289, 2018.

[Me09] Meyer, B.: Touch of Class. Learning to Program Well with Objects and Contracts 51/, 2009, URL: `https://link.springer.com/book/10.1007/978-3-540-92145-5`.

[9] `https://github.com/ls1intum/Orion`