Computational Thinking as Springboard for Learning Object-Oriented Programming in an Interactive MOOC

Johannes Krugel
School of Education
Technical University of Munich
Munich, Germany
johannes.krugel@tum.de

Peter Hubwieser
School of Education
Technical University of Munich
Munich, Germany
peter.hubwieser@tum.de

***Abstract*—The prerequisite knowledge regarding Computer Science (CS) varies strongly among freshmen at university and it seems advisable to compensate for these differences before the first lecture starts. Massive open online courses (MOOCs) might represent a possible solution. We therefore designed and developed a MOOC (called "LOOP: Learning Object-Oriented Programming") which provides a gentle introduction to computational thinking and object-oriented concepts before the programming part. In addition to the common quizzes, we developed various we-based interactive exercises to enable the learners to experiment and interact directly with the presented concepts. Furthermore, we implemented programming exercises with constructive feedback for the learners using a web-based integrated development environment and additionally an automatic grading system. The target group of the course are prospective students of science or engineering that are due to attend CS lessons in their first terms. The course was conducted as a prototype with a limited number of participants. In a concluding survey, the participants submitted textual feedback on the course; some of them proposed specific improvements for the employed interactive exercises. Yet, the overall feedback was encouragingly positive. In this paper, we describe the design and the development of the course, as well as our initial results.**

Keywords*—Computer science education; Courseware; Education courses; Educational technology; Electronic learning; Self-study courses; STEM

I. Introduction

Due to the absence of systematic Computer Science (CS) education in many countries, the prerequisite knowledge of freshmen at universities varies considerably, depending on their prior engagement in CS, the country or state where they graduated from school, or on the school branch they have attended [1]. We performed a survey in a CS1 lecture at our university in October 2015 using a limited version of the MoKoM instrument [2] to test the competencies in object-oriented programming (as proposed, e.g., by [3]). The results of the 874 participants revealed huge differences in the students' prior knowledge. It is very difficult for CS lecturers of the first terms to handle this diversity, in particular regarding programming abilities.

Thus, it seems advisable to compensate for or reduce these differences in knowledge before the first lectures starts. As the students cannot be expected to be present at university at this time, MOOCs (massive open online courses) seem to represent a potential solution since the learners can take the course independently of location and time. At the same time, as learning to program is a substantial cognitive challenge [4], such MOOCs run in the risk of overstraining the students, frustrating them prior to the beginning of their studies.

To avoid this danger, we designed a MOOC (called "LOOP: Learning Object-Oriented Programming") that starts which provides a gentle introduction to computational thinking [5] and object-oriented concepts before the programming part to avoid excessive cognitive load (following the concept "objects strictly first" [6]). In addition to the common quizzes giving direct feedback to the learners, we included "really" interactive tasks for every learning step. Special care was devoted to the selection and development of those interactive exercises to enable the learners to experiment and interact directly with the presented concepts. It can be a major obstacle for potential participants having to install special software [7, 8], which is especially problematic in an online setting without a teacher who could help in person. We therefore decided to use only purely web-based tools. The target group of the course are prospective students of science or engineering that are due to attend CS lessons in their first terms.

In this paper, we describe related courses (Section II), the design and development of our course (Section III), its conduction (Section IV), our initial results and experiences (Section V), and conclude with a discussion (Section VI).

II. Related Courses

There are many online courses for learning the basics of computer science. In the following, we provide a short overview of some introductory MOOCs and SPOCs (small private online courses) that explicitly cover computational thinking or object-oriented programming (OOP) and that were recently published in the scientific literature.

Liyanagunawardena et al. [8] describe the experiences with a MOOC for the introduction to programming where the learners have the opportunity to build an Android game. They report on a good community experience, but, they note that one barrier for the learners was to install the development software.

Piccioni et al. [9] describe a SPOC used to complement an existing course for the introduction to programming. As a gamification element, badges are awarded to learners.

Falkner et al. [7] developed a MOOC in which the participants learn programming by producing digital artwork.

Alario et al. [10] developed a MOOC with interactive exercises using, among others, the software Greenfoot [11].

III. Course Design

LOOP was developed on the MOOC platform edX Edge[1] in German. In this section, we describe the contents and didactical considerations of the course design, followed by a discussion of the course elements.

A. Course content

In principle, LOOP has a similar content structure as the CS course in grade 10 of Bavarian Gymnasiums (in a reduced form), which is described in detail in [12]. It follows the "objects first" approach that was introduced as a reaction to problems students faced in writing their first object-oriented programs [13].

Computational thinking (CT) as introduced by Wing [5] is a universal personal ability that can be used in many disciplines. Since the target group of our course comes from various different fields of study, we incorporated CT as integral part of the course. CT is on the one hand intended to facilitate learning programming and on the other hand a sustainable competency that can be used also outside of our course.

As pointed out in [4], there is a fundamental didactical dilemma in teaching OOP: On the one hand, modern teaching approaches postulate to teach in a "real life" context [14], i.e., to pose authentic problems to the students. Therefore, it seems advisable to start with interesting, sufficiently complex tasks that convince the students that the concepts they have to learn are helpful in their professional life. However, if we start with such problems, we might ask too much from the students, because they will have to learn an enormous number of new, partly very difficult concepts at once [4].

Following a "strictly objects first" approach [6], we solved this problem by distributing the learning objectives over the parts of the course that precede the "serious" programming part and thereby avoiding confronting the learners with too many unknown concepts when they have to write their first program. Basically, we suggest to the students to look at an object as a state machine [4]. In order to realize this in a learner-oriented way, the students need to be able to understand a simulation program of a typical state machine, e. g., a traffic light system.

Before introducing a textual programming language, we introduce the structure of algorithms and use a block-based programming language to reduce the cognitive load [11]. We also try to reduce the cognitive load when actually introducing Java and, e. g., initially hide "advanced" aspects (like access modifiers) to let the learners focus on the essential parts of the class definitions. LOOP consists of the following five chapters:

1. Object-oriented modeling
2. Algorithms
3. Object-oriented programming
4. Implementing algorithms and arrays
5. Associations and references

B. Videos

All topics of the course are presented in short videos with an average length of 5 minutes. The videos were produced based on the suggestions of [15] and similar to the suggestions by [16] published shortly after our recording.

Each of the 24 videos begins with a short advance organizer to help the learners focus on the relevant aspects. This is augmented with the talking head of the respective instructor (using chroma key compositing) facilitating the learners to establish a personal and emotional connection [16]. For recording, we used the software *Blackmagic Media Express*.

For the actual content of the videos, we decided to use a combination of slides and tablet drawing. The background of the video consists of presentation slides and the instructor uses a tablet to draw and develop additional aspects or to highlight important part of the slides ("Khan-style"). This turned out to yield quite engaging videos with a reasonable effort for preparing and recording. As software tools we used *Camtasia Recorder* and *Camtasia Studio*. All slides are provided for download and we additionally added transcripts for the videos. By such video, audio, and textual representations, several senses are addressed simultaneously, making the content accessible to learners with different learning preferences or impairments.

C. Quizzes

After each video, the course contains quizzes as formative assessment. The main purpose is to provide the learners with direct and instant feedback on the learning progress. The quizzes use the standard assessment types offered by the MOOC platform, e.g., single-/multiple-choice questions, drop-down lists, drag-and-drop problems or text input problems. Depending on the answer, the learner gets a positive feedback or for example hints which previous parts of the course to repeat in more detail.

D. Interactive exercises

The videos introduce new concepts to the learners and the quizzes test the progress, which is, however, in general not sufficient to acquire practical competencies [10]. Following a rather constructivist approach, we let the learners experiment and interact with the concepts directly. Considering that, we include interactive exercises or programming task for all learning steps throughout the course. There are already many web-based tools for fostering computational thinking and learning OOP concepts available on the web. We selected the in our view most suitable tools supporting the intended learning goals and integrated them into our course. Where necessary we adapted or extended them to meet our needs.

Following the concept *objects strictly first* and to acquaint the learners with the notion of objects, we included the web-based vector graphic drawing tool *SVG-edit* [2]. The learners are given the task of drawing a simple graphic using rectangles, circles and lines which implicitly also already introduces the idea of classes. The learners are then asked to publish their drawing in the discussion forum of the course and to introduce themselves to the community.

To let the learners experience that objects have a state, that the state can change, and that this is usually achieved by method calls we developed a new interactive exercise. The learners can draw a picture by using simple commands in a restricted pseudo programming language, which only allows

[1] https://edge.edx.org (date of last access for all URLs: 2017-02-02)

[2] https://github.com/SVG-Edit

[3] https://trinket.io

the creation of graphical objects and method calls. We therefore combined the tool *trinket* [3] (providing an online code editor connected to a canvas) and the JavaScript library *SVG.JS* [4] (providing an interface for drawing objects). We adapted and extended this such that the learners can inspect the drawn objects by showing the UML object diagram when hovering over an object. We prompted the learners to draw an animal or a cartoon figure and to share it in the discussion forum.

Enabling the learners to interact with and to visualize simple algorithms we integrated the geometric JavaScript framework *CindyJS* [5] [17]. As example, we use the Euclidian algorithm for calculating the greatest common divisor of two numbers. The learner can modify the input by moving a point in the plane and observe at the same time the steps of the algorithm.

To facilitate the understanding for the structure of algorithms we included a gamification element using block-based programming. We integrated a series of maze riddles from *Blockly-Games* [6], which can be solved by combining move-operations with structural elements like loops and conditional statements.

Syntax can be a major obstacle when learning to program [11]. We therefore tried to make the first steps easier by providing a gentle introduction. Before the learners start to implement their first Java class from scratch, we let them experience the connection between the UML class diagram and the corresponding Java implementation using the web-based tool *UmpleOnline* [7] [18]. This tool enables the learner to modify a class diagram and simultaneously observe the changes in the Java implementation and the other way around.

For visualizing the execution of a program, we chose to use the tool *Java-Tutor* [8] (based on the very similar *Python-Tutor*) [19]. The learners can run a program step-by-step with the possibility to navigate forward and backward while observing the control flow. It also includes a graphical representation of the memory contents, supporting the understanding of related concepts such as, e. g., references.

E. Programming exercises

While in several introductory CS MOOCs the learners have to install an integrated development environment (IDE) for writing their first computer programs, we decided to rely on web-based tool also for this purpose (like [9]). We chose to use *Codeboard* [9] [20] (among several alternatives [21–23]) because of the usability and seamless integration into the edX platform using the Learning Tools Interoperability (LTI) standard.

The programming assignments are graded automatically and the main purpose is to provide helpful feedback to the learner. We therefore implemented tests for each assignment that make heavy use of the Java reflection functionality. While standard unit tests would fail with a compile error if, e. g., an attribute is missing or spelled differently. Reflection makes it possible to determine for a learner's submission if, e. g., all attributes and methods are defined with the correct names, types and parameters. Writing the tests requires more effort than for standard unit tests but can give more detailed feedback for the learners in case of mistakes.

Additionally we integrated the automatic grading and feedback system *JACK* [24] using the external grader interface of the edX platform. Apart from static and dynamic tests, JACK also offers the generation and comparison of traces and visualization of object structures; however, we do not use this extended functionality yet.

IV. Conduct of the Course

We prepared the course on the platform edX Edge and conducted it during the summer holidays 2016 with a limited number of participants as prototype for a MOOC. The course was announced only internally at our university as preparation course for CS basics. Participation was voluntary and did not count towards a grade but we announced to issue informal certificates for successful participation (= obtaining at least 50 % of the possible points in at least 12 of 16 course units).

In an introductory online questionnaire, we asked the participants about their gender, major, and previous programming experience. Additionally we asked about the intentions to complete the course, providing four options (see TABLE I).

The course took five weeks (one week for each chapter) and the targeted workload of the learners was 10 hours per week. The communication among the learners and with the instructors took place entirely in the discussion forum. The main task of the instructor during the conduction of the course was to monitor the forum and to react accordingly, e. g., answer questions or fix problems with the grading system.

In a concluding online questionnaire distributed after the course, we asked for positive and negative textual feedback regarding the course.

V. First Results

The course attracted 187 registrations. For the introductory questionnaire, we received 77 responses (female: 21, male: 52 male, no answer: 4) with a very diverse study background (33 different majors, including, e. g., Biology, Business studies, Engineering, and Mathematics). Regarding programming, 10 participants had no experience, 35 had basic knowledge, and 27 participants had already written a "bigger" program of at least 100 lines of code (no answer: 5 participants).

The discussion forum contained in total 178 posts at the end of the course. However, there was not a lot of discussion and communication among the participants themselves and most posts were answers to the exercises as required by the assignments (see Section III.D). This is presumably also because we did not actively focus on initiating lively discussions in this prototypical conduction of the course.

From the 77 responses of the introductory questionnaire, 41 stated that they want to complete most topics or the whole course (see TABLE I). In general, MOOCs have a rather high dropout [8, 25, 26]. At the end of the course, we were happy that 40 participants gained the course certificate (however, not necessarily the same learners as the 41 from the questionnaire).

[4] http://svgjs.com
[5] http://cindyjs.org
[6] https://blockly-games.appspot.com
[7] http://try.umple.org
[8] http://www.pythontutor.com/java.html
[9] https://codeboard.io

TABLE I. INTENTIONS TO COMPLETE THE COURSE

Option	Answers
I just want to have a look at the course.	14
I want to study some topics that are relevant for me.	18
I want to study most topics of the course.	12
I want to complete the whole course.	29
(No answer)	4
Total	**77**

In the concluding survey distributed after the course, we received 11 answers. The participants proposed specific improvements for the employed interactive exercises, among others to use a more user-friendly web-based drawing tool (or to additionally allow the use of offline software) and to include more difficult exercises. Yet, the overall feedback was encouragingly positive. The learners stated to like the videos, the explanations, the interactive exercises and the overall alignment of the course elements.

VI. Discussion

Based on the experiences with our course LOOP, carefully designed MOOCs seem to be a possibility to reduce the differences in the CS-related prerequisite knowledge of freshmen at university. The advantages of e-learning in general, and the use of interactive exercises in particular, are approved by the learners. Especially in computer science, it seems in many cases possible to rely the education on web-based tools, without the need to install further software on the learners' computers.

So far, we focused mostly on the design and creation of the material and the exercises. In the future, we plan to lay the focus more on the communication aspects and to incorporate also collaborative elements and peer-grading. We are going to offer the course as a MOOC and aim to evaluate the learning processes by mining all data produced by the students. We furthermore plan to measure the effect on the previous knowledge of the freshmen.

Acknowledgment

We thank Alexandra Funke and Marc Berges for developing parts of the course and Elias Hoffmann, Elisabeth Eichholz, and Simon Zettler for testing and evaluating a preliminary version of the course. We furthermore thank the anonymous referees for helping to improve the paper.

References

[1] P. Hubwieser *et al.*, "A global snapshot of computer science education in K-12 schools," in *ITiCSE Working Group Reports*, New York, NY, USA: ACM, 2015, pp. 65–83.

[2] B. Linck *et al.*, "Competence model for informatics modelling and system comprehension," in *IEEE Global Engineering Education Conference (EDUCON'13)*, 2013, pp. 85–93.

[3] W. Hering *et al.*, "On benefits of interactive online learning in higher distance education," in *6th International Conference on Mobile, Hybrid, and On-line Learning (eLmL'14)*, 2014, pp. 57–62.

[4] P. Hubwieser, "Analysis of learning objectives in object oriented programming," in *Informatics Education: Supporting Computational Thinking, 3rd International Conference on Informatics in Secondary Schools - Evolution and Perspectives (ISSEP'08)*: Springer, 2008, pp. 142–150.

[5] J. Wing, "Computational Thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[6] D. Gries, "A principled approach to teaching OO first," *ACM SIGCSE Bulletin*, vol. 40, no. 1, p. 31, 2008.

[7] K. Falkner, N. Falkner, C. Szabo, and R. Vivian, "Applying validated pedagogy to MOOCs," in *ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'16)*, New York, NY, USA: ACM, 2016, pp. 326–331.

[8] T. R. Liyanagunawardena, K. O. Lundqvist, L. Micallef, and S. A. Williams, "Teaching programming to beginners in a massive open online course," in *Building Communities of Open Practice (OER'14)*, 2014, pp. 1–7.

[9] M. Piccioni, C. Estler, and B. Meyer, "SPOC-supported introduction to programming," in *Innovation & Technology in Computer Science Education (ITiCSE'14)*, 2014, pp. 3–8.

[10] C. Alario-Hoyos *et al.*, "Interactive activities: the key to learning programming with MOOCs," in *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'16)*, Norderstedt: Books on Demand, 2016.

[11] M. Kölling, "The Greenfoot Programming Environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1–21, 2010.

[12] P. Hubwieser, "Computer science education in secondary schools – The introduction of a new compulsory subject," *ACM Transactions on Computing Education*, vol. 12, no. 4, pp. 1–41, 2012.

[13] S. Cooper, W. Dann, and R. Pausch, "Teaching objects-first in introductory computer science," in *34th Technical Symposium on Computer Science Education (SIGCSE'03)*, New York, NY, USA: ACM, 2003, p. 191.

[14] S. Cooper and S. Cunningham, "Teaching computer science in context," *ACM Inroads*, vol. 1, no. 1, p. 5, 2010.

[15] P. J. Guo, J. Kim, and R. Rubin, "How video production affects student engagement," in *1st ACM Conference on Learning@Scale (L@S'14)*, New York: ACM, 2014, pp. 41–50.

[16] M. Alonso-Ramos *et al.*, "Computer science MOOCs: A methodology for the recording of videos," in *IEEE Global Engineering Education Conference (EDUCON'16)*, 2016, pp. 1115–1121.

[17] M. von Gagern, U. Kortenkamp, J. Richter-Gebert, and M. Strobel, "CindyJS," in *5th International Congress on Mathematical Software (ICMS'16)*, Cham: Springer New York Inc., 2016, pp. 319–326.

[18] T. C. Lethbridge, "Teaching modeling using Umple: Principles for the development of an effective tool," in *IEEE 27th Conference on Software Engineering Education and Training (CSEE&T)*, 2014, pp. 23–28.

[19] P. J. Guo, "Online Python Tutor," in *44th ACM Technical Symposium on Computer Science Education (SIGCSE'13)*, 2013, p. 579.

[20] H.-C. Estler and M. Nordio, *Codeboard.* [Online] Available: http://codeboard.io/. Accessed on: Dec. 01 2016.

[21] T. Staubitz, H. Klement, R. Teusner, J. Renz, and C. Meinel, "CodeOcean - A versatile platform for practical programming excercises in online environments," in *IEEE Global Engineering Education Conference (EDUCON'16)*, 2016, pp. 314–323.

[22] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. van Roy, "Automatic grading of programming exercises in a MOOC using the INGInious platform," in *European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15)*, 2015, pp. 86–91.

[23] I. Skoric, B. Pein, and T. Orehovacki, "Selecting the most appropriate web IDE for learning programming using AHP," in *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'16)*: IEEE, 2016, pp. 877–882.

[24] M. Striewe and M. Goedicke, "JACK revisited: Scaling up in multiple dimensions," in *8th European Conference, on Technology Enhanced Learning (EC-TEL'13): Scaling up Learning for Sustained Impact*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 635–636.

[25] C. Delgado Kloos *et al.*, "Experiences of running MOOCs and SPOCs at UC3M," in *IEEE Global Engineering Education Conference (EDUCON'14)*, 2014, pp. 884–891.

[26] F. Garcia *et al.*, "A practice-based MOOC for learning electronics," in *IEEE Global Engineering Education Conference (EDUCON'14)*, 2014, pp. 969–974.