Developing a MOOC on Introductory Programming as Additional Preparation Course for CS Freshmen

Elisabeth Wetzinger
Institute of Information Systems Engineering
TU Wien, Austria
elisabeth.wetzinger@ifs.tuwien.ac.at

Bernhard Standl
Institute of Information Systems Engineering
TU Wien, Austria
bernhard.standl@ifs.tuwien.ac.at

Gerald Futschek
Institute of Information Systems Engineering
TU Wien, Austria
gerald.futschek@ifs.tuwien.ac.at

Abstract: At our university, the undergraduate program in Computer Science and Business Informatics (CSBI) faces a high student dropout rate especially during the first year of studies. One reason is the students' heterogeneous pre-university CS education, in particular concerning programming, due to inhomogeneous biographical backgrounds and different types of secondary schools. To tackle this knowledge-gap by levelling up those without or with little programming experience we have developed the massive open online course (MOOC) *Programming with Processing*. The MOOC comprises various learning activities and materials to study, exercise and self-assess basic programming concepts at individual learning speed using the programming environment Processing. The first course run comprised 6 chapters and has been implemented and evaluated with 577 prospective CSBI students. In this practice-based paper, we present the MOOC course design and the development, testing and implementation processes, the pedagogical background and we discuss our lessons learned so far.

Introduction

During the first year of the program in Computer Science and Business Informatics (CSBI) we are confronted with a high dropout rate especially among female students. In order to bring in an improvement, the Faculty of Informatics incorporated the project *START Informatik*, which addresses these challenges by researching the quality of teaching and the learning experience during the first year of CSBI studies and by developing actions to increase quality of teaching. Currently the faculty offers gender homogeneous programming bridging courses of one week and a preparatory course covering CS, maths and programming basic tutorials during a period of two weeks. All are held in September during the three weeks before the actual first term of the academic year starts. Recent evaluations in course of *START Informatik* have shown that female students who have mastered the bridging course accomplish more ECTS than their peers. Table 1 shows that attending the bridging course has in particular a positive impact on female students' study performance.

Gender	with bridging course	without bridging course
male	9.3	9.35
female	8.35	6.81

Table 1: Impact on average number of credits [ECTS] accomplished during first term related to bridging course participation (source: internal study data)

However, despite the success of these two courses, still approximately 50% of the programming lecture participants do not successfully complete the introductory programming course in the first semester. Primary reason is again the heterogeneous pre-university CS and programing knowledge of freshmen as there is a correlation between the secondary school type and the success in this course.

Grade	Technical high school	Regular high school
1	32	4
2	52	29
3	43	20
4	6	3
5	33	73

Table 2: Student performance in the compulsory Introductory Programming course related to graduation of secondary school type [grade 1 = Very Good, grade 5 = Failed] (source: internal study data)

Table 2 visualizes the *Introductory Programming* grades distribution of winter term 2016/17. It shows a significant lower performance among students without technical focus at high-school level. The bridging courses have a limited number of participants and in addition, students might not be able to participate due to job responsibilities or vacation. Despite programming pre-knowledge is not required, for freshmen without any pre-university programming experience the workload in the bridge-course as well as preparatory course might be challenging. In addition, content is usually taught on a concrete operational stage level (tracing level) rather than on sensorimotor or preoperational stage levels (pre-tracing level) (Lister, 2016). However, according to Teague and Lister in (Teague & Lister, 2014) novice programmers always start at sensorimotor stage and after programming for one year still the majority is at preoperational stage rather than on concrete operational stage (Lister et al., 2004).

The overall goal of our efforts is, to create an online learning space, which provides students a space to compensate missing knowledge and skills for the introductory programming lecture. As prospective students are invited to participate at the MOOC during summer, while ahead of the actual academic year where we also already offer bridging-courses on-site. Hence, our MOOC also integrates and enhances such existing measures at our university taking place when the actual lectures are starting. In offering different differentiated courses for addressing different students' level of knowledge, we are also establishing the ideas and methods as developed at the Carnegie Mellon University since the early 90s (Margolis & Fisher, 2002). Hence, in our MOOC students will be able to study at their individual level of knowledge and at their preferred location according to their available time and individual speed. For those students who are not able to attend the bridging courses respectively preparatory courses at the beginning of the term, furthermore will have the possibility to level up their programming experience prior to study start as neither the course duration nor the number of participants are limited.
In the following chapter, we describe the background of this paper followed by the MOOC curriculum development. Subsequently, we present the design and development of the MOOC as well as the delivery of the MOOC. Finally, we briefly summarize first results and lessons learned and conclude this work with an outlook to future work.

Background

Bridging courses in mathematics and programming have been implemented widely as a measure to prepare students for the first year in computer science studies, e.g. (Bausch et al., 2014; Fraser, Malone, & Taylor, 1990). Our university has offered a preparatory course covering CS basics, math and programming as well as a bridging course on Introductory Programming. However, they are on-site courses requiring the attendance of prospective students. Since MOOCs have been introduced in 2012 (Pappano, 2012) a vast number of MOOCs on programming based on various programming languages have been released at platforms, such as edX[1], coursera[2] or Udacity[3] by universities (Dasarathy, Sullivan, Schmidt, Fisher, & Porter, 2014). MOOCs have become a possibility to learn programming independent from location, education or background of the participants (Dillenbourg, Fox, Kirchner, & Wirsing,

[1] https://www.edx.org (last access: 01/2018)
[2] https://www.coursera.org (last access: 01/2018)
[3] https://www.udacity.com (last access: 01/2018)

2014). There also exists a so-called *MicroMaster* program[4] implemented as MOOC, which provides graduate level courses from universities, which are accepted for a future admission on-site. However, there is not much literature on MOOCs on introductory into programming being developed and implemented describing such bridging courses as we are offering it. In related literature MOOCs are applied in introductory programming for children and adolescents using block-based programming languages, such as Scratch[5] or Blockly[6] (Bau, Gray, Kelleher, Sheldon, & Turbak, 2017).

Before we started the MOOC design and development, we explored existing related MOOCs. The University of Technology Graz (TU Graz), for instance, offers a MOOC "Programming with PocketCode"[7] using a block-based mobile programming language. It implements professional animation videos and many practical tasks to attract young learners respectively high-school students. The MOOC is available for a course duration of six weeks. Recently 700 participants attended the MOOC, where each week a video was made available with further information on theoretical background. The authors of the "Programming with PocketCode" emphasize on "making", i.e. setting the focus of learning on practicing how to code (Ebner, Janisch, Höllerbauer, Grandl, & Slany, 2017). However, considering the target group of prospective CS students and the requirements at university freshmen programming courses this MOOC using block-based programming languages might address a younger target user group and hence be too easy and superficial as to improve their start of studies at a university. The University of Technology Munich (TU München) offers a bridging course MOOC on introductory programming called "Learning Object-Oriented Programming (LOOP)"[8], which is hosted at edX. Beyond videos and quizzes it includes a range of interactive web-based exercises and a browser-based IDE with automatic grading. As for the similar target users we have taken their experiences as orientation into account (Krugel & Hubwieser, 2017). TU München uses similar video settings and production techniques as we do. They also include various open source tools and applications in their MOOC. Still, LOOP differs from our MOOC significantly: While TUM implements an objects-first approach and uses JAVA as programming language, we decided for the *Processing*[9] environment and a procedural approach in order to align with our university's bridging course and Introductory Programming lectures. The MOOC described in (Falkner, Falkner, Szabo, & Vivian, 2016) incorporates *Processing* with a media computation approach and emphasizes on exploring the development of digital artworks and animations with code. However, no MOOC exists on introductory programming using *Processing* as programming language for bridging the transition from high school to CS-studies. Hence, we have developed a MOOC following these criteria, which we present in this paper.

Our MOOC addresses young people in the transition-phase between high school and college. Considering this, we chose *Processing* as easily accessible environment, which allows for smooth transition to introductory programming with Java lectures as part of the first semester of CSBI bachelor programs. Furthermore, related studies have found out, that learning programming with *Processing* leads to a higher feeling of confidence for student in developing short programs as they would have felt with learning Java from the beginning on in a traditional way (Chilana et al., 2015).

Considering this, we decided for *Processing* (Reas & Fry, 2014) as the course programming language because:

- The programming language used in Processing is based on JAVA. Even though there also exists a "Python-mode" in Processing, which would allow for operating Processing with Python programming language, we decided for the simplified use of JAVA in Processing as this is the language that is introduced first at our university and used in the main freshmen year programming courses.
- The IDE of Processing and the simplified representation of JAVA makes it easy to learn and it has a simple, intuitive, and easy-to-use IDE; it is fun to work with (Greenberg, Kumar, & Xu, 2012). Beyond the fact, that Processing is easy to understand (Xu et al., 2016), the easy accessibility makes it also possible to realize ideas within a short period of time and furthermore makes it easy to write software for drawing or animation (Reas & Fry, 2006). This is motivating for students as it focuses on coordinates-based graphics and animations from very beginning and thus offers instant and visible results.
- The introduction with Processing also allows for a later introduction into structured programming that is in alignment with the pre-university bridging courses offered at our faculty.

[4] https://www.edx.org/micromasters (last access: 01/2018)
[5] https://scratch.mit.edu (last access: 01/2018)
[6] https://developers.google.com/blockly (last access: 01/2018)
[7] https://www.catrobat.org (last access: 01/2018)
[8] https://www.ddi.edu.tum.de/lehre/moocs/loop (last access: 01/2018)
[9] https://processing.org (last access: 01/2018)

The level of difficulty in our MOOC was set in such way that school students from grade 9 upwards can understand the underlying math used. Thus, e.g., the sinus function was replaced by a piecewise linear function. As pedagogical principle, we introduce basic programming concepts following a pre-tracing approach as introduced by (Teague & Lister, 2014) with explaining the sequence of commands repeatedly. Furthermore, the used spoken phrases are of operational style so that the participants learn to understand the effect of the commands onto the state of the computer. Further, we use practical examples and metaphors to introduce new programming concepts before explaining the respective abstract definition. Hence, we support a bottom-up learning process from concrete examples to general principles and concepts. Therefore we further developed a *golden thread approach* (Standl, Wetzinger, & Futschek, 2017), which is an organizational and methodical-pedagogical approach where content of the MOOC should be easier graspable through improved consistency in content and structure. The golden thread approach aims at optimizing, rearranging and modifying existing structures in a way as to improve the teaching-/ and learning-organization experience. This also includes measures for increasing gender equality aimed at increasing number of female students in CS studies and retaining them. The principle of *golden thread* is implemented using two scenarios, which are developed further step by step throughout all chapters of the MOOC: Max searching for the oldest person in a group, and the computer game Pacman. Usually online courses provide individual learning only, but cooperative learning eases learning and provides higher success rates referring to students' retention rates. Hence, we implement exercises requiring students to interact with each other, to learn from one another and base their exercise on the work of their peers. A mix of media (videos, text, quizzes, and exercises) addresses several perception channels and follow the principle "learning by doing".

Design and Development of the MOOC

We developed the MOOC curriculum (see Table 3) in cooperation with two lecturers and with tutors of the compulsory Introductory Programming lecture at the first semester of CSBI bachelor programs. It is oriented on the needs of the main target groups:

a) Prospective CSBI students, ideally before they start their studies
b) Interested high school students and teachers; students of other programs

#	Chapter Topic
1	Welcome; Computers & programming in a nutshell
2	Introduction to Processing (and the IDE)
3	Variables and (arithmetic) operators
4	Data types and operators
5	Conditions (if, if-else, if-else if-else)
6	Loops (while, for)
7	Arrays
8	Functions
9	Recursions
10	Introduction to JAVA

Table 3: Curriculum structure of the *MOOC Programming with Processing* by chapters

Considering this, the common objective is to bridge between school and university education.

For the development of our MOOC we used *Making a MOOC at TUM* (Medienzentrum, 2016) as a guideline. Based on the overall course curriculum (see Table 3) and the pedagogical concept, we created a detailed concept for every chapter: It involved formulating learning objectives, breaking down the chapters into ordered lists of sub-topics and rating them (crucial, "nice to have", advanced level). Subsequently, based on the detailed chapter concepts, the specific learning object (LOB) types for the parts of the chapters have been selected and the LOBs have been prepared. During all these steps, dynamic feedback loops with the Introductory Programming lecturers and CS didactics experts were implemented. The most important sub-topics of every chapter were selected to be provided as videos (see below). The remaining sub-topics are realized as further course materials and activities.

Learning Platform

Currently, we host our MOOC on the Moodle-based virtual learning platform of our university. The preceding decision

process included an evaluation of the learning platforms coursera, edX, open edX and iMooX[10]. The key aspects, which supported our decision, were the well-maintained technical infrastructure of the already existing Moodle platform and the support services provided by our university, which reduce running costs to zero. Furthermore, compatibility is kept as Moodle supports the LTI-standard. However, drawbacks are, for instance, that we are not able to make our MOOC open by standard and we need to create accounts for users manually. This restricts our MOOC and rather keeps it limited as small private online course (SPOC), but this issue could be solved in future by setting up a separate clone on our virtual learning platform solely dedicated to MOOCs that allows self-registration of users.

Videos and Handouts

The main course LOB are videos. They introduce the core contents of each chapter to the students and demonstrate practical programming examples. The process of creating the videos comprised the script design, the video production and the post-production:

For every video, we wrote a tabular video script in close collaboration with the corresponding professors and senior lecturers as well as CS didactics researchers. The script contains the text to be spoken (word by word) by the speaker as well as stage directions, example code to be programmed live and references to additional visuals. In trial presentations, the presentation time for the video contents were estimated. Following the recommendations of (Guedes da Silva, Moura Santos, Albuquerque Costa, & Viana, 2016) and taking into account the subtopics within the chapter video contents the overall video script for every chapter was divided in separate small videos with a maximum target duration of 7min.

Video Production. We produced two types of learning videos: animations explaining videos for chapter 1 and explainer videos with speakers and live programming combined with animated sequences for the remaining chapters. *Animated cut-out explainer videos* were created based on animations using the Common Craft cut-out library[11], self-made hand drawings and public domain images as well as a speaker off-camera explaining.

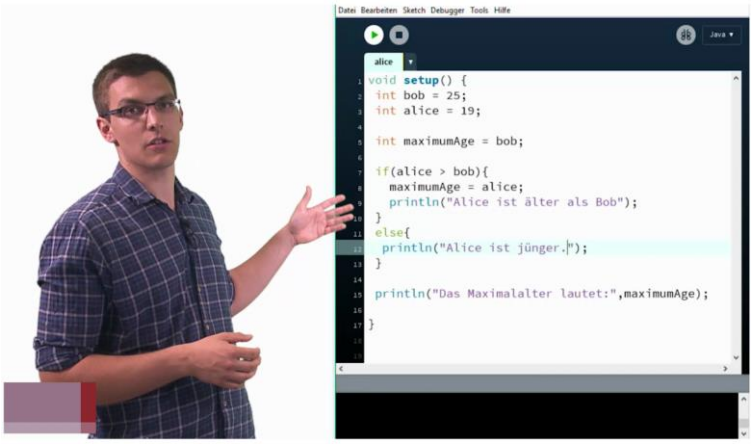

Figure 1: Side-by-side setting showing the speaker and the Processing IDE

Requirements for the speaker selections were experienced programming skills and CS knowledge, thus ideally a CSBI student or staff member, an extroverted and open attitude as well as presentation skills. In order to provide a gender-balanced team of speakers, the two speakers we recruited are one female member of the project team and a male master thesis student. This also had the benefit of being able to keeping the financial costs low. The production itself was carried out by a professional film-production company, which set up a film-studio in one of our office rooms. For the videos, Chroma keying technology was used to show both the speaker as well as programmed code at the same time and to enable the speaker to interact with the code. Concerning the overall video production, we follow the recommendation of (Guedes da Silva et al., 2016) to create personalized and content-focused videos. Hence, we focus in our videos that the speakers are visible as much as possible. They also interact with the visuals and displayed code, e.g. point at code lines or perform live programming. This allows the speaker to take over the role of a personal tutor

[10] https://imoox.at (last access: 01/2018)
[11] https://www.commoncraft.com/cut-out-library (last access: 01/2018)

or coach and to address the participant watching the video directly.
The following screen settings are used:

- Speaker centrally positioned to introduce or conclude topics
- Speaker and visual content or Processing IDE side-by-side to present visual aids or live-programming with the benefit of being able to interact with the visual content due to chroma keying (see Figure 1).
- Visual content or Processing IDE in full-screen to use maximum screen space when presenting and explaining Processing programs and for animations in Chapter 1.

Post Production. The project team reviewed the draft versions of the produced video sequences in two feedback loops and provided feedback, corrections and missing inserts to the production company. The final videos were rendered in as mp4/H.264 with a resolution of 720p. Currently, the videos are hosted at a private YouTube channel and are embedded in the course.

Handouts. For every chapter additional readings were provided as PDF. These handouts summarize video contents, provide additional code examples and include deeper insights on the topics.

Further Course Materials and Activities

We composed our MOOC by combining a variety of course materials and activities to provide students an extensive learning experience and to consider different forms of learning. We produced 23 videos explaining the core concepts and applying those to practical examples using Processing, as well as we developed activities, readings, quizzes and further interactive elements. Following, we briefly present the used types of course materials and activities.

Readings. The content extending the videos and handouts is provided by readings in PDF data format.
Depending on the scope of a chapter, a maximum of five, and at average three, readings are available per chapter. The readings are rather short with an average of three pages per reading (min. 1 page, max. 7 pages). They are consistent in layout and design with the handouts and the syntax highlighting in the Processing IDE. Every reading describes one sub-topic at sensorimotor or preoperational stage levels (Lister, 2016), includes gender-balanced metaphors and application examples as well as various practical code examples. Finally, all handouts and readings together form a comprehensive course script.

Exercises and Quizzes. For each topic exercises were implemented using the Moodle Quiz module. These exercises are self-assessments for the students. To make the exercises more interesting we did not only implement Multiple-Choice questions but also fill-in-the-blanks, Drag-and-Drops (using images, text and code snippets) as well as drop-down selections. An example for a question using fill-in-the-blanks with drag&drop is visualized in Figure 2(left).

Figure 2: Examples for a question on nested loops implemented as fill-in-the-blanks drag&drop using graphics and syntax-highlighted Processing code (left) and for a glossary entry example on ”Initialisierung” (Initialization) (right)

In addition to the quiz-based exercises, practical programming assignments are implemented focusing on graphical programming. For instance, students have to create an artwork using geometric objects and peers have to try to

program a copy of it with only having the rendering of the artwork. Another example is the programming of their own ghost used for their Pacman game and its animation. Two exercises are optional bonus programming exercises and advanced level quizzes to challenge participants in their learning process and to provide an additional learning path.

Programming Examples. Throughout the videos, handouts and readings, code snippets are integrated to explain and visualize the introduced programming concepts. In videos, the speaker programs the code mainly on the fly using the Processing IDE displayed in real-time within the video in a side-by-side or full-screen view. All Processing code snippets which are embedded in the handouts, readings and quizzes the Processing syntax highlighting and font have been used. All code samples presented in the videos are available for download in the respective chapters. This makes it easy for students to run the programs and to experiment with the code, e.g. to modify or extend it.

Glossary. We created a glossary with specific Processing and programming concept terms introduced in the course. It implements the Moodle glossary activity and provides search functionality as well as automatic cross-referencing between text-based information in the course and the corresponding glossary entry. Currently, the glossary holds 51 terms and students have been encouraged to suggest further terms which they want to be added. Figure 2 (right) shows an example for a glossary entry.

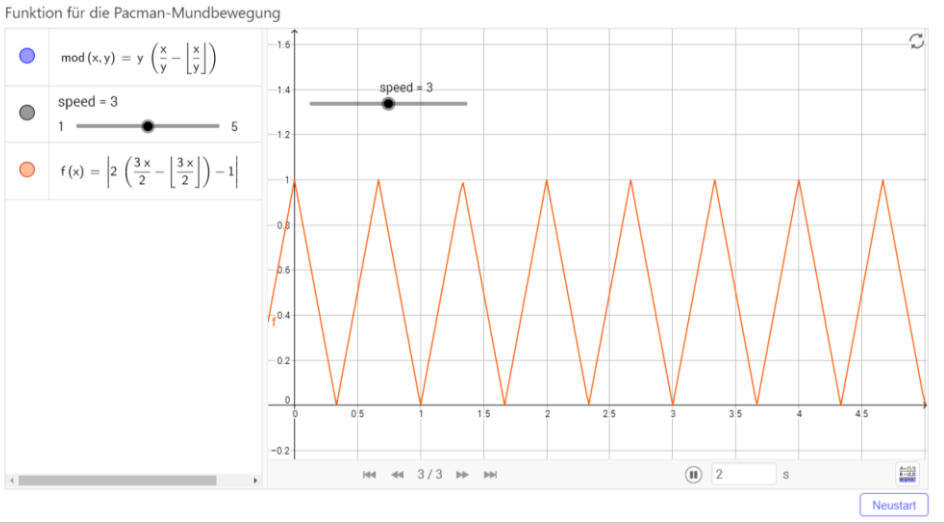

Figure 3: Interactive visualization of a triangle function embedded in the MOOC using Geogebra

Graphical Visualizations. We created the graphical visualizations of geometric functions with the interactive application GeoGebra[12] and embedded it directly in the MOOC (see Figure 3).

Reference of Common Processing Error Messages. To help students to understand Processing error messages and to investigate on potential causes, we created a collection of Common Processing Error Messages. Additionally, we provide a list of reserved keywords in Processing which therefore cannot be used for variable or function names.

Discussion Forum. It has been our priority to offer with our MOOC not only a knowledge source, but also a place for students to connect to each other and to get in touch with our team and lecturers. In order to create such climate for community building, we actively service all discussion forums in the course and we schedule online office hours twice per week. Using the latter, students are able to clarify issues and have their questions directly answered by the course team, whether the issue is related to the course itself or to their start of studies.

Online Office Hours and Course Email. Twice per week, we offer online office hours of two hours each to the course participants using the chat activity. At least one member of the course team is present to discuss with course participants and to answer their questions related to course-specific or administrative topics. In addition, we provide email support using a course-specific email address.

[12] https://www.geogebra.org (last access: 01/2018)

Delivery of the MOOC

Preceding the delivery of the MOOC to the prospective students, we implemented a preliminary testing phase. In this section, we summarize the experiences of both the testing phase as well as the first course run.

Preliminary Testing

Eight test participants of different CS-related pre-knowledge tested the MOOC prototype over a duration of seven days. The test participants were members of the course team, two prospective students, one student at secondary level (15 years old) as well as three lecturers of first year CSBI courses. To collect and receive the feedback we set up a feedback WIKI and provided a guideline for it. We encouraged the test participants to report their findings using the WIKI. Alternatively, we also offered to receive feedback via email. The main findings were (total occurrences are given in parentheses): Typos or missing words (25+), misspeak in videos (9), missing description of technical words (4), exercise description/answer was not clear enough (3), bugs (2), missing files or wrong links (1), missing code (1).

First Course Run

We informed all prospective students about the MOOC during their entrance application process in mid-June. Every student accepted for a CSBI bachelor program received detailed information about the MOOC program by begin of August. Two days ahead of the course opening everyone received his personal course login data. The first course run opened on August 16th, 2017, 6.5 weeks ahead of winter term start. For that course run the first six chapters of the MOOC were implemented completely and released to the participants. The remaining four chapters are still under development and will be available from October 2018.

First Results & Lessons Learned

First results from data, how many participants were involved actively during the MOOC shows that the number of participants decreased from unit to unit as visualized in Figure 4. Starting with 300 students at the end only 65 students finished the course completing all tasks and exams.

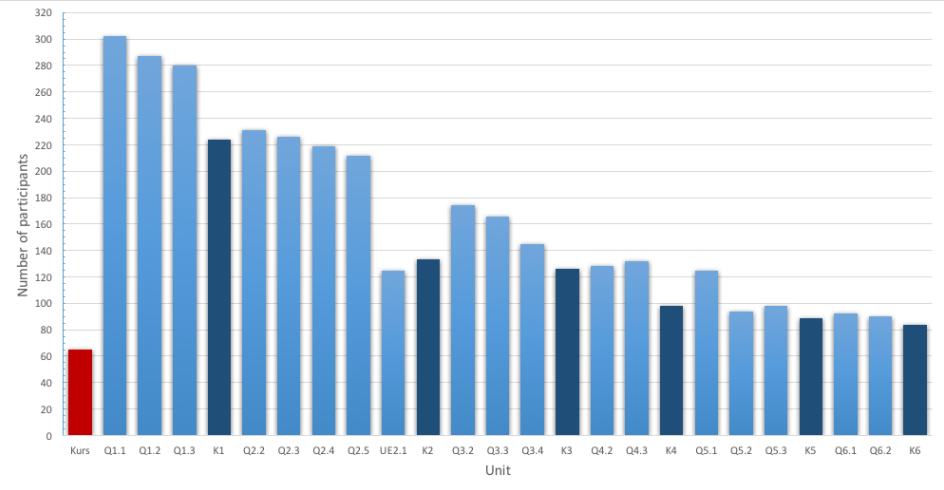

Figure 4: Number of participants finished course units (K1, K2, K3, K4, K5, K6) with sub-chapters.

Within two weeks, more than 300 students have been active in the MOOC and we have received useful feedback based on two large feedback surveys at the start and end of the MOOC and a short feedback survey after each chapter. From the 65 students, who finished the complete MOOC n=49 students reported their feedback. Figure 5 (left) shows, that the MOOC meets the expectation of the majority of the participants.

Figure 5 (right) shows the evaluation results on how the students perceived the difficulty of the MOOC. The results show, that the majority experiences MOOC just right in the level of difficulty with a tendency to it being rather easy than too difficult.

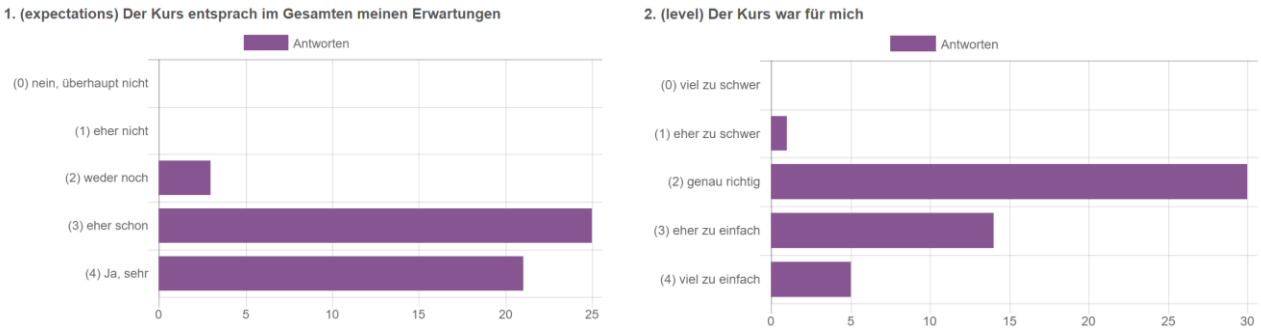

Figure 5: Evaluation results on "How did the MOOC meet the expectations?" (left) and "How difficult the MOOC was experienced?" (right) (0…not at all to 4…very much)

Conclusion

One key lesson learned during the MOOC development was that the process of video production is time-consuming. The use of a teleprompter eases and speeds up production time significantly as it reduces the number of takes needed for every sequence. On an average, we produced one chapter (20-25min of final video) per working day. It is helpful to have a project team member at site who was involved in the script design to check the spoken content. Finally, a detailed script allows for a smoother and faster video production and postproduction. Hence, the script design phase is of uttermost importance to the quality of the produced videos and the production and postproduction time and costs. Keeping consistency across all different material and activity types is challenging as well as creating group work exercises for such a heterogeneous participants group, which also are fun to work at. Close cooperation with the Introductory Programming course staff has proofed crucial and essential in order to provide a MOOC, which is aligned with the preparation course and bridging course as well as of high technical quality. Finally, the whole project of developing a bridging course MOOC on introductory Programming requires a highly interdisciplinary team of not only programming and CS didactics competencies, but also experts in media and digital content production, visual design, project management, digital rights, technical support, marketing, administration and more.

Future Work

In this paper, we presented an approach to tackle the problem on heterogeneous pre-university knowledge in programming. Once winter term and university programming courses will have started in October 2017, the first course run from August 16th to September 30th will be evaluated. For that, we will use course statistics information provided by TUWEL and feedback we will have received via eight surveys across the course chapters. Based on the outcome of the evaluations we will optimize the course. This includes, e.g., adding transcripts or subtitles to the videos, troubleshooting logical or language issues in the course material or adding further practical exercises or group work assignments. Furthermore, adding four more chapters on arrays, functions, recursions and transferring the Processing programming skills to JAVA will complete the course in order to align it with the programming lecture contents of the first CSBI year. Finally, we consider implementing learning paths regarding different experience levels (beginner, intermediate, advanced programmers). The MOOC (chapters 1 to 6) will be released to the public in spring 2018 at iMooX. The next exclusive course track for prospective students is scheduled to start in July 2018. The technical implementation of our MOOC in the Moodle environment of our university will be improved and alternative hosting options will be discussed further. However, the feedback we have received so far indicates that prospective students accept a bridging course on Introductory Programming offered as a MOOC and consider it a useful preparation for their CSBI studies.

Acknowledgments

This work was partly funded by the Austrian Federal Ministry of Science, Research and Economy in course of the HRSM project "MINT MOOCs by TU AUSTRIA" in cooperation with TU Graz (Lead) and MU Leoben as well as by Social City Vienna. We also thank the Introductory Programming staff for their input and excellent collaboration.

References

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable Programming: Blocks and Beyond. *Commun. ACM*, *60*(6), 72–80. https://doi.org/10.1145/3015455

Bausch, I., Biehler, R., Bruder, R., Fischer, P. R., Hochmuth, R., Koepf, W., … Wassong, T. (2014). Mathematische Vor-und Brückenkurse. *Konzepte, Probleme Und Perspektiven. Wiesbaden*.

Chilana, P. K., Alcock, C., Dembla, S., Ho, A., Hurst, A., Armstrong, B., & Guo, P. J. (2015). Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2015–Decem*, 251–259. https://doi.org/10.1109/VLHCC.2015.7357224

Dasarathy, B., Sullivan, K., Schmidt, D. C., Fisher, D. H., & Porter, A. (2014). The Past, Present, and Future of MOOCs and Their Relevance to Software Engineering. In *Proceedings of the on Future of Software Engineering* (pp. 212–224). New York, NY, USA: ACM. https://doi.org/10.1145/2593882.2593897

Dillenbourg, P., Fox, A., Kirchner, C., & Wirsing, M. (2014). Massive Open Online Courses: Current State and Perspectives. *Dagstuhl Manifestos*, *4*(1), 1–27. https://doi.org/http://dx.doi.org/10.4230/DagMan.4.1.1

Ebner, M., Janisch, S., Höllerbauer, B., Grandl, M., & Slany, W. (2017). Pocket Code – Programmieren für Alle mit einem offenen Online-Kurs Abbildung 1: Lego-artiges Bausteine-Prinzip von Pocket Code, 16–19. Retrieved from http://www.fnm-austria.at/fileadmin/user_upload/documents/Magazin/2017-01.pdf

Falkner, K., Falkner, N., Szabo, C., & Vivian, R. (2016). Applying Validated Pedagogy to MOOCs. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16* (pp. 326–331). New York, New York, USA: ACM Press. https://doi.org/10.1145/2899415.2899429

Fraser, B. J., Malone, J. A., & Taylor, P. C. (1990). Tertiary Bridging Courses in Science and Mathematics for Second Chance Students in Australia. *Higher Education Research & Development*, *9*(2), 85–100. https://doi.org/10.1080/0729436900090201

Greenberg, I., Kumar, D., & Xu, D. (2012). Creative coding and visual portfolios for CS1. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12* (p. 247). New York, New York, USA: ACM Press. https://doi.org/10.1145/2157136.2157214

Guedes da Silva, A., Moura Santos, A., Albuquerque Costa, F., & Viana, J. (2016). Enhancing MOOC Videos: Design and Production Strategies. In *Proceedings of the European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS 2016)* (pp. 107–121).

Krugel, J., & Hubwieser, P. (2017). Computational thinking as springboard for learning object-oriented programming in an interactive MOOC. In *2017 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1709–1712). IEEE. https://doi.org/10.1109/EDUCON.2017.7943079

Lister, R. (2016). Toward a Developmental Epistemology of Computer Programming. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education on ZZZ - WiPSCE '16*, 5–16. https://doi.org/10.1145/2978249.2978251

Lister, R., Seppälä, O., Simon, B., Thomas, L., Adams, E. S., Fitzgerald, S., … Sanders, K. (2004). *A multi-national study of reading and tracing skills in novice programmers*. *ACM SIGCSE Bulletin* (Vol. 36). https://doi.org/10.1145/1041624.1041673

Margolis, J., & Fisher, A. (2002). *Unlocking the clubhouse : women in computing*. MIT Press.

Medienzentrum. (2016). *Making a MOOC at TUM A Handbook for Instructors and Course Teams*. (H. Pongratz, M. Stross, & E. Schulze, Eds.). München: TUM Medienzentrum. Retrieved from http://www.tum.de/studium/weiterbildung/oeffentlichkeit/moocs/%5Cnhttps://hochschulforumdigitalisierung.de/sites/default/files/dateien/2016_TUM_MOOC_Handbook.pdf

Pappano, L. (2012). The Year of the MOOC. *The New York Times*, *2*(12), 2012.

Reas, C., & Fry, B. (2006). Processing: programming for the media arts. *AI & SOCIETY*, *20*(4), 526–538. https://doi.org/10.1007/s00146-006-0050-9

Reas, C., & Fry, B. (2014). *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press.

Standl, B., Wetzinger, E., & Futschek, G. (2017). Student Retention: Towards defining measures for improved quality of teaching and learning in the first year of computer science studies. In A. Tatnall & M. Webb (Eds.), *Tomorrow's learning: Involving everyone - Learning with and about technologies and computing (World Conference on Computing Education 2017, Dublin, Irland)*. Heidelberg: Springer International Publishing.

Teague, D., & Lister, R. (2014). Blinded by their plight: Tracing and the preoperational programmer. *Ppig*, (JUNE 2014).

Xu, D., Cadle, A., Thompson, D., Wolz, U., Greenberg, I., & Kumar, D. (2016). Creative Computation in High School. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16* (pp. 273–278). New York, New York, USA: ACM Press. https://doi.org/10.1145/2839509.2844611