Evidence-Based Re-Design of an Introductory Course "Programming in C"

Dion Timmermann
Engineering Education Research Group
Hamburg University of Technology
Hamburg, Germany
Email: dion.timmermann@tuhh.de

Christian Kautz
Prof. for Eng. Education Research
Hamburg University of Technology
Hamburg, Germany
Email: kautz@tuhh.de

Volker Skwarek
Hamburg University of Applied Sciences
Hamburg, Germany
Email: volker.skwarek@haw-hamburg.de

***Abstract*—This paper describes the step-wise re-design of a traditional introductory programming course into a blended learning course. By combining lectures with an online programming environment, short instructional videos, and Tutorial-worksheets, we intend to increase student learning and help students develop the ability to write software. To ensure a successful revision of the course, student learning is closely monitored. The course is changed in separate steps spread over several semesters. By administering a German translation of the SCS1 Assessment at each step, the effects of the changes to the course can be analyzed. The translation of the SCS1 and the data gathering process are described and preliminary data on student learning in the course are shown.**

I. Introduction

Modern engineering is highly dependent on software. Not only a growing fraction of devices contains programmable components, but also the design process of virtually all engineering projects is accomplished with the help of software. Consequently, most engineers trained today can benefit from a foundation in computer science, and especially in programming. Thus, it is not surprising that most engineering degree programs contain at least an introductory computer sciences course ("CS1").

Most introductory programming courses are taught in a traditional manner, with lectures supplemented by practical work. [1] This practical work is either done in laboratories with a tutor present or by the students on their own, where only the finished source code or program is checked by a teaching assistant.

For almost as long as this approach to teaching has been used, studies have reported that the results of such courses are below expectations. Winslow summarizes that "study after study, regardless of the methodology, reaches the same conclusion" that "novice programmers know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs". [2] This is also reflected in the high failure rates of introductory programming courses, with an international average of 32.3% students failing such courses. [3]

This paper describes the re-design of such a traditionally taught course with the aim of improving student learning, especially students' ability to program. In several steps, the course is transformed to adapt a blended learning format which combines e-learning and face-to-face learning. [4]

In the following subsections, we will describe the context of the course. We will then detail the individual changes made to the course and explain our reasoning for each change. Afterwards, we will describe how this change is implemented over the corse of several semesters, reducing the work-load of the teaching staff per semester. The new design of the course is founded on evidence gathered from literature and through tests as part of this research. An overview of the data obtained so far is given in Section IV.

A. CS1 at Hamburg University of Applied Sciences

The CS1 course described in this paper is an introductory programming course held at Hamburg University of Applied Sciences (HAW) in Germany. The course is mandatory for first semester "Industrial Engineering and Management"[1] students, i. e. students that do not major in computer science. It is offered each semester, with about 125 students attending in winter and about 50 students attending in summer term. To receive the 3 ECTS[2] points for this course, students have to pass the exam and 50% of the labs. The course covers basic programming in C, including datatypes, conditionals, loops, arrays, pointers, functions and recursion. It also covers simple algorithms as e. g. bubble sort.

In the summer term 2015, the third author of this paper took over the course. Until then, it was taught in a traditional manner.

B. Programming cannot be learned in a lecture

As stated above, many studies have shown a lack of success in traditional introductory programming education. A common research design for such studies is the analysis of source code written by students. In 1982, Soloway et al. for example investigated the programming skills of students who had completed a single semester programming course. Even when ignoring syntax errors, only 38% of the students in their study were able to write a loop which calculates the average of several numbers with the end of the loop indicated by a sentinel

[1] in German: Wirtschaftsingenieurwesen

[2] European Credit Transfer and Accumulation System; 3 ECTS points equal 90 hours of work

value. [5] In 2001, McCracken et al. reported on a multi-national study where students were tasked with programming an simple calculator. A significant number of answers "showed [either] that the student had some idea about what was needed, but completed very little of the program" or "that the student had no idea about how to approach the problem." [6] However, studies have also shown that students do know the syntax and semantics of individual statements. Thus, what they fail to learn is how to use and connect statements to create a program. [2]

These problems could also be observed among the students attending the traditionally taught course at HAW. This prompted the new instructor to re-design the course. This re-design is focused on a stronger separation between the abstract, theoretical knowledge that students have to learn and the practical skills of programming they have to acquire.

II. Concept of a new Blended Learning Programming Course

The concept of the course was switched from a traditional lecture to blended learning. With blended learning, face-to-face teaching and e-learning are combined in a way that the strengths of both educational formats are combined without any significant drawbacks. [4] In the course described here, face-to-face lectures are used to discuss fundamental concepts and introduce new ideas. The e-learning components support students in testing and developing their conceptual understanding. Additionally, they allow students to practice programming in an environment suited for beginners.

The components of the course and their interaction are explained in more detail in the following subsections of the paper. The e-learning components are developed as part of the Hamburg Open Online University-Project (HOOU) and will be made available as Open Educational Resources (OER).

A. Face-to-Face Lecture

The lecture will focus on programming concepts as well as the design of algorithms.

B. Online Programming Environment

Automated assessment tools have been shown to be helpful in improving student learning. Similar to detailed error messages and warnings from a compiler, these tools can help students by giving quick feedback on the quality of a program, allowing for faster progress through shorter feedback loops. [7]

However, in addition to a compiler and an integrated development environment (IDE), the automated assessment tool would be the third piece of software that students have to install before they can start to practice programming. While the installation and configuration of such software is no problem for experts, it can be quite troublesome for novices. To circumvent this problem, the re-designed course will use an online IDE for all programming exercises. This IDE will contain the programming tasks and allow for the automated assessment of students' solutions. This way, students can practice programming whenever they sit in front of a computer with internet access.

An added benefit of this online IDE is that the interface can be simplified quite easily. Using professional IDEs allows students to get familiar with software they might be using in industry and gives them access to most features they might need at some point. However, such IDEs have significant drawbacks in the context of education. As Reis et al. note, professional IDEs often provide a "complex interface that is bewildering to novices. For beginning students, providing a simple, intuitive user interface is far more important than offering every conceivable feature." [8]

C. Short Video Lectures

When solving programming tasks in the online IDE, students will have access to short video lectures. Each video gives information on one topic, such as variables, input/output commands, or loops. The verbal explanation in the video is supplemented by a screen recording and hand-written notes. Each video ends with a short quiz, which has to be passed before the next video can be accessed.

The videos are produced by the lecturer, 2 student assistants for the video post production and 2 media-assistants for storyboarding and multimedia design. The overall concept was to keep the duration of the videos within the attention span of about 10 to 15 minutes. This has the benefit that students are more likely to view the full video and can find relevant information more easily, as each video is devoted to only one topic. [9]

D. Tutorial-Worksheets focussing Conceptual Understanding

So supplement the programming exercises, so-called Tutorial-Worksheets are developed. These are designed to help students test and develop their understanding of concepts such as *functions*, *loops*, or *arrays*. Tutorials have been shown to significantly improve student understanding in other subjects. [10] The development of the Tutorials is described in detail in a different publication. [11]

E. Focus on C11

To ease the learning process of the students, the version of the C-language taught was changed from C99 to C11. In this new version, some new statements are available which in the old version were rather error prone for novice programmers. Examples are runtime declarations of fields, "bool" variables as native types, and range-based "for"-loops, automatically counting over all elements of a structure.

This helps students avoid simple mistakes in some more complex syntactical tasks.

III. Step-Wise Change of the Course

The re-design of the course as described above requires substantial effort. To make this more manageable, the process is separated into three steps, spread over several semesters. At the time of writing, Step 1 has been completed and Step 2 is being worked on.

A. Step 1

The course was given in the old, traditional format with 8×90 minutes of lecture and weekly recitation sections held by student tutors. The lectures were recorded on video. The strengths and weaknesses of this course were analyzed using the SCS1 Assessment as a pre- and post-test.

B. Step 2

Using the recorded lectures, short video segments are produced for the online learning environment. New videos are recorded when necessary. The exercises for the online IDE are created based on the tasks used in recitation sections. Tutorial-Worksheets are also added to the course. The videos, tasks and Tutorials are selected based on the analysis of the course during Step 1.

The online learning environment is added to the course. The lecture and recitation sections are largely the same as before. Pre- and post-tests are used again to evaluate the effectiveness of the changes and allow further modification. In addition, answers to online tests are analyzed and used for a further improvement of the course.

C. Step 3

The recitation sections are completely replaced by the online learning environment and the lecture is changed to focus on algorithms and data structures.

IV. Evaluation

To assess the course described in this paper and the success of the re-design, pre- and post-tests are used. Based on students' scores in these tests, the so-called *learning gain* [12] or the more general *normalized change* [13] can be calculated. These measures indicate how much a student learned, normalized to how much he/she could have learned. With s_{pre}, s_{post}, s_{max} indicating a student's pre-test score, his/her post-test score, and the maximal possible score, respectively, the student's normalized change c calculates as:

$$c = \begin{cases} \dfrac{s_{\text{post}} - s_{\text{pre}}}{s_{\text{max}} - s_{\text{pre}}} & \text{if } s_{\text{post}} \geq s_{\text{pre}}, \\ \dfrac{s_{\text{post}} - s_{\text{pre}}}{s_{\text{pre}}} & \text{if } s_{\text{post}} < s_{\text{pre}}. \end{cases} \tag{1}$$

As shown by Hake, a course's *average learning gain* is largely affected by the type of instruction but less so by the individual instructor. [12]

As a standardized testing tool, we decided to use the SCS1 Assessment.

A. SCS1 Assessment

The SCS1 Assessment is a standardized 27-question multiple-choice test that evaluates students' understanding of programming using code completion, code tracing, and qualitative questions. It covers programming fundamentals, loops, conditionals, arrays, function parameters and return values as well as recursion. The test uses a pseudo language, for which a reference sheet is provided with the test. The usage of the pseudo language allows researchers and instructors to use the test regardless of the programming language students have learned.[3] [14], [15]

To our knowledge, the SCS1 is the only available standardized assessment on programming.

B. Translation and Adaption of the SCS1 Assessment

The course described in this paper is taught in German and does not require any knowledge of English. Thus, we cannot expect students to be fluent in English. We therefore decided to translate the SCS1 Assessment to German.

The original English version of the test was translated into German by the first author of this paper, who is a native German speaker. This German version was then translated back into English by a bilingual student. Then, the original English version and the re-translated version were compared. Each difference between the two documents was discussed and, where necessary, changes were made to the German translation to better reflect the English original.

Finally, the German translation and the original English version of the test were given to two students. Their task was to check the source code in the test for any missing words or inconsistencies that occurred during the translation and to check for any typing errors or unspecific language in the German version.

We suspected that many students who took the test had no prior contact with computer science or programming. For these students, most questions in the test would not be intelligible, because of the frequent use of subject specific language. Additionally, students were only given 55 minutes to complete the test. Even for experts, this is very demanding. We wanted to differentiate *questions that students read but could not answer* from *questions that students did not read*, e. g. because they did not have enough time. To achieve this, we added a sixth answering option to each question, phrased "I am not sure". While this change may affect the test results, it was in our opinion necessary. Students were instructed to select the answer "I am not sure" if they had read a question, but would have had to guess the correct answer.

C. Administration of the Pre- and Post-Test

As part of Step 1 of the re-design process, student learning in the un-changed course was measured using pre- and post-tests. The SCS1 was given as a pre-test during an orientation course, one day before the lectures of the semester started. While this orientation course was not mandatory, virtually all first semester students attended. The test was not mandatory. Although students did not receive credit for the test, they all showed genuine effort.

During the last lecture, but before the last laboratory, the SCS1 was administered as a post-test. Again, the test was not mandatory and students did not receive any credit for the test. Again, all students did participate and showed genuine effort.

[3] Please contact `scs1assessment@gmail.com` for access to the SCS1 Assessment.

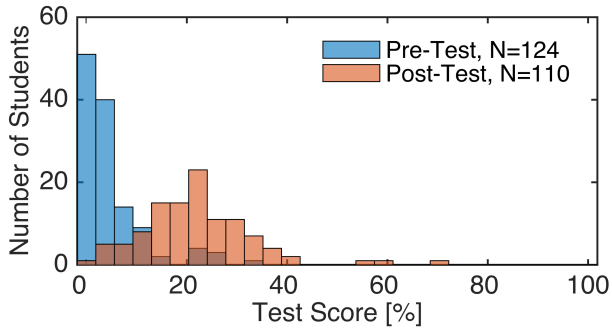

Fig. 1. Distribution of pre- and post-test scores for all students that participated in any of the tests in Step 1. Blue bars are used for the pre-test and red bars for the post-test. Dark areas are caused by an overlap of both bars.

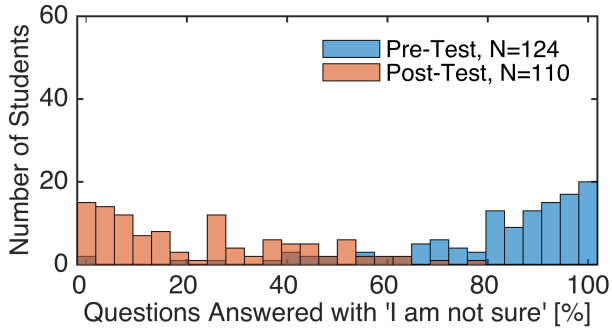

Fig. 2. Distribution of the percentage of questions answered with "I am not sure" in the pre- and post-test in Step 1. Blue bars are used for the pre-test and red bars for the post-test. Dark areas are caused by an overlap of both bars.

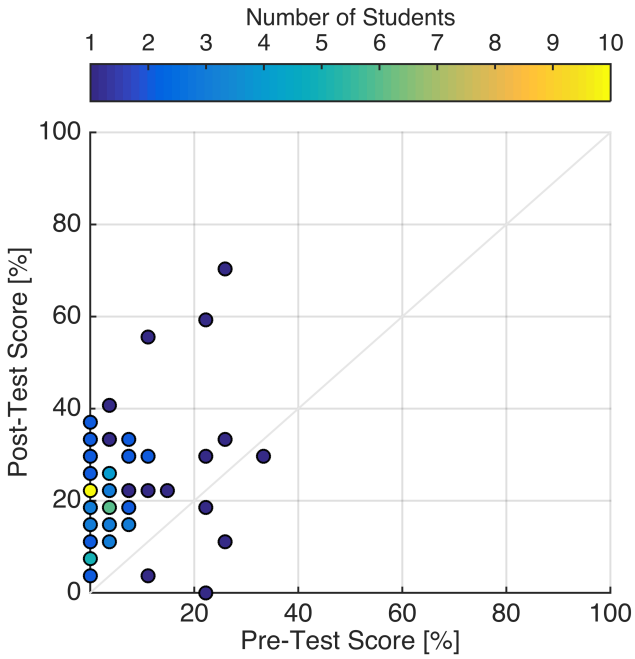

Fig. 3. Comparison of pre- and post-test scores for all students that attended both tests in Step 1. N=78

Self-generated identification codes were used to match pre- and post-tests by the same student. [16]

D. Test Results

In all, 124 students participated in the pre- and 110 students participated in the post-test. Based on the self-generated identification codes, 78 students participated in both tests. As can be seen in Fig. 1 the pre-test scores were very low, showing a clear bottom-effect. The highest pre-test score was 33 % and the vast majority of students received 0 or 1 point, resulting in a mean score of 4 %. While many students stated after the test that they were unable to interpret the questions, the number of times students selected the answer "I am not sure" (shown in Fig. 2) is less strongly skewed than the overall score. This indicates that many students thought they could at least answer some of the questions, but did not know the correct answer.

The average score in the post-test was 22 %. As can be seen in the histogram in Fig. 1, the scores are almost normally distributed. The number of times students selected the option "I am not sure" has also decreased significantly (see Fig. 2).

As can be seen in Fig. 3, only 5 students had a lower score in the post-test than in the pre-test. However, many students only gained a few points in the post-test. Consequently, the average learning gain reached by students is 15.0 %. Compared to the average learning gain of 23 % measured by Hake for traditionally taught courses, this value is quite low. However, it is not clear if learning gains measured in completely different subjects with different tests can be compared. [12]

V. Summary

This paper described the re-design of an introductory programming course. Pre- and post-tests showed that the *average learning gain* in the traditionally taught course was quite low. These results align with reports from literature. A detailed description of the planed re-design of the course into a blended learning course was given. In the future, pre- and post-tests will be used to measure the learning gain of this re-designed course. These data can then be compared to the ones presented in this paper, allowing for an objective evaluation of the re-design.

Acknowledgment

Part of this work is funded by the state of Hamburg, Germany, within the Hamburg Open Online University-Project (http://www.hoou.de/).

References

[1] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1076/csed.13.2.137.14200

[2] L. E. Winslow, "Programming pedagogy a psychological overview," *ACM SIGCSE Bulletin*, vol. 28, no. 3, pp. 17–22, 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=234872

[3] C. Watson and F. W. Li, "Failure rates in introductory programming revisited." ACM Press, 2014, pp. 39–44. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2591708.2591749

[4] S. Hadjerrouit, "Towards a Blended Learning Model for Teaching and Learning Computer Programming: A Case Study." *Informatics in Education*, vol. 7, no. 2, pp. 181–210, 2008.

[5] E. Soloway, K. Ehrlich, J. Bonar, and J. Greenspan, "What do novices know about programming?" in *Directions in Human-computer Interaction.* Norwood, NJ: Ablex, 1982.

[6] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," *ACM SIGCSE Bulletin*, vol. 33, no. 4, pp. 125–180, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=572181

[7] R. S. Pettit, J. D. Homer, K. M. Holcomb, N. Simone, and D. S. A. Mengel, "Are automated assessment tools helpful in programming courses?" in *122nd ASEE Annual Conference & Exposition*, Seattle, WA, Jun. 2015.

[8] C. Reis and R. Cartwright, "Taming a professional IDE for the classroom," in *ACM SIGCSE Bulletin*, vol. 36. ACM, 2004, pp. 156–160. [Online]. Available: http://dl.acm.org/citation.cfm?id=971357

[9] J. Kim, P. J. Guo, D. T. Seaton, P. Mitros, K. Z. Gajos, and R. C. Miller, "Understanding in-video dropouts and interaction peaks in Online lecture videos." ACM Press, 2014, pp. 31–40. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2556325.2566237

[10] N. D. Finkelstein and S. J. Pollock, "Replicating and understanding successful innovations: Implementing tutorials in introductory physics," *Physical Review Special Topics - Physics Education Research*, vol. 1, no. 1, Sep. 2005. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevSTPER.1.010101

[11] D. Timmermann and C. Kautz, "Design of Open Educational Resources for a Programming Course with a Focus on Conceptual Understanding," in *Proceedings of the 44th SEFI Annual Conference*, Tampere, Finland, 2016.

[12] R. R. Hake, "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses," *American journal of Physics*, vol. 66, no. 1, pp. 64–74, 1998. [Online]. Available: http://scitation.aip.org/content/aapt/journal/ajp/66/1/10.1119/1.18809

[13] J. D. Marx and K. Cummings, "Normalized change," *American Journal of Physics*, vol. 75, no. 1, p. 87, 2007.

[14] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment," in *Proceedings of the twelfth annual International Conference on International Computing Education Research*, Melbourne, Australia, 2016.

[15] A. E. Tew and M. Guzdial, "The FCS1: a language independent assessment of CS1 knowledge." ACM Press, 2011, p. 111. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1953163.1953200

[16] D. Timmermann, J. Direnga, J. Lund, and C. Kautz, "Evidence-Based Education Research with Matched Pre- and Post-Tests using Self-Generated Identification Codes (SGICs)," in *Proceedings of the 44th SEFI Annual Conference*, Tampere, Finland, 2016.