# Design of Open Educational Resources for a Programming Course with a Focus on Conceptual Understanding

**D. Timmermann**[1]
Research Assistant, Engineering Education Research Group
Hamburg University of Technology
Hamburg, Germany
E-Mail: dion.timmermann@tuhh.de

**C. Kautz**
Professor for Engineering Education, Engineering Education Research Group
Hamburg University of Technology
Hamburg, Germany
E-Mail: kautz@tuhh.de

Conference Key Areas: Open and Online Engineering Education
Keywords: student understanding; tutorial worksheet

## INTRODUCTION

Programming is part of many – if not most – engineering degree programs. However, traditional introductory programming courses are often considered difficult by students. [1] Internationally, almost a third (32.3%) of students fail their introductory computer sciences course. [2] Consequently, the improvement of programming education will be beneficial for many students.

Funded by the Hamburg Open Online University (HOOU) Project, the Engineering Education Research Group at Hamburg University of Technology is currently engaged in the development of new Open Educational Resources on programming education. In the following, we will describe how we identified the most relevant concepts to address in the materials, why we decided to use “Tutorials” instead of traditional end-of-chapter type exercises, and show an example for the new materials. We hope this paper allows us to get feedback from the education research community to further optimize our materials.

## 1 METHODOLOGY

Our work is based on the assumption that students develop misconceptions (often also called *alternative conceptions*) when constructing their own understanding of subject matter. [3–5] While this cannot be avoided, carefully designed learning environments can help students overcome their misconceptions through a process of conceptual change. [6] In order to create instructional materials for such environments, one has

[1]Corresponding Author
D. Timmermann
dion.timmermann@tuhh.de

to be aware of conceptual understanding of subject matter. [7] To investigate students' understanding, Concept Inventories can be used.

## 2 PRIOR RESEARCH ON STUDENT UNDERSTANDING OF PROGRAMMING

There is a rich body of research on programming education. Many studies investigate the benefits and drawbacks of different programming languages and programming paradigms as well as pedagogical strategies. [1, 8] There are also plenty of studies investigating the learning process required to become an expert programmer and the reasons for bugs in novices' source code. This kind of research is often named psychology of programming, i. e. the investigation of the learning of programming using approaches from cognitive psychology. [5]

The research that investigates the reasons for bugs is based on the assumption that at least some of the mistakes students make are not random. This assumption is similar to the idea of common student misconceptions, i. e. students' alternate conceptions that have to be overcome in order to reach expert level thinking. [3, 4] As part of this research, as well as independently from it, several misconceptions on programming have been identified. [5, 9] As Clancy notes "Users of modern programming languages and environments have no shortage of opportunities to form misconceptions", but he also stresses that "many of the 'old' aspects of programming" as e. g. loops and function return behavior are not fully explored. [9]

To measure the frequency of common misconceptions and student understanding in general, many researchers employ Concept Inventories. Concept Inventories are standardized tests that evaluate student understanding of all concepts relevant for one topic. In the field of computer science, there are Concept Inventories for some topics, as e. g. "Logic Design". [10] The newly developed SCS1 Assessment [11] seems to be the only concept inventory on programming that is publicly available.

## 3 ANALYSIS OF STUDENTS' DIFFICULTIES IN PROGRAMMING

In this section, we are going to present how we measured students' difficulties with programming. We will show how we identified the most troublesome concepts and explain why we believe these are caused by common misconceptions.

We intend to develop materials that specifically address the topics most troublesome for students. To identify these topics, we measured students' conceptual understanding of programming using the SCS1 Assessment before and after they took a traditional introductory programming course at Hamburg University of Applied Sciences. The university course is described in more detail in a different publication. [12]

### 3.1 The SCS1 Assessment

To investigate students' conceptual understanding of programming, we used the SCS1 Assessment, a standardized 27 question multiple-choice test that evaluates students' understanding of programming using code completion, code tracing, and qualitative questions. The test covers programming fundamentals, loops, conditionals, arrays, function parameters and return values as well as recursion. It uses a pseudo language for which a reference sheet is handed out together with the test. The use of this pseudo language makes it possible to compare test performances regardless of the programming language students have learned.[2] [11, 13]

### 3.2 Identification of Troublesome Concepts

As part of our research, a German translation of the SCS1 was given twice. [12] Once, as a pre-test several days before the course started and then as a post-test during the last lecture, but before the last lab exercises. 124 students participated in the pre-test and 110 in the post-test. Using self-generated identification codes, each test was marked with an identification code by the student who answered it. [14] For 78 pre-tests, we were able to find a post-test that was answered by the same student. As the SCS1 is quite new and has not been used by our research group before, we did not have

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

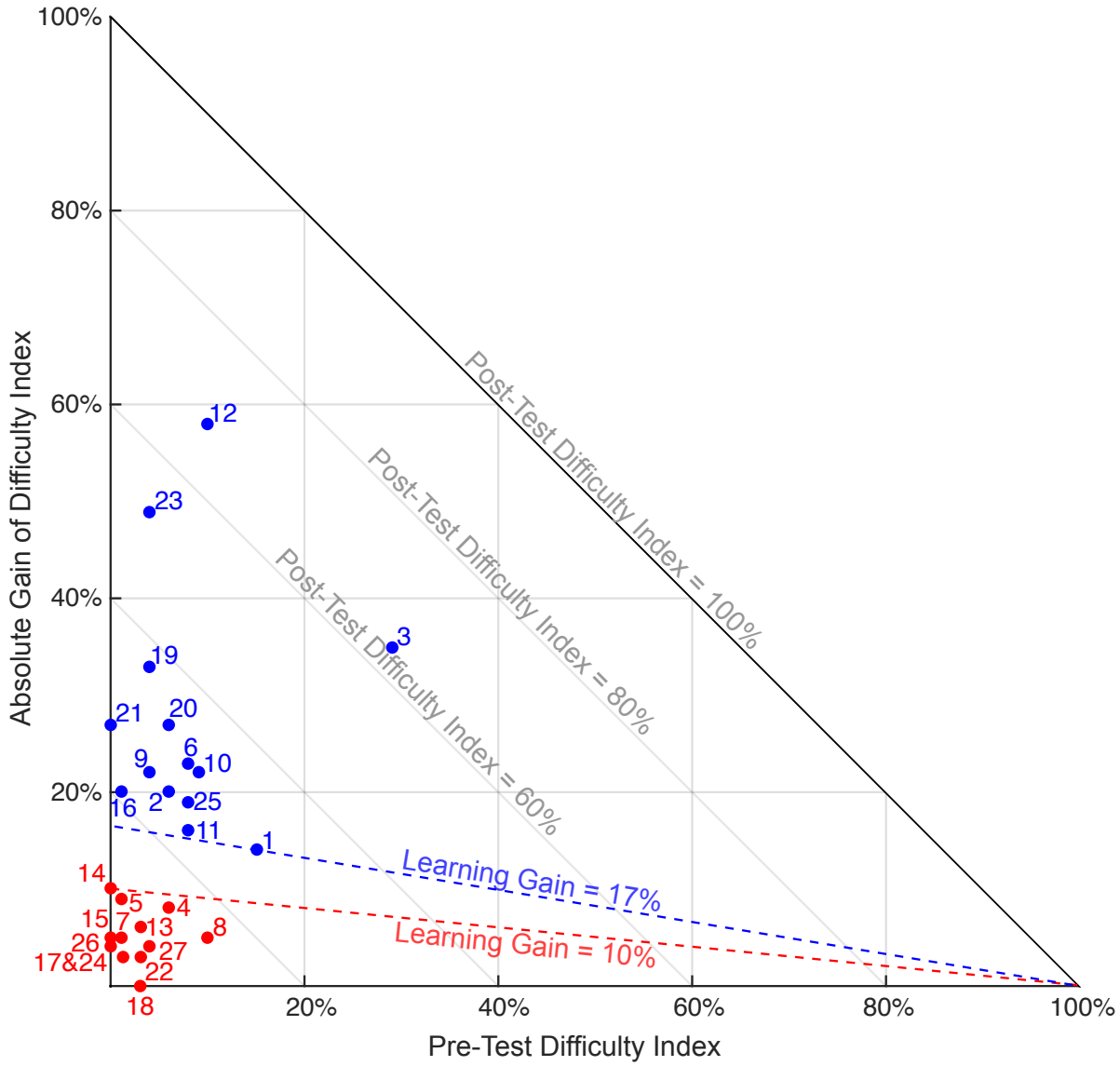



*Fig. 1:* Difficulty indices (percentages of correct answers) of all questions in the SCS1 for the pre- and the post-test. While the horizontal axis shows the pre-test difficulty index, the vertical axis shows the absolute gain of the difficulty index from the pre-test to the post-test. The post-test difficulty index can be found on the diagonal axis. The questions colored red were considered particularly troublesome. The dashed lines indicate learning gains calculated as a fraction of absolute gain and maximum possible gain. N=78.

access to reference values for the difficulty of the questions. Thus, we were only able to compare the questions with each other but not the performance of the students in the course under investigation with the performance of students attending other courses.

To identify the questions most troublesome for students, we used the plot shown in Figure 1. This plot is similar the one used by Hake in [6] to compare the effect of traditionally taught courses with the effect of courses using interactive engagement. The horizontal axis indicates the pre-test difficulty index, i.e. the percentage of students that correctly answered a question. The vertical axis indicates the gain in percentage points, a measure for how many more students correctly answered a question in the post-test. Consequently, the diagonal lines indicate post-test item difficulty. Each point corresponds to one question. The data shown are based on the 78 pre- and post-test pairs. The data for the tests without a respective post- or pre-test, however, do not differ much.

The pre-test difficulty indices of most questions were between 0 % and 15 %, indicating that students had quasi no knowledge of programming before attending the course. The post-test difficulty indices of the questions were between 3 % and 68 % and thus varied much more. Consequently, there was also a high variation in the absolute gains of the difficulty indices.

The 13 questions denoted by red data points, were considered to be most troublesome for students. As can be seen, these data points are quite noticeably separated from the blue data points, marking questions *not* considered most troublesome. The difference between both groups of questions is best described by the learning gain, the ratio of the absolute gain of difficulty index and the maximum possible gain of the difficulty index. While the learning gains of all questions marked in blue are at least 17 %, the learning gains for all questions marked in red do not exceed 10 %. The learning gains are highlighted by the dashed lines in Figure 1.

Based on these troublesome questions, we identified the most troublesome concepts. For each concept in the test, we identified how often it occurred among the troublesome questions, compared to how often occurred among all questions. Our analysis revealed that the most troublesome concepts were *nested definite loops*, *arrays*, and *recursion*. While it is not surprising that recursion is troublesome for students [9], we were surprised to find that students had trouble with nested definite loops.

### 3.3 Distribution of answers

The SCS1 has 5 answering options for each question, resulting in an expected difficulty index (percentage of correct answers) of 20 % when students guess. However, in many concept inventories, the difficulty indices of some questions are actually significantly lower than the guessing probability. These low difficulty indices are caused by some of the answering options being good distractors, i. e. answers that seem to be correct when a student has a certain misconception. Distractors are usually chosen so that each common misconception maps to exactly one distractor.

We had expected that many students in the pre-test had no prior contact with programming and would quite possibly not even understand the questions. In order to not force these students to guess an answer, we added a sixth option to each question. This option was labeled “I am not sure” and students were instructed to select this option if they would otherwise have to guess the correct answer.

The histogram in Figure 2 shows how many students selected the option “I am not sure” for how many questions. While the majority of students selected “I am not sure” for most

questions in the pre test (see Figure 2a), the majority of students did only select this option for a few questions in the post-test (see Figure 2b). This shows that students were quite unsure about subject matter in the pre-test but felt fairly sure about their answers in the post-test. The low number of correct answers in the post-test indicates that students did not guess but instead erroneously thought to know the answer.

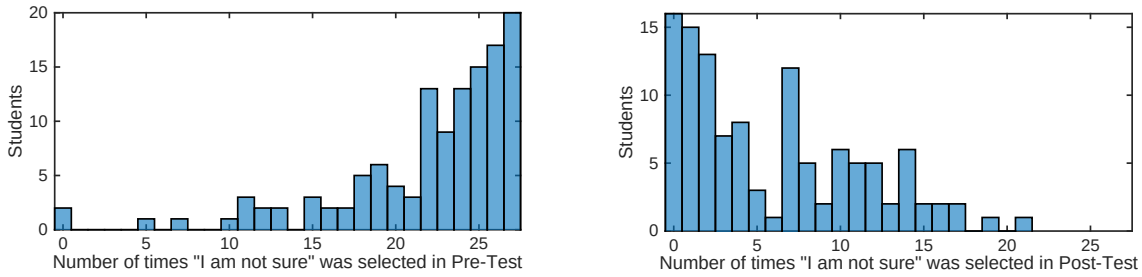
Our analysis showed that the incorrect answers were not randomly distributed, but that instead one or two incorrect answers were selected for most questions. Thus, there must be common misconceptions that many of the students had.

In the following section, we will show how common misconceptions are addressed in subjects as physics, mechanical enginnering, or electrical enginnering.

## 4 ADDRESSING KNOWN STUDENT MISCONCEPTIONS IN OTHER SUBJECTS

To help students overcome common misconceptions, several didactical approaches have been suggested and tested. [15, 16] One of these approaches are the so-called “Tutorials”. Tutorials are worksheets first developed by the Physics Education Research Group at University of Washington. They are usually worked on “face-to-face” by students in groups of three or four. Tutorials guide students through a careful analysis of a concept by using many short questions. The questions are designed so that students are forced to investigate and discuss every relevant aspect of the concept.

McDermott explains the reasoning behind this as follows: “There are difficulties for which no significant conceptual change appears to take place unless students become engaged at a deep intellectual level”. [7] One of the various methods used to achieve this deep intellectual engagement is to provoke students “to make a particular error [...]. The underlying conceptual or reasoning difficulty is then explicitly addressed”. [7] One example for this would be the “elicit, confront, resolve” [7] approach used in several Tutorials. An example for this strategy can be found in a Tutorial on the electric potential and open circuit voltage developed by our group. [17] The Tutorial starts with a small quiz. Students are given the circuit shown in Figure 3 and are asked to rank the voltages $V_{\mathrm{AB}}$, $V_{\mathrm{AC}}$, $V_{\mathrm{AD}}$, and $V_{\mathrm{BC}}$ by their absolute value. With this ranking, students’ knowledge on the voltage across open circuits/switches is *elicited*. The Tutorial then introduces students to the concept of the electric potential and the use of color coding to visualize the different potentials in a circuit. The middle section of this Tutorial is shown in Figure 3. Students are asked to use color coding on and to calculate the potentials in the circuit from the quiz (see 2.1. in Figure 3). Based on the electric potentials of the circuit they then calculate the voltages from the quiz again (see 2.6 in Figure 3) and are then *confronted*



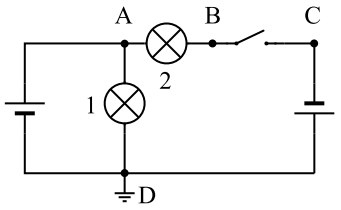

*(a)* Pre-Test, N=124. *(b)* Post-Test, N=110.

*Fig. 2:* Histogram of the number of questions for which a student indicated they did not know the answer to a question.

Consider the following circuit, which contains identical bulbs, ideal batteries with a source voltage of 1.5 V, a ground and an open switch. The switch stays open until Task 3.

2.1 Colorize all points and wires, according to the color coding introduced in part 1.1.

2.2 Assign a potential to each color.




Circuit 2

2.6 Rank the voltages $V_{AB}$, $V_{AC}$, $V_{AD}$ und $V_{BC}$ according to their absolute value. Use the potentials to determine the voltages. Does your answer match the answer you gave in the pre-test?

Discuss your findings with a tutor.

2.7 What is the voltage between both ends of the open switch?

*Fig. 3:* Excerpt from the middle of the tutorial worksheet.

(see 2.7) with their previously different answer in the quiz. The *resolution* of this cognitive conflict is not shown in the figure.

## 5 DESIGN OF TUTORIALS FOR COMPUTER SCIENCE

When we started this project, we originally planned to design regular programming tasks, similar to end-of-chapter tasks in many textbooks. This is a proven format for programming exercises and would have allowed students to practice programming in a manner similar to programming in a work environment. However, as problems in programming often have several possible solutions, we could not have predicted with certainty how students' solutions would have looked like. Consequently, we could have hardly "forced" students to make certain mistakes or ask questions on specific parts of their code. Additionally, students could get stuck of things we felt much less important at that point, as e. g. the correct inclusion of a library.

To mitigate these issues, we decided to design Tutorial-worksheets, which were described in the previous section. Tutorials allow us to address specific concepts much better, as the questions can be more specific. We can show students excerpts of code without having to show the rest of the program. Additionally, we can much better provoke students to make the errors we intend to discuss instead of having to deal with the errors students make. Thus, we have more control over what students think about.

In the following subsections, we will outline our plans for the Tutorials.

### 5.1 Concepts Addressed in the Tutorials

As described above, we have found the concepts of *nested definite loops*, *arrays* and *recursion* to be most problematic for students. These are the concepts we will first develop Tutorials for. Each concept will be addressed in a separate Tutorial. However, as

## 3 Nested Loops

Examine the source code at right.

```
1  for (y = 1; y <= 3; y++) {
2      for (x = y; x <= 3; x++) {
3          printf("O");
4      }
5      printf("\n"); // Line Break
6  }
```

3.1 What does this code print? Explain your reasoning.

3.2 Consider the following conversation between three students. With which statements do you agree? With which statements do you disagree?

Student 1: A for-loop is used when something should be repeated a fixed number of times. In this case both loops count to four. As the outer loop is for counting the rows and the inner loop for counting the 'O's in each row, a square of 3-by-3 'O's is printed. (:D)

Student 2: You are right, for-loops are used to repeat something a fixed number of times. How often the inner loop is repeated, however, depends on the outer loop. Each time the inner loop is entered, the variable x is initialized with the current value of y. Thus, a triangle is printed. (:O)

Student 3: But both loops increment the counting variables with the "++" expression. Thus, in the lower lines, where y is larger, x must also be larger. The triangle must be wider at the bottom than at the top. (:o)

3.3 In the following, the table at right will be filled to visualize the values of the counting variables x and y.

a. What is the first value the variable y is assigned?

b. What is the first value the variable x is assigned? Mark this combination of values for both variables in the table at right by writing an 'A' into the field.

c. How do the variables change when each line of code executed? Mark each step with a letter ('B', 'C', …).

| | | x | | | | |
|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 |
| y | 0 | | | | | |
| | 1 | | | | | |
| | 2 | | | | | |
| | 3 | | | | | |
| | 4 | | | | | |

d. How do the values in the table compare to your answers in 3.1 and 3.2?

e. Which aspects of your entries in the table are represented in the code output? Which ones are not?

*Fig. 4:* Early Example of the middle section from a Tutorial on Nested Loops.

the understanding of recursion most likely requires a solid understanding of functions, we plan to also develop a Tutorial on functions.

### 5.2 Example for a Tutorial

Figure 4 shows the middle section (page 3) of a Tutorial on nested loops. This section follows the *elicit*, *confront*, *resolve* approach introduced above. In Task 3.1, students are asked what the output of the given source code would be. With this task, their understanding of the nested for-loops is made explicit (*elicit*). Then, students are *confronted* with different interpretations of this code and asked if they agree with those interpretations. To answer this question, they have to analyze and criticize the reasoning of three fictitious students. Then, a technique for visualization that was introduced at the beginning of the Tutorial (not shown in Figure 4) is applied again so that students can check their interpretation. Ideally, this last task *resolves* all remaining issues.

## 6 SUMMARY

We used the SCS1 Concept Inventory to analyze the programming concepts most troublesome for students attending a introductory programming course at Hamburg University of Applied Sciences. We found that students had most trouble with the concepts of *nested definite loops*, *arrays* and *recursion*. The incorrect answers to many questions concerning these concepts were not distributed evenly, but instead students favored certain answers, indicating they did not guess but instead applied a misconception in their reasoning.

To help students overcome misconceptions, these need to be addressed explicitly. We plan to develop Tutorials, structured worksheets with many short tasks prompting students to think about concepts in depth. Examples for such worksheets were given above. As Tutorials haven been shown to be effective in helping students overcome common misconceptions, we are confident the newly developed OER will also be helpful to students.

## ACKNOWLEDGMENTS

We would like to thank Prof. Skwarek from Hamburg University of Applied Sciences, who made this research possible by providing access to his students and valuable lecture time for conducting the pre- and post-test. This OER-project is funded by the state of Hamburg, Germany, within the Hamburg Open Online University-Project (http://www.hoou.de/).

## REFERENCES

[1] Robins, A., Rountree, J., and Rountree, N. (2003) Learning and teaching programming: A review and discussion. *Computer Science Education*, **13**, 137–172.

[2] Watson, C. and Li, F. W. (2014) Failure rates in introductory programming revisited. *Proceedings of the 2014 conference on Innovation & technology in computer science education*.

[3] Wandersee, J. H., Mintzes, J. J., and Novak, J. D. (1994) Research on Alternative Conceptions in Science. Gabel, D. L. (ed.), *Handbook of Research on science teaching and learning*, Macmillan.

[4] McDermott, L. C. (1993) Guest Comment: How we teach and how students learn - A mismatch? *Am. J. Phys.*, **61**, 295.

[5] Ben-Ari, M. (2001) Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, **20**, 45–74.

[6] Hake, R. R. (1998) Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, **66**, 64–74.

[7] McDermott, L. C. (1991) Millikan Lecture 1990: What we teach and what is learned - Closing the gap. *Am. J. Phys.*, **59**, 301.

[8] Hazzan, O., Lapidot, T., and Ragonis, N. (2011) Research in Computer Science Education. *Guide to Teaching Computer Science*, pp. 47–62, Springer London.

[9] Clancy, M. (2004) Misconceptions and Attitudes that Interfere with Learning to Program. Fincher, S. and Petre, M. (eds.), *Computer Science Education Research*, pp. 85–100, Taylor & Francis.

[10] Herman, G. L., Loui, M. C., and Zilles, C. (2010) Creating the digital logic concept inventory. *Proceedings of the 41st ACM technical symposium on Computer science education*, pp. 102–106.

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

[12] Timmermann, D., Kautz, C., and Skwarek, V. (2016) Evidence-Based Re-Design of an Introductory Course “Programming in C”. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Erie, USA.

[13] Tew, A. E. and Guzdial, M. (2011) The FCS1: a language independent assessment of CS1 knowledge. *Proceedings of the 42nd ACM technical symposium on Computer science education*.

[14] Direnga, J., Timmermann, D., Lund, J., and Kautz, C. (2016) Design and Application of Self-Generated Identification Codes (SGICs) for Matching Longitudinal Data. *Proceedings of the 44th SEFI Annual Conference*, Tampere, Finland.

[15] Mazur, E. (1997) *Peer Instruction: A User’s Manual*. Prentice Hall.

[16] Novak, G. M., Gavrin, A., Patterson, E., and Christian, W. (1999) *Just-in-time teaching: blending active learning with web technology*. Prentice Hall series in educational innovation, Prentice Hall.

[17] Timmermann, D., Lehmann, F., and Kautz, C. (2015) Using Potential to Help Students Understand Voltage: First Steps in Implementing Effective Instruction. *Proceedings of the 43rd SEFI Annual Conference*, Orléans, France.