An individual training program for learning programming at university

by Julia Moschitz

Abstract
At the beginning of their computer science studies students have often problems with learning programming. These problems are different and individual. Therefore we developed an individual training program for students based on constructivism, which responds to the special needs of every student, even if they are taught in a group. This individual training program was tested during a teaching experiment at university. This paper shows the individual training program for students and the results of the experiment.

Keywords Computer science education research, constructivism & learning programming

1. Introduction

One of the hardest tasks for students when they want to pass the first semester of computer science at university are courses for learning programming. Especially, at the first semester a lot of students drop out of the study computer science. Universities try with different strategies to minimize the drop-out rate at the first semester. These strategies vary from self-assessment centres to special courses for beginners, where they can catch up their missing knowledge in area of computer science.
At university courses for learning programming are in most cases split into a lecture and an accompanying lab. In the lecture the professor(s) present(s) theoretical information about programming. This kind of instruction is often teacher-centric and the students' activities are minor during these lessons. Some universities try to introduce active learning in their lectures in different ways [4][12][13]. Additionally to lectures some universities offer an accompanying course for learning programming.
At our university we also have problems with a high drop-out rate in the computer science studies and a high drop-out rate in the courses of introductions in learning programming [7]. During a research study we asked the students about their problems with learning programming. In the data evaluation of this online-survey we identified problems of students when they wanted to learn programming. The problems of the students they mentioned were different and individual. So we decided to create an individual learning program for learning programming in a group which offers a possibility to learn programming in a constructivist way.

2. Research Design

The individual training program for learning programming is one of the outcomes of a research project which deals with learning programming through competences. In the first step of this research project we defined some basic competences of learning programming through literature analysis and surveys with experts [6]. We evaluated these surveys with the method 'analyzing of content' according to Mayring[10]. At the next step we specified learning objectives based on the basic competences, which can be ordered by the revised Bloom's taxonomy [8] to see the different levels of learning programming. In connection to the curricula of computer science for becoming a teacher the most important learning objectives were defined. From these learning objectives and competences a training program was developed with regard to the special needs of learners who want to learn programming. In winter term 2011 we compared two different homogenous groups of students to test the individual training program for learning programming. At first the students participated in an online survey. In this online survey we asked for the pre-knowledge of the students, their problems of learning programming and they had to find solutions for a few tasks which deal with different competences of algorithmic thinking. According to the result of this survey the students were split into two homogenous groups. The first group was the control group which was trained actively but not individually. The second group learned individually with the new training program. During this period all students wrote their own learning diaries and we collected data through participant observation. [5] In the middle and at the end of the semester the students had to pass a test, which assesses the main competences of programming. Before each test, the students filled in a self-assessment-survey which asked the students, how they think about their competences of learning programming. At the end the students had the possibility to give us feedback about the training program. We evaluated all the collected data which we had collected during this research project with a mix of qualitative and quantitative methods. Some of the research project results are mentioned in this paper.

3. Learning strategy of learning programming

At the beginning of a course as a teacher you have to decide which kind of programming language you will use and which kind of programming paradigm the programming language supports. Every programming language follows at least one programming paradigm. In our teaching methods at first we followed an imperative programming paradigm. [2] We decided to use the programming environment Scratch [11] for the introduction of learning programming at the university, because the students, who attended this course, wanted to become a teacher at secondary school. Later when the students knew the basic of programming we used other programming paradigms as well.

3.1 Constructivism

The reasons, why we decided to choose a constructivist learning environment may be found in the advantages of this kind of learning theory. Tom Boyle describes five basic principles of constructivism like authentic learning tasks, interaction, encourage voice and ownership in the learning process, experiences with the knowledge construction and metacognition. [4]
We tried to follow these ideas and used didactical methods like active learning, learning diaries and self – regulated learning. In studies regarding such kind of learning methods scientists [4] discovered, that students, who are more actively involved in their learning process are higher motivated, develop better skills in connecting pre-knowledge, new ideas and remembering information.

Self-regulated and active learning are one of the main characteristics of the developed training program. Self-regulated learning is a teaching method, which offers students an unrestricted learning environment, where students are invited to regulate their learning process themselves [9]. In active learning environments students are active in the learning process. Students do not only listen, they reflect about their learning process, think about solutions for problems and discuss questions in groups [3] [9].
Students and teachers take on their roles in a different way in an open learning environment. In student-centric and active learning environments the learner is in the focus of the learning process and environment. The learners have to be more intrinsically motivated, proactive and need more information of their own learning strategy to manage this kind of learning environment [3] [9].Like the role of the students, also the role of the professor or teacher has changed in his/her role in such learner-centric environment. The teacher changes his role to a coach, who supports the students. For example if the students have some questions, the coach will give some suggestions, but not the whole solution to the students. Additional he or she describes the exercises and asks them critical questions to help the students to start thinking about their problems. If necessary, the teacher may motivate the students, but this depends on the grade of self-regulated environment.

4. Individual Training Program

The individual training program is based on different learning objectives. For building these learning objectives we defined a set of competences which are important for learning programming. First we tried to split up these competences of learning programming in smaller competences. The result of this process was a set of more than 100 different competences dealing with programming.

4.1 Basic competences

In this chapter we describe exemplarily basic competences of programming. We divided the competences in four different areas: Basics, Data structures, Sorting and Advanced competences in programming. Additionally to the competences we built following levels, where every competence can be classified:

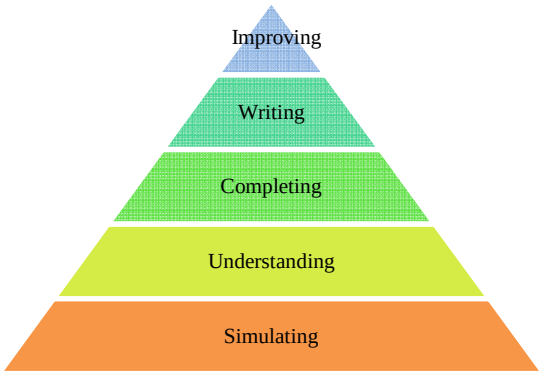


Figure 1: Levels of learning programming

Simulating of algorithms means to play a given algorithm with examples of e.g. numbers. At this step not every student really understands the given algorithm. Assuming that we defined an own category of understanding a given algorithm. One possibility to see if someone really understands a

given algorithm is that the student explains the algorithm to someone else. Completing algorithms is a competence where students fill some missing information in a half-finished algorithm. Writing algorithms means the competence of finding a solution to a problem and writing it down. Writing down happens on different levels. For example, at first to write the solution with own words in mother tongue, on the second level in pseudocode and in the third level in a specific programming language. Improving means to improve for example algorithms to determine redundant code or to give a more efficient solution.
In the introduction of learning algorithms, data structures and programming the scope is that the students learn basics of programming like usage of variables, implications, loops and different data structures in basic algorithms. These general competences can be transferred to learning objectives, which can be ordered in a matrix with the four mentioned steps. In the following table the learning objectives for variables and their assignments for beginners are mentioned.

Steps	**Learning objective** **Use of Variables**	**Explanation**
Simulating	Simulating the assignments to variables	Simulating assignments using one or two variables.
Understanding	Understanding assignments	The student understands easy algorithms, where up to 5 variables are used.
Completing	Completing assignments	The student is able to fill in missing statements of variable assignments
Writing	Writing Assignments	The student is capable to write an easy algorithm, where up to 3 variables are used properly.
Improving	Improving assignments	The student is able to improve given assignments in the aspect of avoiding redundant code or to find a faster solution for a problem.

Table 1: Example for a learning objective

4.2 Feedback & Reflecting the own learning process

Feedback is one of the most important points in the individual training method to train the students individually. Via Feedback we try to motivate the students, give them hints for improvements of their solution and which kind of exercises they may do to get better in programming.
In this training program are a lot of points where students get feedback. Every student gets individual feedback for his/her solution he/she found at home. Additionally the students have the possibility to ask for individual feedback or for their developed program during the laboratory courses. To students who have a lot of problems with learning programming, we give additional support by telling them, which kind of examples they should try to solve to improve their competences.
Reflecting the own learning process is the second important point for students during a self-regulated learning environment. For activating this metacognition the students have to write their own learning diary. They write an entry what they did, where they still have problems with the stuff and how they find a solution for their problems. We give them feedback every week, sometimes advices how they can solve their problems and questions to activate their reflection on their own learning process. This way is an additional opportunity to communicate with the students in a confidential way.

4.3 Laboratory Course

The content of the training program may be defined regarding to the learning objectives, the curricula and the special needs of the students. Of an educational prospect it would be good to harmonize the content of the lecture and the accompanying laboratory course. The scope of this laboratory course is to deepen the knowledge of programming in a practical way. For example in the lecture the students get theoretical information about loops. In the accompanying laboratory the students have the possibility to do some practice on this topic on different levels.
The laboratory courses should not be attended by a number of students exceeding 15 - 20. If more students are attending the course, it will be more difficult for the teacher(s) to keep the overview of all students' individual learning process. The laboratory courses are accompanied by at least one lecturer. The duration of the laboratory course should be more than one and a half hour. Because it is more comfortable for the learners to have 45 – 60 minutes for the individual learning process. In the table below the training plan of the laboratory course is written.

Time	Content of the laboratory course
Beginning 15 min	Welcome Discussion of open questions Explanation of new exercises in the pool
Practice 45 min –60 min	Phase of unrestricted, active learning
Ending 15 min	Presentation of solutions

Table 2: Time table and content of the laboratory course

At first the students are welcomed and it is time for discussing open questions, give some general feedback for the last week in plenum. After this phase new exercises according to the content of the lecture are explained. In the unrestricted learning phase the students have the possibility to choose different exercises of the exercise pool, if they want. During this time it is also possible to choose their own learning materials which they have by themselves. So the students are not forced to do exercises, if they do not want.
The students may choose their individual learning strategies and the way they want to learn.
The exercise pool offers various exercises according to the learning objectives, which are mentioned in the next chapter.

4.4 Examples of the exercise pool

Depending on the basic competences of programming and the learning objectives we developed different exercises. The used methods and the scope of these exercises are different. Some of the exercises contain open questions, where the scope is to create an algorithm. Other exercises invite the student to discuss or explain an algorithm or to improve a given algorithm.
This chapter shows for instance two exercises with their objectives and competences which should be trained with this special exercise. At the beginning when the stuff of learning objectives is constricted and level is easy, it is possible to train only one competence. If a competence is more complex, for example understanding recursion, the students have to deal with other basic competences like, loops or use of variables. So at the beginning it is quite easy to emphasize an exercise to one competence. If the complexity grades of competences are increasing, it is nearly impossible to train only one competence without training another competence indirectly.

The exercises are made with Scratch. However it is possible to translate all of these exercises to other programming languages easily. Every exercise has a name, lists the competence(s), which can be trained with this special exercise. Also the level of the exercise and the necessary pre-knowledge are mentioned. Additional information states if it is an easy exercise or an exercise, you need advanced knowledge in this area. In the table “classroom organization” the students find suggestions about the learning form.
The first exercise deals with assignment of variables. The class organization is to work in pairs or in groups. The learning objective is to familiarize generally with assignment of variables, simulate the given algorithm and to simplify the algorithm.

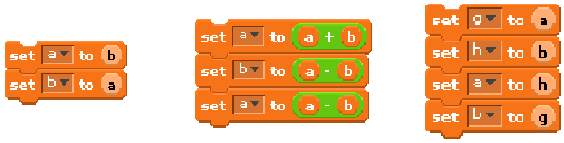
Exercise 1: Assignments of variables
Learning objectives: familiarize with assignments of variables, understand given algorithms with assignments of variables, improve a given algorithm,
Pre-knowledge: You have to know what variables and their assignment are and operations like add and subtract.
Level: easy
Classroom Organization: pair work or team work
Questions: Find out: What are the following algorithms doing? How can you simplify one of these algorithms? Write down your ideas and solutions. **A)** **B)** **C)** 

Table 3: Exercise with assignments to variables

The first example ‘A’ deals with the assignments to the variables ‘a’ and ‘b’. In fact they get the same value, but this is not obvious. The second assignment is redundant and may be cancelled for improving the algorithm. The second example ‘B’ is a variant of swapping two variables without using a temporary variable. The objective here is to understand the assignments. The third example ‘C’ mentions swapping of two variables with two temporary variables. The aim is to simplify the algorithm and use only one temporary variable.
Table 3 shows only an example for one kind of exercise. Another possibility for an example for this kind of exercises is shown in the next task that deals with enhancing the knowledge of binary search. This exercise is for advanced students. The aim of this exercise is to complete a half-done algorithm.

Exercise 2: Binary search
Learning objectives: familiarize with assignments of binary search, understand binary search, find mistakes in algorithms, create exit condition for loops in binary search.
Pre-knowledge: You have to know what variables and their assignment are. You have to know how loops and exit conditions work. You have to understand the principle of binary search.
Level: advanced
Classroom Organization: pair work or team work
Questions: What do you have to change in the following algorithm that binary search works? 

Table 4: Exercise with Binary search

The first mistake is in the first conditional statement. The list ‘a’ and the variable ‘m’ are changed. The second mistake is in the last command. In the functional call the variables ‘m’ and ‘a’ are exchanged.
These two examples are only a short overview, how tasks can look like. However, it is possible to create a range of different tasks which train different competences of programming supporting different learning strategies.

5. Practices of the individual training program

We tested this kind of individual training program during the course ‘introduction to learning algorithms, data structures and programming’ at the university with two comparable homogeneous groups. Traditionally this introduction is split into two different courses: a lecture and an accompanying laboratory course. The lecture follows a mostly teacher-centric method and is held by two different lecturers. The lectures try to integrate student-centric methods during the lectures. In these lectures the students get some knowledge about programming, algorithms and data structures. These lectures take place weekly for one and a half hours, in average ten times per semester. At the end of the semester is an exam, which the students have to pass. The active learning part of the students is less during the readings. However the students may deepen their knowledge in the accompanying laboratory courses in addition to the readings. The laboratory course takes place weekly for one hour, ten times in average per semester. Every week the students have to solve a given example at home and have to write their learning diary.

In the first group (control group) the students present their solutions which they found at home. They get feedback of their colleagues and the professor. The first group was taught in this traditional learning environment of the laboratory course which we described at the beginning of this chapter.
In the second group (experimental group) we tested the individual training program for learning programming.
In the middle and at the end of the semester tests ask all students about their competences they have learned during their courses. All the data we collected during this time we evaluated and interpreted. In the following chapters you find the results of the individual training program.

5.1 Learning Experiences

One aim of the individual training program was that more students pass the course. The following table shows the development how many students attended the course.

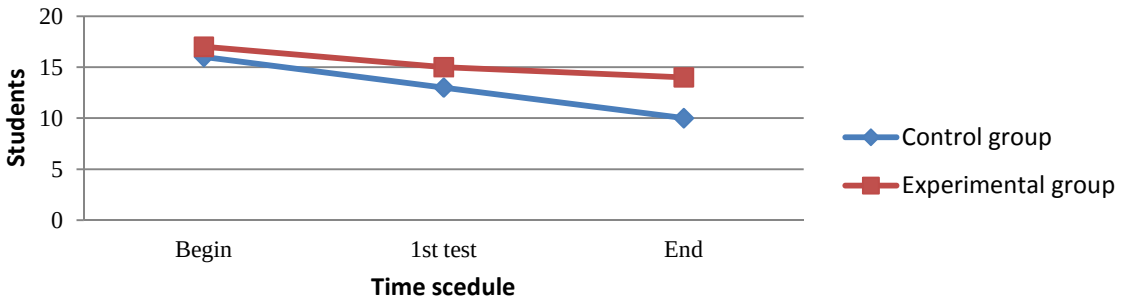


Figure 2: Comparison of the progress of attending students for each group

At the beginning of the semester 33 students started with this course. The control group started with 16 students and the experimental group with 17 students. After the first test 13 students attended the course control group and at the end of the semester 10 students passed the course. In the experimental group after the first test 15 people went on with their studies. In the end 14 students passed the course.
The majority of the students preferred to work in groups together. Most of the students work in pairs or in a group of three and used the exercise pool for their own learning process.
We had a lot of students with different mother tongues in the experimental group and it is remarkable that most of the students prefer to work with students who are able to talk in their mother tongue. We also invited the students to support their colleagues, when they thought they are finished with their own learning process. The students enjoyed helping each other and the students realized how difficult it may be to explain an algorithm or solution to someone else. All of the students told us clique atmosphere was delightful.
The motivation of students was different. Some of the students were highly motivated, some of the students did not accept the self-regulated and active learning environment. If the students were not intrinsically motivated, we tried to motivate them through methods like asking questions or offering help. We are not able to identify the factors, because the factors of influences are complex and it is hard to isolate them. Possible factors are that the learning materials are too easy or too difficult, social reasons, wrong learning strategies or stress.
However, the highly motivated students liked to stay voluntarily longer for finishing their discussion or their solutions for their problems.

The preparation of a learning environment which emphasizes on self-regulated and active learning takes a lot of time. It takes a lot of time to prepare good questions and examples for the students. Additionally to the preparation the teacher spends a lot of time for giving individual feedback for learning diaries and the solutions of students. For the learning success it is important, that the teacher has a good overview of the learning process of every student. This overview is necessary for asking the right questions and giving helpful suggestions. For students who had serious problems with learning programming we supported additional help per email or we gave them some of the examples of the laboratory courses, to do more practice at home. The students who accepted the offer of additional help had more chances to pass the course.

6. Conclusion

We developed an individual training program for students which supports constructivism, self-regulated and active learning strategies. We tested this program with 33 students during an experiment at the university with the scope to decrease the drop-out rate of students who begin to study computer science. As a result we recognized that more students who joined the individual training program passed the course compared to the control group.

7. References

[1] E. Bolshakova, Programming Paradigms in computer science education. Information Theories & Applications, 12, 2005 pp. 285-290.

[2] C.C. Bonwell and J. A. Eison, Active Learning: Creating Excitement in the Classroom. ASHEERIC Higher Education Report No.1, George Washington University, Washington, 1991.

[3] T. Boyle, Constructivism: A suitable pedagogy for information and computer sciences?, 1st LTSN- ICS Annual Conference, Heriot- Watt, Edingburgh, 2000, pp. 2- 5.

[4] M. Cavanagh, Students' experiences of active engagement through cooperative learning activities in lectures. Active Learning in Higher Education, 12(1), 2011, pp. 22-33.

[5] K. Dewalt, B. Dewalt: Participant Observation: A Guide for Fieldworkers, Alta Mira Press. 2010

[6] U. Flick: An introduction to Qualitative research, Sage Publications Ltd. 2014.

[7] G. Futschek, Algorithmic Thinking: The Key for Understanding Computer Science, In R. T. Mittermeir (Eds.), Lecture Notes in Computer Science, Berlin-Heidelberg: Springer, 2006, pp.159 -168.

[8] D. Kratwohl, A Revision of Bloom's Taxonomy: An Overview. Theory into Practice, 41(4), 2002, pp. 212-224.

[9] D. C. Leonhard, Learning Theory, Westport: Greenwood Press. 2002.

[10] P. Mayring: Forum: Qualitative Social Research. 1(2), 2000, pp. 105-114.

[11] "Official Scratch Website" 2013. [Online]. Available: http://scratch.mit.edu/ [Accessed 18 3 2014]

[12] V. Tirronen and V. Isomöttönen, Making teaching of programming learning-oriented and learner-directed. In A. Korhonen (Ed.) Proceedings of the 11th Koli Calling International Conference on Computing Education Research New York: ACM, 2011, pp. 60-65.

[13] K.J. Whittington, Infusing Active Learning into introductory programming courses. Journal of Computing Sciences in College, 19 (5), 2004,pp. 249-259.