CHALLENGES IN TEACHING PROGRAMMING

Marcin Fojcik[1], Martyna Katarzyna Fojcik[2], Sven-Olai Høyland[1], & Jon Øivind Hoem[1]
[1]Western Norway University of Applied Sciences (Norway)
[2]Volda University College (Norway)

Abstract

Teaching is a profession that helps learners to gain new knowledge and insight. Therefore, a teacher needs to choose what to teach the students and how to approach them in an engaging and understandable way. In teaching programming, choosing the content and engaging students can be a challenge because the term programming is used in a variety of ways and contexts, which in turn demands different competencies. This paper uses the Didactical Triangle to discuss some challenges that arise when teaching programming on content, teacher, and student level. Some challenges arise from the structure of programming (syntax, interfaces, approaches, experience, and qualifications), while others are developed from the individual context of the learning situation (role of the teacher, students' motivation, expectations). While programming in computer science is relatively well described in the subject literature, programming in other professions is not well defined. Teaching computer programming in different courses can cause different challenges. Some situations of learning programming might be difficult for computer science students, while other situations might cause challenges for «non-data» students. This paper will present teachers' experiences combined with the theoretical view of challenges that arise when teaching programming in different study programs.

Keywords: *Programming skills, digital competencies, 21st-century skills, didactics in IT education, introduction to programming.*

---

1. Introduction

"Why is programming hard?" asked Guzdial in his paper (2003, p.1). He is not the only researcher that asked this question, but as he stated himself, "It may be that "What makes programming hard?" is not the most fruitful question to ask" (p.21). There are challenges in introducing programming, some are similar to challenges in teaching in general or teaching technological courses, but some are unique to programming education.

Hartree (1950) stated that there are two stages to organize a calculation for an automatic digital calculating machine. He distinguished them as programming and coding, where the first term consisted of structures of breaking down and dividing the calculations in "a sequence of elementary operations which the machine can carry out" (p.248), and the second term described writhing instructions in a way that the machine would execute. These stages were separate at the beginning of the computer era. Today many people overuse the term programming or use it as a synonym for coding, or "[p]erhaps we don't know yet what programming really is or what it could be" (Guzdial, 2003, p. 21). Blackwell stated that since there are different views of programming, then "when people say they are programming, we should not question whether this activity is genuine programming, but instead analyse their experiences in order to understand the general nature of programming activity" (2002, p. 208).

This paper will present theoretical perspectives on the challenges that the authors experience in teaching programming. The aim of this paper is to analyse and structure the challenges that arise in teaching programming using the Didactical Triangle.

2. Challenges in practice

There are general challenges in teaching and specific challenges when teaching programming (Blackwell, 2002; Guzdial, 2003). For example, presenting information clearly and interestingly, motivating the students, keeping a preferred speed, assessing students, giving students feedback, and many more. To understand the challenges in teaching programming, one must understand the didactics of

programming. For the purpose of this paper, the Didactical Triangle (Kansanen & Meri, 1999) was used. The model emphasizes communication between a teacher, a student, and content, with special attention to the context of a learned situation that affects the teacher and the student.

2.1. Content

2.1.1. Different approaches. Programming has a set of rules and structures that a program needs to follow. However, there are still many options on how to plan and build a program. One approach is to look at procedural programming. When the programmer gets several new features and writes a program, the student can use the procedural method to enrich the code from what they have previously learned. This approach builds student knowledge step by step, always returning to the presented facts, even if that might share a narrow perspective on the new topic (Berglund & Lister, 2010). Another approach is to look at the most significant advantages (and differences) of the new topic, present the ideas first, and then go slowly back to recap how the new topic fits with the rest. (Berglund & Lister, 2010).

2.1.2. Specialized content to the subject. The content in teaching programming will vary according to the course it is presented in. For example, in the Norwegian school curriculum introduced by Kunnskapsløftet (LK20), the concept of computational thinking is used to connect algorithms to systematic problem solving and support «thinking like a computer» (Wing 2006). This approach is also understood as a way of experimenting, tinkering with technology (Csizmadia et al., 2015). Computational thinking is both seen as a means to uncover a problem field, as well as necessary when specific sub-problems are to be solved. In teacher education and in schools, building and programming small robots have been introduced to facilitate encounters between computational thinking and the connection between the physical properties of robots and the behaviour that is implemented virtually in code. When students build and code their own robots, they open up for discussions about how automatons become influential on an individual level and affect us at the societal level. This approach may not be as relevant in teaching programming to students in technical subjects, but in teacher education, it is very relevant to address the pupils' way of computational thinking and learning (Fojcik et al., 2020).

2.1.3. The choice of interfaces. When programming with a text-based interface, users may experience difficulties when it comes to overview and the program's internal structure. This may make it difficult for novice programmers to learn how to program (Blackwell, 2002; Guzdial, 2003). Different interfaces have been developed to address such challenges. Block-based programming languages, like Scratch, make the users able to program by dragging and dropping visual blocks. On the other hand, text programming languages such as Python and JavaScript can be more complex and better suited for advanced tasks.

2.2. Teacher

2.2.1. Experiences with programming. Teachers need to demonstrate interesting cases, show good practices, and instil the joy of programming in students. This can be done by someone who truly knows what he/she is doing, both in the subject's content and in pedagogical approaches. If not, the results might not be comparable to the students. The same goes for professors that have not been involved in software development. Then, the only experience shared with students can be a theoretical one. (Berglund & Lister, 2010).

2.2.2. Updating qualifications. "One of the greatest dangers in teaching is the routine, and habitual repetition of actions often observed in one's teachers or colleagues" (Czerepaniak-Walczak, 2014). This is particularly evident in the teaching of engineering subjects related to computer science. This area is quickly changing all the time. By updating the teacher's qualifications and experience, one might avoid repetitions. At the same time, the industry introduces many new terms and ideas that are more relevant for the students when they apply for jobs after finishing their studies.

2.2.3. Motivating students. In many subjects' students will have very different motivations. As a teacher, one will have to place the programming tasks in contexts where the students see the larger relevance and importance of the specific tasks they are given by the teachers.

2.3. Students

2.3.1. Learning approach. If students have a misconception about how programming works, their first meeting with programming can be confusing. Many students start their learning process with memorization and creating habits they do not fully understand. When the students choose a code sequence that works, they might use it again in a different setting to see if it is still working. This might

create a “reward” for this habit, even if the student does not understand why it works (or does not work). (Ertmer & Newby, 2013). Presents that constant patterns, repetitive actions (not necessary with understanding), signals for positive and negative responses can be desirable and help at the beginning of the course. Students have to learn names and definitions. But the programmer cannot rely only on repetition and memorization. It is necessary to have understanding, problem-solving skills, to talk with others – to divide the problem into smaller parts, where the parts can be worked on separately. Such an algorithmic approach may also make it more feasible to work on a problem in groups.

2.3.2. Student expectations. Students observe the world, use modern tools (mobile phone, PC, smart home/watch, etc.), and often communicate that they would like to have something similar in their studies. Sometimes the students have interesting ideas that could be implemented in the course. In contrast, at times, students want to learn about big ideas like the deeper functionality of social media, the Internet of Things, autonomous systems, etc. Education needs to take into account that many students want to learn as part of a much larger context.

3. Discussion

By using the Didactical Triangle, we can see that teaching is not the achievement of one individual. There is still possible to have a relationship as a mentor-apprentice, but today’s structure often consists of even more elements than the triangle: colleagues, administration, management, library, IT services, assistants, and others. Teachers have to share experiences to help and motivate each other as well as their students. When cooperating, different people with their ideas, backgrounds, point of view can do much more than individuals on their own.

To increase such synergies, collaboration must be organized. One approach is to develop and implement a system/rules that facilitate knowledge sharing and collaboration among teachers, as well as including students. Shared resources may distribute the work and allow individuals to contribute where they are strongest and find their motivation.

Teaching programming has many possibilities and requirements. The use of modern tools often requires particular knowledge and skills. Students who specifically study programming need different theoretical and practical knowledge and skills compared to students of other majors. These students don’t need the same theory and basic knowledge of algorithmics, problem-solving, and will often benefit from perspectives that introduce what programmed computers can do, not so much how the computers are programmed using advanced programming languages.

4. Conclusions

Real-world examples show that there is no “best solution”. There are different expectations in scope (speed, standards, knowledge of technologies, libraries, programming environments) as well as the area (industry, banking, marketing, education) shows that there are very different expectations, and it is impossible to meet them all in all courses. Teachers require a different approach than computer scientists. Their purpose of education and the challenges they meet in their professional lives differ.

A programming course is a challenge for many students. It should be taught by a competent teacher with knowledge of the subject and pedagogy. It’s easy to alienate students and harder to motivate them if there are problems. In programming, almost every element builds on the previous ones. Lack of mastery of the primary material will give rise to deficiencies in subsequent elements as a result.

References

Berglund, A., & Lister, R. (2010, December). Introductory programming and the didactic triangle. In T. Clear and J. J. Jamer (Eds.), Twelfth Australasian Computing Education Conference (ACE2010), Volume 103, pp. 35–44. Retrieved from https://www.researchgate.net/publication/228848373_Introductory_Programming_and_the_Didactic_Triangle

Blackwell, A. F. (2002). What is programming? In J. Kuljis, L. Baldwin R. Scobble (Eds.). 14th workshop of the Psychology of Programming Interest Group, Brunel University, (pp. 204–218). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.1345&rep=rep1&type=pdf

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking-A guide for teachers. Retrived from https://eprints.soton.ac.uk/424545/1/150818_Computational_Thinking_1_.pdf

Czerepaniak-Walczak M., (2014). Miedzy teorią a praktyką, Funkcje koncepcji pedagogicznych w pracy nauczycieli i nauczycielek, „Refleksje" nr 6, s. 10–14.

Ertmer, P. A., & Newby, T. J. (2013). Behaviorism, cognitivism, constructivism: Comparing critical features from an instructional design perspective. Performance improvement quarterly, 26(2), 43–71. https://doi.org/10.1002/piq.21143

Fojcik, M. K., Fojcik, M., Sande, O., Refvik, K. A., Frantsen, T., & Bye, H. (2020, November). A content analysis of SOLO-levels in different computer programming courses in higher education. In Norsk IKT-konferanse for forskning og utdanning (No. 4). Retrieved from A content analysis of SOLO-levels in different computer programming courses in higher education | Norsk IKT-konferanse for forskning og utdanning (bibsys.no)

Guzdial, M. (2002). Programming Environments for Novices. Retrived from novice-envs2.pdf (umich.edu)

Hartree, D. R. (1950). Automatic Calculating Machines. The Mathematical Gazette, 34(310), 241–252. https://doi.org/10.2307/3611023

Kansanen, P., & Meri, M. (1999). The didactic relation in the teaching-studying-learning process. Didaktik/Fachdidaktik as Science (-s) of the Teaching profession, 2(1), 107–116. Retrieved from https://www.researchgate.net/publication/313645561_The_didactic_relation_in_the_teaching-studying-learning_process

LK20 (2020). Læreplanverket i Kunnskapsløftet. Utdanningsdirektoratet. Retrieved from https://www.udir.no/laring-og-trivsel/lareplanverket/

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33–35. http://dx.doi.org/10.1145/1118178.1118215