"Why Can't I Learn Programming?" The Learning and Teaching Environment of Programming

Zsuzsanna Szalayné Tahy[1] and Zoltán Czirkos[2]

[1] Eötvös Loránd University, Faculty of Informatics
Budapest, Hungary
[2] Budapest University of Technology and Economics, Department of Electron Devices
Budapest, Hungary
http://www.eet.bme.hu/
sztzs@caesar.elte.hu, czirkos@eet.bme.hu

Abstract. This article focuses on teaching a textual programming language as the first programming language (allowing for previously studied visual programming languages). The teaching process is placed into a real educational environment in connection with the national curriculum, social expectations and students reactions. In order to write a program, several abilities and pieces of knowledge are required. There are tools and syllabuses for teaching these skills but the result mainly depends on the personality of the students and teachers. We use the term "Learning Activity Unit" to describe the teaching–learning process of programming and detecting gaps in every day practice. This very simple model is practical for teachers to detect problems. In the global view of teaching programming, the implementation of the curriculum could be analysed.

Keywords: computational thinking, curriculum design, programming, teaching-learning process, Learning Activity Unit.

1 Introduction

Twenty years in practice gives many impressions in teaching informatics. Focusing on textual programming, the main concepts are almost the same, but the tools have changed. The choice of language has changed from Pascal to C-based languages and nowadays the object oriented programming concept is preferred. Several methods have been tested to improve the effectiveness of teaching. At the beginning it was 10-20% of the students who had to learn programming, nowadays this number should approach 100% [3]. The extension of teaching programming skills raises new question: Is everybody able to learn programming?

Teachers have some practice based on other subjects to use special methods in several cases. There are studies on how to teach some skills in a selected, already-known group, or well-known prior knowledge/abilities [1]. However, the digital age is too young to know all the answers.

The Learning Activity Unit is a diagnostic tool used for analysing expectations and individual achievement of knowledge and skills required to develop computational thinking [4]. This article presents some sample cases to demonstrate how this tool can be used in the learning environment analysis. Curricula, syllabuses and lesson plans can be analysed by the Learning Activity Unit concerning aims, methods and timing. This analysis gives a global view of the learning process, guiding the long-term work. The presented collection is far from being complete but the examples describe real problems or contra-productive practices and their recommend solutions.

2 Programming – The Learning Activity Unit Framework

In this section we present the Learning Activity Unit (LAU), a framework whose main objective is to understand the learning process of programming, and to diagnose learning problems. The Learning Activity Unit encompasses the complete learning cycle, from the starting point, when the students first meet a concept of programming, to the point when they are able to use it on their own. The phases defined by the model are as follows:

1. **Initial learning**: As the first step, programs, solutions of programming tasks are written by following the instructions.
   A) **Active** learning: students are motivated to learn, they read sources, listen to lectures. They follow the logic of problem solving.
   M) **Moderated** learning: students make notes; they observe what the teacher does and try to do the same.
   P) **Passive** learning: students are outside observers, they scroll the readings.
2. **Trying** phase: students (or the teacher) explore how the new knowledge or skill could be used.
3. **Experimenting** phase: students have to change the written program, or they have to solve some very similar tasks with the help of teacher.
4. **Pause**: some learned details could be forgotten.
5. **Using**:
   a. **Repeating** means repeating the learned things in phase 1.
   b. **Modifying**: students are able to learn creativity, but only at the original level presented.
   c. **Creating** new programs could give the sense of success.
6. Back to phase 4 or phase 1. It implies a lifelong learning process; however, the process could be broken in every phase.

2.1 The Using of the Learning Activity Unit

The LAU is not homogeneous and not absolute. It is a practical tool to focus on the main points of the learning and teaching process of programming. We apply LAU in a similar way to how programmers work with functions and threads. Every LAU allocates a part of capacity in the human' mind. There are LAUs

running parallel, calling each other and embedded into each other. The LAU Model – with the relationships between LAUs – describes a complex learning process.

There are many cases in the preparatory period of learning textual programming, where we can observe this activity model with the return to phase 1, for example learning a visual language or learning how applications work. If a curriculum is well-structured, these studies develop many important skills and always gives new knowledge to the student. On the other side, there are many courses, programming actions what provide only a foretaste of the knowledge, a short insight, but no more. There are courses focusing on phases 1–3. These courses seem to be very successful, but the outcome is useless.

For illustration only, a humanities educated parent wrote: "My 12-year old daughter is very good in logical tasks. She does programs in Scratch but follows always the trodden path... She attends courses but it seems there is no novelty for her." However, we do not explore those courses, but this opinion shows two problems: 1) This girl prefers 5a **Repeating** activity and courses do not motivate her to choose 5c **Creating**. 2) These courses are only good for wakening up interest for programming, but cannot improve or develop the knowledge to a usable level. Even if courses could develop skills, they are not in connection, they are not structured therefore every course starts from the same basic level.

The LAU seems to be simple enough to use in every day practice, nevertheless, it fits to Bloom's renewed taxonomy and takes into consideration the effectiveness of learning methods as well as the forgetting rates and benefits of linear and spiral curriculum design.

3 Programming Curriculum in Hungary

According to the Hungarian National Curriculum[1], students are required to learn programming. The advanced level secondary school final exam in Informatics includes a task testing the algorithmic and programming skills. The history of informatics education is similar to the Polish system, described in 2015 at the ISSEP conference [5]. The Hungarian National Curriculum was accepted in 2008 but it was renewed in 2012, expanding knowledge expectation compared to the former version. Skill expectations are similar to the new Polish Curriculum but supplemented with topics of hardware and network knowledge (e.g. ISO OSI Model). Unfortunately, the Core Curriculum[2] – prescribed by government – cut the number of lessons to the third compared to the National Curriculum. However, the Hungarian IT sector, the representatives of universities and civil groups (e.g. parents) demand an increase number of lessons of Informatics.

The elimination of informatics lessons has two consequences. On the side of public education, skills and knowledge is to be learnt in only one third of the

[1] Nemzeti alaptanterv (National Curriculum), `http://www.kozlonyok.hu/nkonline/MKPDF/hiteles/MK12066.pdf`, Magyar Közlöny vol 66 (2012) (in Hungarian)

[2] KT 9-12G (Core Curriculum of informatics for grade 9-12) `http://kerettanterv.ofi.hu/03_melleklet_9-12/3.2.16_informat_9-12.doc` (2012) (in Hungarian)

required time. Although this seems to be nonsense, it is written in the certified syllabus[3]. Analysing the syllabus, timing limits teaching to the list of concepts. Students hear (or do not hear) the concepts but there is no time to practice them. Described in LAU terms, this is only part of the **Initial learning** (1), because it is based on informal learning, too. The **Trying** (2) is homework, the **Experimenting** (3) and sometimes the **Using** (5) would be part of other subjects. In many cases, the **Pause** phase is too long, or there is too much time between the next **Initial learning** and previous **Using** phase.

On the other side, many companies from the IT sector try to supplement the programming lessons, to fill in the gaps of public education. But this effort cannot reach the goal because they are not able to ensure long-term development. A 10-hours crash course, or a 30-hours weekend-only courses, maybe a one-week-long camp or a half-year-long course in learning programming gives "a sneak peek". It looks very good, but these courses are not connected to each other, therefore long-term effectiveness is uncertain. In the view of LAU: Phases 1–3 are prepared but phase 4 is too long, phase 5 may never come. The return (loop back to phase 1) will result in random development or backwardness. Moreover, this practice is very dangerous at the point when governmental education management envisions the teaching of informatics as activities of summer camps.

4 Introduction to Textual Programming

4.1 Expected skills and hidden gaps

In order to code a program, one needs almost a dozen skills. Studies of teaching programming explore the role of these skills; describe methods of developing one or a group of skills [6]. Modern educational systems offer opportunities to improve these skills before learning textual programming. Skills and knowledge are mentioned in different ways according to the focus of research. The following skills were found useful to learn before text-based programming: 1) typing, 2) mother tongue based comprehension, 3) basic reading and writing in English, 4) practice in multi window software using, 5) abstraction, 6) logical decision, 7) recognizing and defining data types (boolean, character, integer, real, string), 8) recognition and defining data structures (array, 1D, 2D, 3D, record, graph), 9) object modelling, 10) algorithmic thinking (sequence, alternation, loop) in real word situations, 11) understanding and using functions of applications (e.g. text editor, spreadsheet, animation designer), 12) system (process) modelling.

The list, of course, may not be complete, but the more important aspect is the knowledge level of skills. When one writes a for-loop, six skills are activated from the above mentioned ones, and every, missing skill is a gap. As it is described in [2], "small steps" are very important in effective teaching. The authors of this article analysed books and described a tool for detecting gaps in textbooks, but practically there were no textbooks without big gaps. It seems that

[3] Informatika 10. tanmenet (Syllabus for grade 10) `http://ofi.hu/sites/default/files/attachments/nt_17173_informatika_10.docx` (2016) (in Hungarian)

text-based programming is too complex, and the success of teaching text-based programming depends on how many items are known before using them.

This problem can be observed at the Basics of Programming 1 course[4] of Engineering Information Technology at BME. We conducted an experiment in 2014, asking 3–10 questions from 525 students every week about the topic of the lecture. For example, after the second lecture, 225 students answered this question about variables:

Have you ever heard about variables before this lecture?
1. I haven't ever heard about them, this is new for me.
2. I have heard about them, but I've never tried them.
3. I have seen, I've tried in some cases
4. I have used this knowledge, I am experienced.

The average result of the first test written by students who used variables before the course (ie. who had chosen answers 3. or 4. in the questionnaire) was 72%. On the other hand, those who had chosen answers 1. or 2. only scored 46% in the test. The lack of prior knowledge caused difficulties in their learning progress.

In September 2015 we asked the 565 new-coming students to fill in a 26-item questionnaire about the input skills and knowledge and some question about the learning habits. There were 346 students who answered 77% of the questions on average. We correlated the answers with the test results, created a table of 26 rows (the questions) and 13 columns (the test results) with values between 0.45 and −0.13. Selecting the highest three values from every column (for the tests of different topics during the semester) we get the highest ranking, most correlating questions. This way we found that the most relevant prior skills are:

1. Knowing data structures (12) – What kind of data structures have you used?
2. Knowing Code::Blocks (11) – Have you used Code::Blocks before?
3. Programming (6) – How many points did (could) you get in the secondary school final exam's programming task?
4. Maths knowledge (6) – What mark did you get in the Maths exam?
5. Algorithms (2) – What kind of algorithms have you learnt?
6. Physics knowledge (1) – How would you mark your physics knowledge?

Physics was in connection with the homework, the students' own programmed game. We asked about music, spreadsheet, databases, Nassi-Shneiderman chart, languages, grammar and other topics as well. We can say that the most correlated skills are the most relevant in the course.

We would like to extend our research to the Eötvös Loránd University, Faculty of Informatics. There are also almost 600 students but the courses are in Maths science while BME is the centre of engineering education. The gathered information would be very important to determine programming expectations in secondary schools.

Even though we still have to clear the details, we can say at this moment that the compensating the lack of prior knowledge needs more time. It involves the

[4] Z. Czirkos, G. Nagy: INFOC Portal for course `https://infoc.eet.bme.hu/`

multiple usage of the LAU, and the preparation phase should be longer for successful teaching. The formal courses of Introduction to Computer Science, Basics of Programming or Introduction to <any text-based> Programming Language starts with a huge gap for real beginners.

4.2 De-gap before start

Teaching programming must be preceded by a long preparatory period, when students learn particular competences. This period starts at the beginning of education and the effectiveness depends on the awareness of educators. Many people – students, parents, teachers and experts among them – say, the preparatory period is not for programming. However, as cooking starts with shopping the ingredients, learning textual programming starts with learning the necessary skills. Shopping is much more effective if you know what you want to cook. By the analogy, it would be very useful if primary school teachers would be able to write codes. Not to actually teach programming, but to understand how they should teach the basics.

5 Summary

Textual programming is based on several abilities and skills. The successful learning programming requires the creative usage of basics, therefore text-based programming should be preceded by a designed preparatory period. We described a Learning Activity Unit model based on main concepts of pedagogy to characterize the skill level of computational thinking. Having applied it for analysing Hungarian curricula and courses, we detected problems of effectiveness: only **Initial learning** is planned but sometimes it is also compacted. Our further research will focus on teaching practice of programming in classroom.

References

1. Heintz, F., Mannila, L., Nygårds, K., Parnes, P., Regnell, B.: Computing at School in Sweden – Experiences from Introducing Computer Science within Existing Subjects In: A. Brodnik, J. Vahrenhold,(eds.) ISSEP 2015. LNCS 9378, pp. 118–130. Springer, Heidelberg (2015)
2. Hofoku, Y. Cho, S., Nishida, T. and Kanemune, S.: Why is programming difficult? - Proposal for learning programming in "Small Steps" and a prototype tool for detecting "gaps" In: ISSEP 2013, pp. 13–14 Universitätsverlag Potsdam, 2013.
3. Informatics Europe & ACM Europe Working Group on Informatics Education: Informatics education: Europe cannot afford to miss the boat, (2013)
4. Lee, I. et al.: Computational Thinking Resources, `hhttps://csta.acm.org/Curriculum/sub/CompThinking.html` [Accessed on 21-07-2016] (2011)
5. Syslo, M. M., Kwiatkowska, A. B.: Introducing a New Computer Science Curriculum for All School Levels in Poland In: Brodnik, A., Vahrenhold J.,(eds.) ISSEP 2015. LNCS 9378, pp. 141–154. Springer, Heidelberg (2015)
6. Szlávi, P. and Zsakó, L.: Methods of teaching programming 1(2). In Teaching Mathematics and Computer Science, 1.02, pp. 247-258. Univ. of Debrecen, Hungary (2003)