Computer Science Education with a Computer in the Background

Maciej M. Sysło1[0000-0002-2940-8400]

1 Warsaw School of Computer Science, Warsaw, Poland
syslo@ii.uni.wroc.pl

Abstract. The original premise of the unplugged approach was to introduce students to the concepts of computer science (CS) in a way that does not require access to computers, in particular for programming. It is difficult to fully maintain this approach today, when almost all schools and all students are equipped with digital equipment. The Bebras challenge is another initiative addressed to students of all ages in K-12 in which it was originally assumed that students have no prior knowledge of CS. The new CS curriculum was introduced in Poland in 2017/2019 and since then we witness a variety of approaches taken by teachers and schools to meet the curriculum requirements. In this paper we present an idea of teaching and learning CS with computers which are in the background and the use of them depends on a particular situation and student's decisions. We consider this approach as an extension of the unplugged approach. Four groups of such activities are distinguished: (1) classical unplugged with a computer in the background, (2) problem situations for which a computer is only a medium, (3) educational robotics, and (4) designing solutions to problems outside computers before using them. We shortly characterize these groups and comment on their use in developing computational thinking and assessment.

Keywords: Unplugged, Computational Thinking, Curriculum

1 Introduction

We believe that the selected approaches to the development of computational thinking (CT), programming skills and learning about computer science (CS) can bring the expected results, as the authors of the papers assume. In our case, we look for an approach that will guarantee the achievements of all students as provided for in the CS core curriculum. Contrary to most research results conducted on selected groups of students from a fixed school level, we are interested in implementing the spiral development of all students throughout the years of their stay in school, i.e. in K-12. It follows from this premise that we cannot limit ourselves to a fixed approach or fixed tools – teachers and especially students should be free to choose.

Computer science (Informatics) education has a long history in Poland. In this history, you can find elements corresponding to today's unplugged and CT approaches that have been used and developed for a long time without being specifically named as they are today.

J.-P. Pellet and G. Parriaux (Eds.): ISSEP 2023 Local Proceedings, pp. 89–101, 2023.
https://doi.org/10.5281/zenodo.8431926

We will focus here on teaching CS and the presence or absence of a computer and its applications in this process. We will justify our approach extending the unplugged approach to CS with a computer in the background in a sense that a computer could be in a reach of students and they can use it when they (or teacher) decide that it can help them to learn better.

Today when all students have an easy access to technology, smartphones in their pockets, tablets and computers in school computer labs, it is difficult to convince students to CS classes with no access to technology.

2 CS education in Poland

2.1 Early History of CS education in Poland

The first regular lessons related to "computers" were held in Poland in two HS in Wrocław in the second half of the 1960' when the terms "computer" and "informatics" did not have counterparts in Polish and a computer was a "mathematical machine". The school subject was called „Programming and using a computer". Since those days a computer was mainly used for numerical calculations, students in this first informatics classes learnt some basic numerical methods for solving mathematical problems and programming languages (assembler, Algol 60). They ran their programs on the real mainframe Elliott 803 located at the University (Sysło, 2014a).

The official history of informatics (computer science in Polish) in Polish schools started in 1985 with the first official informatics curriculum for the school subject called "Elements of Informatics" proposed by the Polish Information Processing Society and approved by The Ministry of National Education. The curriculum covered the topics related to the use of microcomputer applications (for text editing, creating graphics and sounds, building tables and simple databases, making simulations) and also elements of algorithmics and structural programming using Logo, mainly for drawing pictures and operations on lists of characters (Sysło, 2014a).

In Poland, we are very proud that algorithmics and programming in informatics education, introduced to the curriculum in 1985, have remained in the national core curriculum for all these years until today.

2.2 Computational thinking in the CS curriculum of 1997

From 1997 for the next 15 years in Poland, all national core curriculum on CS supported Denning's opinion that: *Computational thinking has a long history within computer science. Known in the 1950s and 1960s as "algorithmic thinking," it means a mental orientation to formulating problems as conversions of some input to an output and looking for algorithms to perform the conversions* (Denning, 2009).

In the curriculum for HS approved in 1997, in the section "Algorithmics and programming" one can read that the school is to provide conditions for students to acquire the following competences, called *algorithmic thinking*:

- Define a problem situation, including data [*abstraction*], the goal and the results.

- Formulate a plan for solving the problem – separate sub-problems [*decomposition*] and indicate connections between them.
- Choose a way to solve the problem:
  - design an algorithm [*algorithmic thinking*].
  - use an existing program or program a solution method in a selected programming language [*implementation, programming*].
- Analyze the correctness of the algorithm and its implementation [*debugging*], and assess its complexity [*evaluation*], test the program [*testing*].
- Complex projects solve in a team [*collaboration*].
- Choose and solve problems from various school subjects [*generalization*].

The above list of competencies resembles the operational definition of CT (Barr et al., 2011). Additionally, we have inserted into the text above some mental tools of CT (in italic) that constitute another definition of CT. Thus, CT as algorithmic thinking has a long tradition in our CS education. In the years that followed, these curriculum statements slightly reformulated were addressed to all school levels.

2.3 The new CS curriculum

In the last 20 years several countries began introducing CS for all students with CT as a main capability. We in Poland continue our efforts to address CS to all students in K-12 with algorithmic thinking as the main approach which, as illustrated above, is another formulation of the operational definition of CT.

The new core curriculum of CS has been introduced to K-8 in September 2017 and to HS, including vocational schools, in September 2019. It benefits very much from our experience in teaching informatics in schools for more than 30 years (Sysło, Kwiatkowska, 2015).

The new curriculum consists of **Unified aims**, which define five knowledge areas in the form of general requirements, they are the same for all school levels. The most important are the first two aims and their order in the curricula: (I) **Understanding and analysis** of problems based on logical and abstract thinking, algorithmic thinking, and information representations; (II) **Programming and problem solving by using computers** and other digital devices – designing algorithms and programs, organizing, searching and sharing information, using computer applications. The content of each aim, defined adequately to the school level, consists of detailed **Attainment targets**. Thus, learning objectives are defined that identify the specific informatics concepts and skills students should learn and achieve in a spiral fashion through the four levels of their education (grades 1-3, 4-6, 7-8, HS 9-12). At each level the implementation of the curriculum varies across three elements – the first element is more important at lower levels and elements 2 and 3 become more important during progression: (1) problem situations, cooperative games, and puzzles that use concrete meaningful objects – discovering concepts, heuristics; (2) computational thinking about the objects and concepts – algorithms, solutions; (3) programming, moving from visual/block to text-based environment, including program testing and debugging. For benefits of such a spiral curriculum see (Webb et al., 2017).

2.4 Computational thinking

As a conclusion to the history of our way to CS4ALL in Poland, where CT appears to be operationally defined and consisting of some mental tools used in the process of solving problems, we avoid to use the terms “CT education”, “teaching CT”, “CT classes” and similar, as used by many authors. CT is an approach and a collection of mental tools used in problem solving as a byproduct in learning CS concepts and methods (algorithms). Therefore, the following definition of CT fits our approach (Wing, 2014): *Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer – human or machine – can effectively carry out.*

We propose not to directly teach CT, but teach how to discover, develop, and use CT in solving problems from various areas of education, especially in CS. Similarly, as we suggest not to “teach Scratch” but to “teach programming using Scratch”.

The introduction of CT to education, along with Jannette Wing in 2006, is also attributed to Seymour Papert in connection with his idea of constructionist learning (Papert, 1980) focused on stimulating students to reflective thinking. The most appealing to us is the saying of Papert from 1970 (Papert, 1970) that: *children learn by doing and thinking about what they do*. Therefore, treating CT as a problem-solving strategy that involves the use of CT-related mental tools in the process, we add also a constructionist viewpoint and expect students’ reflective thinking.

3 CS education with a computer in the background

Computer Science Unplugged (CS Unplugged) has been defined as: *a collection of activities and ideas to engage a variety of audiences with great ideas from computer science, without having to learn programming or even use a digital device*. It originated in 1990’ as an outreach program to engage school students *to help them understand what computer science might involve other than programming*. Then some unplugged activities have been described and published in several languages (see csunplugged.org) and they are widely used in lessons and also in research. The approach is mentioned in textbooks and web services on teaching CS and appears also in recommendations for national and school curricula. However, the available content *was still intended as enrichment and extension exercises, and did not assume that computer science would be part of the curriculum* (all citations from Bell, Vahrenhold, 2018).

From pedagogical point of view, unplugged approach is based on constructivism and partly on constructionism: students construct their own knowledge, sometimes producing also certain artifacts, by utilizing what they have already learnt, using some mental tools, and engaging with problem situations to be solved or questions to be answered. This process of learning leads them to understand important concepts, principles, and mechanisms, mainly of computing nature (Relkin, Strawhacker, 2021).

Looking back at the history of CS education in Poland, briefly described in Sec.2 (Sysło, 2014a), algorithmics plus programming and problem solving using algorithmic thinking were closely related in the 1965 and 1985 curricula for schools as well as in all national core curricula after 1997. In 1980’ when regular CS lessons entered schools, students had a long way (in distance and time) to a computer, therefore they had to spent a lot of time writing their programs on paper before the programs reached a computer. Also teachers were explaining CS concepts and algorithms using traditional tools. It was time of unplugged introduction to CS and preparation for programming. I remember HS classes coming with their teachers to our Institute for CS lessons (the Institute was quite well equipped in computers) – the students spent first hour in a classroom developing their algorithms and programs and then spent one hour in a computer lab uploading, running, testing, and debugging their programs.

Never in the past or in recent years have we referred to classes as unplugged or plugged-in, these CS teaching and learning phases have been naturally intertwined and integrated. Today it is difficult to maintain an unplugged approach when almost all schools are fairly well equipped with digital equipment. Moreover, it is reported that *unplugged activities are effective when used in a context where they will be ultimately linked to implementation on a digital device, either through programming, or by helping students to see where these ideas impinge on their daily life* (Bell, Lodi 2019). Unplugged activities may play a role of introduction to using CT tools. In particular, combining both unplugged and plugged-in activities may help students to better comprehend programming concepts and constructions such as variables, loops, conditionals, and events, which are shared by CT, programming, and CS in general.

Understanding the unplugged approach as an introduction to CS without using a computer, mainly so as not to program it, we extend here the range of unplugged, to teaching and learning environments with **a computer in the background**, in which the computer (and other IT technologies) is in the background of learning activities, closer or further, more or less integrated, but not as the technology used in learning to program, although in the process of CS problem solving including programming.

Almost every CS concept can be introduced to students without using a computer. However, since we focus on rigorous CS education, we propose to use the approach with a computer in the background very flexibly. Ultimately, it is the teacher who decides about the role of computers in his classes, but leaving students the choice so that they have an opportunity to develop also their ability to make decisions about the use and the role of computers and other technologies in the problem-solving process.

We distinguish four types of environments in which a computer has its place in the background, in a certain sense. In the rest of this chapter we focus our attention on these environments and comment how they can be used in learning and teaching to reaching the goals of CS education including – the most important – CT skills.

- classical unplugged, eventually with some computer puzzles
- Bebras tasks
- educational robotics
- algorithmics and programming unplugged.

One may thing also about other types of environments which are combination of different tools, mechanical and electronic calculating machines, games, computer games etc. which can be used to introduce students to fundamental concepts of computing. We use such environments at a children's university (Sysło, Kwiatkowska, 2014b). Our approach contributes to constructionist learning, to learning by doing and making meaningful objects in the real world, computational models of real-world situations. Our learning environments are extensions of classical unplugged ones by encouraging children to purposely and properly use computers for certain activities.

3.1 Classical unplugged

By classical unplugged activities we mean the activities originally proposed by Mike Fellows and Tim Bell and the other activities of similar type used to engage young students with basic ideas and algorithms from computing and problem solving, but without using a computer or another digital device. Since usually there are many digital devices in the classrooms (tablets, smartphones), we have created a package of 25 modules with simple applications that can be used in many ways by the youngest students, hence its name: *Informatics for Kids* – **I4K** (pl. *Informatyka dla Smyk*). The applications are mostly related to CS education, but they can also be useful in classes related to almost any other education: mathematics, natural sciences, languages, art, etc. Some modules are linked to the Bebras tasks or the code.org puzzles.

Almost all activities proposed in this package, intended to be carried out on a computer or tablet, can be transferred to situations arranged outside the computer with appropriately prepared materials (cards, templates etc.). Then such classes take the form of the classical unplugged – a group activity, providing kids with additional impressions, cooperation skills and reflection.

Behind the package there is the idea of Jean Piaget's constructivism, according to which the kids build their knowledge on the basis of what they already know and the experience gained while performing various exercises. This idea of *learning by doing*, which has its roots in progressivism at the turn of the 19th and 20th centuries, was extended at the end of the 20th century by Seymour Papert to constructionism, placing additional emphasis on artifacts (also on a screen) that are the product of learners. It is well characterized by Papert's words that: *children learn by doing and thinking about what they do* (Papert, 1970). Currently, the thinking accompanying children's educational activities is well defined by mental tools that make up CT.

3.2 Bebras tasks

The Bebras Challenge consists in solving a certain number of tasks (called Bebras tasks). Most of the tasks are in the form of illustrated stories that describe certain "real" problem situations. The tasks are related to concepts, issues or methods in CS, usually indirectly, hidden in the stories. The Challenge is an opportunity for students to discover CS concepts and methods (algorithms) by solving short tasks that promote CT (Dagienė et al. 2019). They have about 3 minutes to solve a task: to choose a right multiple-choice entry, write an answer (usually a string of characters) in an open window or interact with a part of the task formulation to complete its solution. A computer is only a medium for presenting the tasks and is used to create and save task solutions. The Bebras tasks may be also used in a full unplugged fashion, printed or arranged on the floor, far from computers.

In Poland, the Challenge is run by a computer system, client-server type – each decision of a student (client) taken at his school computer is recorded on the server. After a challenge, we issue augmented versions of all tasks which contain an additional section consisting of: a correct solution and its development, and comments that are extended version of the original task section “It’s informatics”. The comments are addressed to both, students and to teachers.

In the beginning of the Challenge, it was assumed that the Bebras tasks could be solved without any previous knowledge of CS or programming. On any level of school education, students were not supposed to demonstrate any CS knowledge, but possibly the ability to solve tasks using mental tools of CT. After almost 20 years of the Challenge which have been accompanied by many national initiatives aimed at introducing CS for all students at all education stages, the role of the Bebras tasks should be reconsidered and possibly reviewed. One hour of a challenge a year, usually taken by only some students on only a selection of concepts, topics and tools, is not able to make a significant impact on CS education of all students in general.

Reviewing the pertinent references we could not find any evidence that the Bebras tasks are used beyond the challenge and integrated with regular CS lessons and learning strategies in a class, except assessment, see Lonati (2020). On the way to overcome this situation, we build a repository of Bebras tasks as a collection of individual tasks in both versions, competition and with explanations, used in the Challenge in Poland. The tasks are tagged with CS concepts and CT mental tools used in the tasks (see (Dagienė et al. 2020; Datzko, 2021) for a classification of Bebras tasks). A teacher can choose one or more tasks from the repository by setting the stage of the Challenge and selecting key words characterizing the tasks with CS concepts and CT tools. From selected tasks, a teacher can create a mini-challenge for a class, which can be used in several ways, as a warm-up preparing or introducing students to a lesson topic, as a test how students are prepared for a lesson, or as a test assessing students' knowledge and skills in the range of CS concepts and CT skills at the end of a lesson.

The repository allows easy access to tasks to learn how to solve them. There is no other way to learn than to practice with such tasks – this is our answer to teachers, students and their parents when they ask: How to prepare students for the Challenge.

The idea underlying the Bebras Challenge as a way to introduce students to CS, can be extended on professional development of teachers. This may apply to all teachers who do not have a full ICT/CS education as required by the curriculum. We focus our attention on primary education teachers (grades K-3), who are graduates of pedagogical faculties and usually have contact only with ICT classes.

The Bebras tasks can also be used as measures to assess students' overall development and ability to transfer acquired CT skills while solving problems that, by the nature of these tasks, relate to real problem situations (Román-González et al., 2019). A special moment for such an assessment may be the end of a certain educational stage, for example at the end of primary education K-3 what is very important for a successful spiral development of students. Again, the repository of Bebras tasks may be very useful to properly arrange tests according to expected knowledge and skill of students.

3.3 Educational robotics

Learning with physical robots, such as Dash&Dot, Ozobot, Genibot can be seen as a continuation of the kinesthetic activities from the first group of activities, when for example a robot is supposed to imitate the movements of children or vice versa, on the floor or on the screen. Moreover, physical manipulation of objects promotes children's' constructionist learning through the development of mental representations of the objects. Solving various tasks and problems they create, build, evaluate, and revise their constructions and concepts which are to meet their expectations and goals. Robotics also encourages students to analyze real world problems, think creatively, and apply CT tools in the process of proposing solutions to such problems (Bers, 2008), (Grover, 2011), (Chevalier et al., 2020).

Classes with robots can also play a role of introduction to programming when students turn on robots and control their moves to achieve certain goals with the help of programs made in a language characteristic for given types of robots. In such classes students have opportunity to learn that robots can understand they own language to communicate with them: graphical collection of interactive instructions (Dash&Dot), colors (Ozobot), cards (Genibot), and Blockly (Dash&Dot, Ozobot, Genibot).

Although playing with a robot is unplugged to some extent, almost every robot contains a "mechanism" to control its behavior. Watching the youngest children playing with robots, treating robots as programmable devices goes to the background of their attention, they are mainly interest in the behavior of the robots they want to achieve. Thanks to this, it is quite easy to associate the types of robot moves with concepts that have a broader meaning, such as moving in different directions or distances, repeating selected moves a certain number of times, or performing certain moves depending on the situation encountered by robots. From such learning with robots it is quite close to a more formal approach to programming concepts in general.

A special type of lessons with robots are concerned with controlling them on a computer screen. Such children activities are important to implement the statement in our curriculum for K-3 which reads: "A student [...] programs sequences of instructions which control an object on the screen of a computer or other digital devices". An excellent environment for this type of activities are puzzles in the Hour of Code initiative (https://code.org/learn), which is very popular in Poland – in 2018 there were more than 650 M students registered to code.org from Poland. Such a popularity is due to many thoughtful solutions such as: (1) the heroes of the puzzles are characters known to students from their favorite stories, comics and games; here they can interact with them; (2) puzzles are in sets of increasing difficulty; (3) the solutions of puzzles consist in arranging a program in a block-based language to pre-prepared scenarios; (4) the students can run, debug and improve solutions many times; (5) they can also view the Java Script code corresponding to the block-based solution. Although there is no direct connection of the code.org activities with CT concepts, solving such puzzles arranged in courses which correspond to particular algorithmic and programming constructions, students apply abstraction and pattern matching, then decomposition and finally algorithms in solving puzzles. Moreover, using event blocks students can program interaction what is a quite advanced CS topic.

3.4 Algorithmics and programming unplugged

Modeling, designing and solving problem situations outside the computer as a step preceding the computer solution – in unplugged fashion – has a history as long as CS in professional and educational environments. In 1950' till even in 1980', for a programmer or a student there was a long way (in distance and time) to a computer, therefore they spent a lot of time on writing their programs on paper before they were run on a computer. I remember when students' programs brought to a computer, run successfully without any corrections – I don't think it happens today, now they sit at a computer until their programs run correct.

Caeli and Yadav (2020) in they view on historical development of CS emphasize the importance of combining plugged and unplugged activities, as means to fully understand and take advantage of the power of computing. Unplugged activities can be very efficient in understanding the concepts and methods behind a problem to be solved and computer tools to be used.

Skills of programming are not needed to develop an algorithm for a problem, although programming a solution is needed to fully experience limitations when implementing the algorithm. On the other hand however, after a few first lessons on programming with a properly chosen algorithms to be implemented, any next lesson on creating and implementing an algorithmic solution to a problem cannot be naturally split into unplugged and plugged parts – students working on an algorithmic solutions quite often use programming constructions they have already learnt to describe algorithmic constructions. Finally, a description of an algorithm, even on paper, takes a pseudo programming language form, which can be considered as a result of not only combining plugged and unplugged activities but as an integration of both approaches.

The first informatics textbook *Elements of informatics* (in Polish) for high schools appeared 1989 and contained two chapters on algorithmics and programming. The chapter "From a problem to a program – elements of programming in Pascal" leads from formulating a problem situation to a program in Pascal and the chapter "Calculate faster – the efficiency of algorithms" deals with practical efficiency and theoretical optimality of some searching and sorting algorithms. In 1997, the author published the book *Algorithms* (in Polish) "for those who are interested in learning how to create algorithms and using them to solve problems" and in 1998 the book on algorithms was accompanied by *Pyramids, cones and other algorithmic constructions* (in Polish), which consists of 15 short chapters on various problem situations treated in an unplugged manner for developing some algorithmic topics and techniques, see Table 1.

Each problem situation in the *Pyramids* can be first discussed, analyzed and solved to some extent far from a computer. Popular examples are: social games, short codes, change making, etc. The book contains also a chapter on the stable marriage problem which has a much longer history in the author' teaching using unplugged approach. In a class on algorithmics in the early 1970', the author has decided to introduce the Gale and Shepley's algorithm for creating stable marriages to a group of students (the same number of boys and girls). First, the students created lists of preferences in the other sex group and then they started to perform the algorithm (which is a kind of greedy method) interchangeably choosing in the other group and revising their choices when refused in the other group. Finally a class concluded writing a computer program which in that time was run in the batch mode. The author was able to see the benefits of the applied approach – unplugged – after 20 years, when he met one of the students and he remembered exactly how the algorithm "run" on the living organism of students – he was able to repeat it. I doubt whether he would be able to reproduce a program written for this algorithm. Today, when computers are at hand and everywhere and I still recommend this algorithm to be performed in a group of students before they start programming it.

Table 1. Contents of the book *Pyramids*... Each chapter is characterized by CS topics it deals with and CT tools applied in solving the related problems.

Chapters	**CS topics, CT tools**
Add a pinch of salt to taste – are recipes algorithms	*CS topics*: precision of algorithmic steps *CT tools*: approximation, uniqueness, cook versus computer
How the pyramids were built	*CS topics*: calculations *CT tools*: algorithm
Social games	*CS topics*: who is the idol? leader election. *CT tools*: reduction by elimination
The efficiency of Russian peasants in multiplication – how to simplify your life	*CS topics*: binary system, fast multiplication *CT tools*: multiplication by decomposition
Recursion – how to use what we know, how to "dump the work" to a computer	*CS topics*: generating consecutive digits of a number *CT tools*: recursion, positional representation of numbers
Fibonacci numbers – how to be perfect	*CS topics*: Fibonacci numbers in science *CT tools*: recursive thinking, fast calculations
Filling vessels using the Euclid algorithm	*CS topics*: Euclid algorithm *CT tools*: geometric interpretation, diophantine equation
Prime numbers and composite numbers	*CS topics*: prime and composite numbers *CT tools*: algorithm, testing whether a number is prime
Clock arithmetic – benefits of residuals	*CS topics*: modular arithmetic *CT tools*: fast calculations on large numbers
Searching in ordered and unordered sets – about the benefits of taking care of order	*CS topics*: searching in ordered sets *CT tools*: binary search, divide and conquer
Finding stable relationships – dancing couples, marriages	*CS topics*: stable matching *CT tools*: greedy strategy
Do we always gain from greediness?	*CS topics*: the change making problem, leaving the maze *CT tools*: greedy algorithm
Small trees – fast vending machines and short codes	*CS topics*: Huffman compression, fast vending machines *CT tools*: greedy approach, trees
Backtracking search	*CS topics*: the queens problem, leaving the maze *CT tools*: backtracking, brute force
Dynamic programming	*CS topics*: dynamic programming

	CT tools: optimization by dynamic programming

The topics discussed in *Pyramids…* are introduced there in an informal way, omitting theoretical arguments. Practical applications and examples help the reader to solve some of the tasks in the book, which may be considered as a test of comprehension.

This book can be used by teachers as a demonstration of **pedagogical content knowledge** (PCK) that subject knowledge and teaching methods cannot be considered independently (Shulman, 1986). PCK combines the knowledge of the subject with pedagogy and the practice of teaching it. PCK is (Shulman): "The ways of representing and formulating the subject that make it comprehensible to others", to students and also to teachers when they first approach new topics they are going to teach.

3.5 Conclusions

Activities as puzzles appear in all the above groups. They are important "tools" for algorithmic thinking, accompanied by other CT tools. In particular (Levitin, 2005): (1) puzzles lead to thinking about algorithms on a more abstract level not directly related to programming; (2) strategies of solving puzzles are always special instances of general problem-solving techniques which might be useful in other domains; (3) solving puzzles helps to develop creativity; (4) puzzles are usually very attractive for students more than regular lesson assignments, making them working harder.

The approach to developing CT skills presented in this paper is a proposal to integrate unplugged activities and coding without any restrictions when using one or the other in the spiral development of computing skills and CT. Decisions are in the hands of teachers and students who should be able to choose the best way of learning for them. There is no dichotomy of unplugged or programming, unplugged should be integrated with the process of learning programming and CS concepts in general..

Activities of students outside a computer are offered today to the youngest adepts of CS, but in the past they have also accompanied specialists in CS, especially in times when computers were located in remote and isolated places. Activities in classes without a computer or with a computer in the background have also broader goals of developing the ability to select tools (hardware or/and software) as a decision in the process of designing a way to solve a problem. In some cases, it may turn out that a computer is not needed at all, for example, when certain calculations can be done by hand, and when we decide to use a computer – the solution can be created in a ready-made application without the need to create our own program.

How different is the role of computers in the activities discussed in this chapter. In the first case, computers are really in the background. In the Bebras Challenge, computers are necessary, but they are only a medium for conducting the challenge. Then, in playing with robots, a computer may appear either as a processor built into such devices or as a robot control device, often requiring programming. Finally, in the last type of activities, the computer waits for a prepared student to make proper use of it.

All these four types of activities have one thing in common – they are addressed to all students, including also those who do not think about connecting their professional future with CS. Therefore, they are to bring them closer to CS using various methods and from different points of view, with or away from the computer, to varying degrees of depth. As a result, they are to tear off the secrecy from CS solutions and bring closer the laws and mechanisms of their functioning. Knowledge of these mechanisms can be useful even to a non-specialist to understand their operation, and sometimes even modify them for their own purposes.

References

1. Barr D., Harrison J. and Conery L. (2011), Computational Thinking: A Digital Age Skill for Everyone. Learning and Leading with Technology, 38, 20–23.
2. Bell T., Vahrenhold J. (2018), CS Unplugged – How is it used, and does it work? in: Böckenhauer H.-J., Komm D., Unger W. (eds.), Adventures between lower bounds and higher altitudes. Springer, New York: 497–521.
3. Bell T., Lodi M. (2019), Constructing Computational Thinking Without Using Computers, Constructivist Foundations 3/14, 342-359.
4. Bers M.U. (2008), Blocks to robots, Learning with Technology in the Early Childhood Classroom, Teachers College, Columbia University, New York
5. Caeli E.N., Yadav A. (2020), Unplugged Approaches to Computational Thinking: a Historical Perspective, *TechTrends*, nr 6/2020.
6. Chevalier, M., Giang, C., Piatti, A., & Mondada, F. (2020), Fostering computational thinking through educational robotics: A model for creative computational problem-solving. International Journal of STEM Education, 7, 41.
7. Dagienė V., Futschek G., Stupuriene G. (2019), Creativity in solving short tasks for learning computational thinking, Constructivist Foundation 14, 3, 382-415.
8. Dagienė V., Hromkovic J., Lacher R. (2020), A two-dimensional classification model for the Bebras tasks on informatics based simultaneously on subfields and competencies, ISSEP 2020.
9. Datzko Ch. (2021), A multi-dimensional approach to categorize Bebras tasks, ISSEP 2021.
10. Denning P.J. (2009), Beyond Computational Thinking, CACM 52, 6, 28-30.
11. Grover S. (2011),Robotics and Engineering for Middle and High School Students to Develop Computational Thinking, Annual Meeting of the American Educational Research Association, New Orleans
12. Levitin A. (2005), Analyze That: Puzzles and Analysis of Algorithms, SIGCSE'05, ACM.
13. Lonati V. (2020), Getting Inspired by Bebras Tasks. How Italian Teachers Elaborate on Computing Topics, Informatics in Education, Vol. 19, No. 4, 669–699.
14. Papert S. (1970), Teaching Children Thinking, WCCE, IFIPS, Amsterdam.
15. Papert S. (1980), Mindstorms. Children, Computers, and Powerful Ideas, Basic Books.
16. Relkin E., Strawhacker A. (2021), Unplugged Learning: Recognizing Computational Thinking in Everyday Life, in: Bers M. (ed.), Teaching Computational Thinking and Coding to Young Children, IGI Global, 41-62.
17. Román-González M., Moreno-León J., Robles G. (2019), Combining assessment tools for a comprehensive evaluation of computational thinking interventions. in: Kong S.C., Abelson H. (eds.),Computational thinking education, Springer, 79–98.
18. Shulman L.S. (1986), Those who understand: Knowledge growth in teaching, Educational Researcher, 2/15, 4–14
19. Sysło M.M. (2014a), The First 25 Years of Computers in Education in Poland: 1965 – 1990, in:Tatnall A., Davey B. (eds.), History of Computers in Education, IFIP AICT 424.
20. Sysło M.M., Kwiatkowska A.B. (2014b), Playing with Computing at a Children's University, WiPSCE '14, Berlin, Germany, 104-107.

21. Sysło, M.M., Kwiatkowska, A.B. (2015), Introducing a new computer science curriculum for all school levels in Poland, ISSEP 2015.
22. Webb M. et al. (2017), Computer Science in the School Curriculum: Issues and Challenges, in: Tatnall A., Webb M. (eds.), WCCE 2017, IFIP AICT 515, 421–431.
23. Wing J. (2014), Computational Thinking Benefits Society, http://socialissues.cs.toronto.edu/index.html%3Fp=279.html.