Designing Informatics Curriculum for K-12 Education: From Concepts to Implementations

Author(s):
Dagienė, Valentina; Hromkovič, Juraj; Lacher, Regula

Designing Informatics Curriculum for K-12 Education: From Concepts to Implementations

Valentina DAGIENĖ[1], Juraj HROMKOVIČ[2], Regula LACHER[2]

[1]*Vilnius University, Lithuania*
[2]*ETH Zürich, Switzerland*
e-mail: valentina.dagiene@mif.vu.lt, juraj.hromkovic@inf.ethz.ch, regula.lacher@inf.ethz.ch

Abstract. Computing as a discipline has common roots with mathematics and written languages, and computing as a way of thinking and handling has been integral to human culture since ever. This is not only a reasonable argument for convincing society to consider informatics as one of the very fundamental pillars of education, but it also puts the potential contributions of teaching informatics in schools into the correct perspective in the context of science and humanities. Many European countries are switching from teaching information technologies to informatics education during the current second decade of this century. Informatics curriculum is becoming a central part of school education.

We explain and design a way of developing informatics curriculum that offer the critical competences new generations need to survive and thrive in todays' knowledge society and will allow them to contribute to the future development of society. These competences also strongly support the development of their intellectual potential and creativity. Our design of informatics curriculum takes into account the interaction with other scientific disciplines as well with the subject didactics, pedagogy and psychology.

The starting point is merging constructionism and critical thinking. Constructionism with its "learning by doing" and "learning by getting things to work" enables designing a teaching process in which students acquire knowledge by creating products, analysing the properties and the functionality of their own products, and finally derive motivation to improve these products. Critical thinking asks us not to teach products of science and technology and their application, but to teach the creative process of their development. To implement this approach, we use the historical method allowing the students to learn by productive failures in the process of searching for a solution. To organize the process of learning and make the different steps available to the appropriate age groups we take into account the cognitive dimensions of the revised taxonomy of Bloom. To illustrate how the combination of all these concepts works we present a detailed curriculum for algorithm design, programming, robotics, and communication in networks.

Keywords: informatics, informatics curriculum, informatics concepts, informatics education, computing education, computer science, computational thinking, digital competences, Bloom's taxonomy.

1. Introduction

Technological developments influence education at schools and enhance opportunities for effective learning. Importance and roles of informatics / computer science / computing in the school education are growing and are broadly recognized. Arguments for including informatics education in schools are provided and discussed, for example, Don Passey (2019) presents the six main arguments for wider-scale introduction of the informatics subject, the implications for researchers, schools, teachers and learners, and evidence of outcomes of informatics in compulsory school education. Also Mark Guzdial made a good list of the arguments in his book "Learner-Centered Design of Computing Education" (Guzdial, 2015).

Informatics curriculum tend to be defined nationally or at state level (CECE-Report, 2017). The curriculum is important for schools and especially for education policy makers, and it needs to be matched by well-qualified teachers who deliver topics that resonate with pupils, teachers who motivate them, stimulate their deeper thinking skills, and attract their curiosity to continue the course further. It is important to note

Table 1

Terminology used in this paper

Term	Definition
Informatics	The entire set of scientific concepts that make information technology possible. Informatics is a distinct science, characterised by its own concepts, methods, body of knowledge, and open issues. (CECE-Report, 2017)
Computer Science	The scientific discipline encompassing principles such as algorithms, data structures, programming, systems architecture, design, problem solving, etc.
Computing	The broad subject area incorporating information technology, computer science, digital literacy and problem solving in this context deploying computational thinking. Computing is now the title for the new curriculum in the UK; in Australia and New Zealand "Digital Technologies" is the equivalent term used in curricula.
Information Technology (IT)	The use of computers, in industry, commerce, the arts and elsewhere, including using software packages, aspects of information technology systems architecture, human factors, project management etc.
Digital literacy	The general ability to use computersa – covers fluency with computer tools and the internet. It is a set of skills rather than a subject in its own right.
Computational thinking	Recognising aspects of computation in the world that surrounds us and applying tools and techniques from computer science to understand, reason and solve problems in relation to both natural and artificial systems and processes.
Programming	A process of designing and building an executable computer program to accomplish a specific computing result or to perform a specific task. It involves: analysis and understanding of problems, identifying and evaluating possible solutions, generating algorithms, implementing solutions in the code of a particular programming language, testing and debugging.
Coding	Coding is the translation of natural language into machine commands and coders use an intermediary language to direct the step-by-step action the machine needs to take.

that even a very well-established informatics curriculum cannot reach its goals without well-educated and trained teachers who need to have higher education in informatics as a scientific discipline. Many countries, for example, Finland or Norway, have adopted the integrated approach in primary education: computational thinking should be included in all subjects, from history to biology and arts. The integrated approach also requires that all primary teachers will be familiar with informatics concepts. The worldwide tendencies of teaching informatics in primary education and primary school teachers' understanding of computational thinking issues are provided in a survey of 52 countries (Dagienė *et al.*, 2021).

The variation in terminology in relation to computer science / informatics / computing education / computational thinking or even information and communication technology (ICT) has been a source of much confusion, so we begin by defining the terms used in the paper. The CECE-Report (2017) provided some useful definitions, these form the basis for definitions in this paper with some further clarification (see Table 1). Informatics is slightly broader than computer science, a term which is used widely across Europe. We focus on the primary and secondary school levels (aged approximately 4 to 19), excluding schools within tertiary education.

In this paper we are going to answer the following questions:

1. What to take into account when designing an informatics curriculum?
2. How to design an informatics curriculum interacting with other school subjects and being well adjusted to corresponding age groups?
3. How to motivate pupils to approach deep informatics concepts and to guarantee a high degree of success?

2. Background

Numeracy and literacy are fundamental to any educational system and nobody would argue against the teaching of reading, writing and arithmetic. But symbolic representations of information and the design of efficient algorithms for automation of different processes have roots as old as written language and calculation. Informatics (also known as a computer science or computing discipline) therefore has been integral to human culture since ever. Moreover, the fast increase of the importance of digital competences in our knowledge society which is based on information and communication technology makes a comprehensive education of informatics an unavoidable part of school experience (Hromkovič 2015; Hromkovič and Lacher, 2017a; Hromkovič and Lacher 2017b; Hromkovič and Steffen, 2011).

Being digital native and a mere user of technology is not sufficient. Technology is changing at such a rapid pace that to thrive and succeed in the information age, we need to understand how the digital world works, how it has been created and how it can be improved going forward. As educators, one of our aims should be to educate the young generation not only to be able to control existing technology, but also to invent and develop new technology.

There is a growing recognition of the importance of offering young students the opportunity of informatics education along with four **R**s: **R**eading, w**R**iting, a**R**ithmetic/mathematics and algo**R**ithms. informatics is scientific basis for digital technologies. Informatics radically change the way we think about, understand, and organize our lives, our surroundings, and the whole world. Therefore, informatics is a part of general education and should be recognized by all as "a truly fundamental discipline that plays a significant role in education for the 21st century" (Caspersen *et al.*, 2019).

Informatics is a distinct scientific discipline, characterised by its own concepts, methods, body of knowledge, and students' achievements (Hromkovič 2015; Hromkovič and Steffen, 2011; Kert *et al.*, 2019). Informatics is known as the study of information and data, computers, and algorithmic processes, including their principles, hardware and software designs, applications, and their impact on society. Many countries use the term computer science (Hubwieser *et al.,* 2011), however, it is also referred to as "Computing", "Informatics", and partially as "Digital Technologies" or "Information and Communication Technologies". Although there may exist different opinions on it, we prefer to use informatics and computer science (CS) (or computing education) as synonymous.

Informatics covers the foundations of computational structures, processes, artefacts, systems, their software designs, applications as well as the networking and their impact on society. The basic principles and fundamental knowledge of informatics shape the thinking, expression, and working of each individual and are much more important for education than the technologies themselves. Computers, technologies "should deepen our understanding of the process of design and creation, it should give us better control over the task of organizing our thoughts" (Dijkstra, 1972).

Several globally known scientists have provided characterisation of informatics as a discipline. Already in 1970s A.P. Erschov called informatics / programming the second literacy (Erschov, 1972; Erschov, 1981). For example, Nygaard (1986) applied the informatics term when describing conceptual modelling and information systems. Harel (1987) described algorithmics as the discipline which covers three complexities: computational complexity, behavioural complexity, and cognitive complexity. Denning and Rosenbloom (2009) developed the argumentation that computing is a fourth great domain alongside the physical, life and social sciences. Recently Denning and Tedre (2019), in their book "Computational Thinking", have discerned four stages of computing/computational thinking development:

1) Phenomena surrounding computers (1950s–1970s).
2) Programming as art and science (1970s).
3) Computing as automation (1980s).
4) Computing as pervasive information processes (1990s to present).

Pupils can be exposed to aspects of computational thinking by engaging in algorithms and programming through diverse means such as data analysis, modelling or robotics.

Juraj Hromkovič and Regula Lacher extended these computational aspects of informatics to a more holistic view by adding abstraction and symbolic representations that enable to describe and investigate the world in an efficient way (Hromkovič, 2015;

Hromkovič and Lacher, 2017b). The three roots of informatics (Hromkovič, 2015; Hromkovič, 2018) offer a more general view on informatics in the broader context of science, humanities, and technology than previous approaches, and allows a clear view of the potential contributions of informatics education.

The Committee on European Computing Education (CECE), jointly established by ACM Europe and Informatics Europe have brought forward a detailed picture of the state of informatics education at school level. The first of the three main recommendations for informatics curriculum (Fig. 1) have stated: "All students must have access to ongoing education in informatics in the school system. Informatics teaching should preferably start in primary school, and at the latest at the beginning of secondary school." (CECE-Report, 2017). A crucial component of the informatics for ALL initiative is the two-tier strategy at all educational levels: 1) informatics as an area of specializationa – a fundamental and independent subject in school, and 2) the integration of informatics with other school subjects. These two trends were named as *Learn to Compute* (specialization) and *Compute to Learn* (integration).

We acknowledge that curriculum design is complex. No single theory of curriculum is commonly accepted that can provide us with a basis for developing our vision for curricular design (Pacheco, 2012). Still there is the issue of balance across computer science, information technology, digital literacies and computational thinking. For example, the UK, Australia and Poland have incorporated elements of all these in their cur-

---

The eight recommendations from *Informatics for All: The Strategy.*[12]

Curriculum Considerations
- ▸ All students must have access to ongoing education in informatics in the school system. Informatics teaching should start in primary school.
- ▸ Informatics curricula should reflect the scientific and constructive nature of the discipline, and be seen as fundamental to 21st century education by all stakeholders (including educators, pupils, and their parents).
- ▸ Informatics courses must be compulsory and recognized by each country's educational system as being at least on a par with courses in STEM (Science, Technology, Engineering, and Mathematics) disciplines. In particular, they must attract equivalent credit, for example, for the purposes of university entrance.

Preparing Teachers
- ▸ All teachers at all levels should be digitally literate. In particular, trainee teachers should be proficient (via properly assessed courses) in digital literacy and those aspects of informatics that support learning.
- ▸ Informatics teachers should have appropriate formal informatics education, teacher training, and certification.
- ▸ Higher education institutions, departments of education, as well as departments of informatics should provide pre-service and in-service programs, encouraging students to enter a teaching career related to informatics.
- ▸ Ministries should be encouraged to establish national or regional centers facilitating the development of communities of informatics teachers who share their experiences, keep abreast of scientific advances, and undertake ongoing professional development.

Teaching the Teachers
- ▸ Intensive research on three different facets, curriculum, teaching methods and tools, and teaching the teachers is needed to successfully introduce informatics into the school system.

Fig. 1. Main recommendations of the CECE report in three areas (CECE, 2017).

ricula for all students although the balance is only likely to be clear from more detailed analysis of curricula (Webb *et al.*, 2019). Previous research comparing computer science curricula in different countries revealed the range of factors affecting the curriculum and how it is implemented (Hubwieser *et al.* 2015).

Finding a place for informatics in the school curriculum is a complex issue since it requires to find sustainable solutions for including a new fundamental discipline among historically firmly established subjects in an educational system that in most countries already works at its maximum capacity. Adding a subject to an existing school curriculum is very challenging, at first because of lack of space. Each country needs to find its own solution, matching its constrains and its overall situation.

In one of the first papers on informatics curriculum, J. Gal-Ezer, C. Beeri, D. Harel, and A. Yehudai (1995) provided a high-level description and argumentation of a high-school curriculum in Israel with emphasis on the basics of algorithmics and teaching

Fig. 2. New Zealand Computer Science Field Guide table of contents
(`http://csfieldguide.org.nz`).

programming. The authors discussed background and motivation for the curriculum, its structure and its initial implementation. Later J. Gal-Ezer and D. Harel (1999) continued development of the computer science curriculum by providing a detailed description with all necessary modules.

Informatics curriculum should include the foundation of the discipline, including theoretical and practical aspects. It should be clearly designed at each school level: elementary, lower secondary and upper secondary or high school. It is very important to encourage new curricula research concerning appropriate methodology, learning design, teaching and learning methods, etc.

The CECE (2017) report has emphasized the following informatics fundamental concepts and practices:

- Data, information, and representation.
- Algorithms and programming.
- Patterns and parametrization.
- Abstraction and conceptual modelling.
- Devices, network and the web.
- Computation and communication.
- Design and interaction.
- Security, privacy, and ethics.
- Societal impact.

For example, New Zealand has included key concepts like algorithms, programming languages, various aspects of coding, formal languages, etc. (Fig. 2). The purpose of the curriculum is to give students a taste of the field of computer science, not to teach it in detail (Bell, 2014). For example, formal languages can be introduced by working with some simple Finite State Automata and based on interesting tasks integrated with grammar or expressions in mathematics.

3. Main Concepts for Teaching Informatics: Constructivism, Constructionism, Critical Thinking, and the Historical Method

The famous general concept of constructivist theory or *constructivism* of Jean Piaget (1950) can be expressed in short by "learning by doing". The activity of learners is decisive in the process of learning and especially significant for the sustainability of acquired knowledge and it's interconnecting with already known facts. Constructivist learning is particularly based on students' active participation in problem solving and critical thinking.

For teaching informatics Seymour Papert (1980) has evolved the idea of Jean Piaget with his well-known "learning by getting things to work". According to this, one tries to teach in such a way that:

(i) Learners create or construct "things", i.e., the results of the activity of learners are products (programs, secret writings, self-verifying codes, data organization, etc.).

(ii) Learners investigate the properties and the functionality of their products.

(iii) Learners create motivation for building better products (better properties, advanced functionality, etc.) and continue with (i).

This concept of Seymour Papert is called *constructionism* (which is built on constructivism) and it is exceptionally natural for teaching programming (Papert and Harel, 1991). Papert wrote in his book "The Children's Machine" (p. 142–143):

> *"Constructionism also has the connotation of "construction set", starting with sets in the literal sense, such as Lego, and extending to include programming languages considered as "sets" from which programs can be made, and kitchens as "sets" with which not only cakes but recipes and forms of mathematics-in-use are constructed. One of my central mathetic tenets is that the construction that takes place "in the head" often happens especially felicitously when it is supported by construction of a more public sort "in the world" – a sand castle or a cake, a Lego house or a corporation, a computer program, a poem, or a theory of the universe. Part of what I mean by "in the world" is that the product can be shown, discussed, examined, probed, and admired. It is out there.*

> *Thus, constructionism, my personal reconstruction of constructivism, has as its main feature the fact that it looks more closely than other educationalisms at the idea of mental construction. It attaches special importance to the role of constructions in the world as a support for those in the head, thereby becoming less of a purely mentalist doctrine. It also takes the idea of constructing in the head more seriously by recognizing more than one kind of construction (some of them as far removed from simple building as cultivating a garden), and by asking questions about the methods and the materials used. How can one become an expert at constructing knowledge? What skills are required? And are these skills the same for different kinds of knowledge?"*

A program as a product of learners' activity has a functionality that can be especially well investigated if the execution of the program is visualized e.g., by moving robots or drawing pictures. But one must not restrict "learning by getting things to work" to programming. If, for example, the product of the activity of learners is a cryptosystem, one can investigate this product by applying it in communication process and by trying to break it. For everything we want to teach one can design the teaching process by following the concept of constructionism. Constructionist principles support the strategies of using more kinesthetic and active approaches and this is embodied in the "unplugged" style in informatics (Kirschner *et al.*, 2006). The "unplugged" approach of teaching refers to the use of activities to teach informatics concepts without computers (Bell *et al.,* 2009; Hromkovič 2018; Hromkovič and Lacher 2019, Hauser et al 2020). But one has to be very careful with a well working

implementation. We know that too restricted guidance does not work (Kirschner *et al.*, 2006), and there has to be find a good balance between guidance (interaction with teacher) and learner's activity.

Constructionism as a teaching method fits very well the concept of *critical thinking*. Critical thinking for the purpose of this paper can be summarized as follows:

> Do not teach only the product of science and technology (facts, theorems, methods, models, equipment etc.) and how to apply them, but the processes of their discovery and their development. One has to recognize that each result of science, and each product of technology are only intermediate steps on the way to goals behind the horizon. One has to imagine that all these achievements have their drawbacks and reach only a fraction of posed goals.

This means that the focus is on creative processes. We aim to explore the intellectual potential of learners and support the leaners to become creative personalities. Personalities who do not accept anything, that they are not able to verify by corresponding scientific methods. Personalities who understand the creative processes of research and development so well that they enhance our knowledge by discovering new facts and improve the products of science and technology.

To design teaching based on constructionism and critical thinking we recommend using the *historical method* (Behr, 1996; Bruckheimer and Arcavi, 2000; Bussi and Bartolini, 1996; Man-Keung, 2000; Swetz, 2000). Jean Piaget and Seymour Papert have paid attention to historical approaches to the evolution of knowledge.

> *"In the simplest case the individual development is parallel to the historical development, recalling the biologists' dictum, ontogeny recapitulates phylogeny. For example, children uniformly represent the physical world in an Aristotelian manner, thinking, for example, that forces act on position rather than on velocities. In other cases, the relation is more complex, indeed to the point of reversal. Intellectual structures that appear first in a child's development are sometimes characteristic not of early science but of the most mod- ern science. So, for example, the mother structure topology appears very early in the child's development, but topology itself appeared as a mathematical subdiscipline only in modern times."*
> (Papert, 1980, p. 163)

John P. Smith, Andrea di Sessa and Jeremy Roschele (1994) has used an historical revisitation of Galileo in developing his approach to learning the physics of free fall.

Applying the historical method means that teachers first learn the genesis of a subject they want to teach. Starting with motivations, continuing with unsuccessful attempts, failures, or partial successes. If learners can experience in their activities at least part of these processes and learn from own failures, learners will acquire an understanding of the subject that is incomparable deeper that what learners could achieve

by presentations of the final products of these processes. The sustainability of learning by intense trying, failing, and improving is incomparably higher than by learning the final products of these processes. Moreover, the learners understand that all the currently available products of science and technology are far from being perfect, that they will evolve and that the learners are part of this journey. The current products are mere interim steps on the way to our goals. So, the historical method is a very helpful instrument for teachers who want to design teaching and learning processes based simultaneously on constructionism and on critical thinking.

Also, we should deal with new methods and strategies which are useful for informatics education. Recently flourished the computational thinking term holds hope that informatics covers a thinking tool for understanding out technology-infused world. In 2011, a committee of experts, examining the role that informatics would play in bringing computational thinking to K-12, broadly defined computational thinking as "an approach to solving problems in a way that can be solved by a computer … a problem solving methodology that can be transferred and applied across subjects" (Barr and Stephenson, 2011). Peter J. Denning and Matti Tedre wrote in their book "Computational Thinking" (2019, p. 21): "Computational thinking evolved from ancient origins over 4,500 years ago to its present, highly developed, professional state. The long quest for computing machines was driven not only by the need for speed, but also to eliminate human errors". This is also the main reason why we offer a short history of informatics in the next chapter. This history will be the starting point for developing a curriculum for informatics in this paper.

The key concepts and techniques of informatics have been translated into curricula that incorporate broad areas of algorithms / programming, data structures, data representation, digital infrastructure, digital applications, and human factors (ACARA, 2014; Hubwieser *et al.,* 2015; Seehorn *et al.,* 2011). Some countries have implemented these elements to different levels in their curricula.

4. Three Roots of Informatics

We start here with the concept of "three roots of computer science" as introduced by Hromkovič and Lacher (2017b). Probably the shortest specification of "informatics" is a science of automated storing, transporting and processing of information.

The crucial terms in this definition (and therefore the most fundamental notions of informatics) are *information*, and *automation*. Because of that, we consider the following three roots of computer science:

  (i) Information and data.
 (ii) Automation and algorithms.
(iii) Digital technology.

Let us give a little bit more explanation to understand why this classification of computer science is the most natural one.

4.1. *Information and Data*

The history of computing from the point of view of information and data started some 5400 years ago with the "first big data crisis" in human history. Mesopotamia had that time about 1 Million inhabitants and this empire needed to manage matters related to private properties and taxes. However, at that time the only possibility to store and maintain all information needed was in the minds of the officers. The solution to the almost collapsing management of the empire was the development of writing (scripts). This was the birth of digitalization because digital information representation is the representation of information (called data) as a sequence of discrete symbols (letters, digits) of an alphabet. For the first time in the existence of human civilization, humans were able to save and keep information externally (outside of their minds), to broadcast it, and even to transport (communicate) it across arbitrary distances.

This was a true revolution in information processing (Williams, 1985). Three fundamental and truly computer science related tasks had to be solved as the consequence of this development:

a) How to represent information as data in such a way that the representation is understandable, unambiguous, not too long, and suitable for efficient information processing (for instance calculating with numbers)?
b) How to organize (manage) data in such a way that any information needed could be found quickly?
c) How to protect confidential data and make them available only for persons who are allowed to see them?

Finding better and better solutions for these three tasks so far took thousands of years and may well be a never-ending story. Big subareas of computer science such as security, data management, compression, and self-correcting codes are products of the effort to answer the questions above.

4.2. *Automation and Algorithms*

Human civilization strived to be efficient in everything people did since ever, and so we are unable to fix the birth of algorithmics (for some more involved history see Dasgupta, 2014; Tedre, 2014). The ancient way of automation was to acquire knowledge and use it to develop procedures tailored to the specific goals (for instance, develop some products). Since the procedures were described in such a way that humans were able to successfully apply them without understanding why they work, we may call this automation. The original automation did not need machines. If one wants to look at true algorithms as exact descriptions of activities in the unambiguous language of mathematics, the history is at least 4000 years old (Knuth, 1972). Starting with Babylonian stone plates, continuing with the book "Elements" of Euclid and with the big book of business calculations in the 8th century by Al-Khwarizmi whose name gave us the term "algorithm". Throughout the history of mankind humans tried to automate everything they

were able to sufficiently understand. And the efficiency (computational complexity as the measure of the amount of work executed during the calculation) of algorithms was from the very beginning in the focus of interest.

One of the great ancient stories related to efficiency is the development of number representations (Williams, 1985). Many different number representations have been developed by different civilizations. But in the end the main criterion for choosing an appropriate number representation became clear to be the efficiency in calculation. To underline this statement, we call attention to the following fact: For some algorithmic problems in advanced algebra and number theory, some special number representations were developed. They enable to efficiently calculate some operations, for which the common decimal and binary representations do not allow any efficient execution.

The history of problem solving, algorithm design, and computational complexity is very rich in great ideas that have high potential to enrich our education and to explore the creative potential of pupils.

4.3. *Digital Technology*

The idea to "create" a machine that could execute part of human work is very old. It is a natural continuation of the idea of developing instruments making our work more efficient and simultaneously more accurate or "trustable" in executing different activities. The history of developing mechanical calculators started with Wilhelm Schickard who tried to design and unsuccessfully develop a mechanical device performing the basic arithmetic operations automatically. In 1642 the French mathematician Blaise Pascal invented the first mechanical calculator. Another famous scientist Gottfried Wilhelm Leibniz developed his calculator in 1716. In 1871 Charles Babbage designed his Analytical Engine, the first programmable computer. Lady Ada Lovelace developed first programs for Analytical Engine.

But one is not allowed to reduce the history of IT to the development of computers. The interconnection networks as a communication technology are also an important part of IT and their history spans at least 2 millennia. This is the case because creating signals (visual or acoustic) and using sequences of signals to code information is the very base of communication technology. We are unable to fix the starting point of using sequences of signals in the history of mankind.

Nevertheless, technology is not only about building hardware. This hardware needs to be programmed, operating systems need to be developed, and programming languages and applications need to be built. This is one of the reasons we consider, for instance, programming and communication protocols as part of technology. For sure, programming in the broad interpretation as problem solving and describing the solution method can be assigned to algorithms. But programming in the narrow interpretation as "explaining" a solution method to a machine so the machine can execute the method is strongly related to technology. And to properly understand programming languages one should understand the underlying hardware with its potential and its restrictions.

Summarizing the above, there are several reasons to look into the roots of informatics. It allows to understand that informatics is as old as science itself and that tasks and concepts of informatics have been an integral part of human culture since ever. Since concepts of informatics have been created in strong interaction with other scientific disciplines, especially with mathematics and language development, one can also teach some computer science concepts inside other disciplines and one can contribute to understanding mathematics and languages by teaching informatics.

But the main reason to study the roots of informatics, the history of its main discoveries, and the development of its fundamental concepts is to enable a reasonable design of computer science curricula and to create textbooks that offer successful, suitable, and enjoyable learning.

To illustrate it by an example let us take cryptology as a theory of secret writings. Following the history one can start to teach about 4000 years old method of transposition and then the 2300 years old method of substitution. We follow the security principle of ancient time telling the secret writing must be designed in such a way that one can be learned by heart and does not need to safe the description in a written form. Doing is properly by following the development of crucial ideas in small steps children in age 10 to 12 can become true experts in ancient cryptography, who are able to design and apply completely new, original cryptosystems. Then following the development of the first method for breaking such cryptosystems in the 7th century the pupils learn to use analysis of the relative letters frequency to break all monoalphabetic cryptosystems.

All the development you can relate to the development of human culture and deal with the tasks what kind of data have been protected in different human cultures (not only secrets of the army, but also technology for producing different kinds of products or tax declarations). After getting into dead end pupils can be confronted with different attempts to avoid the possibility to learn something from the letter frequencies in the cypher texts. In this way they can converge to the cryptosystem Viginiere, which uses a repeating key to select different encryption alphabets in rotation, and was considered to be secure for about 300 years. In age about 15 students can use again stochastic to break this cryptosystem based on stochastic concepts as Charles Babbage succeeded in breaking this cipher more than a century ago (Singh, 1999).

Again the development of secret writings has been in deadlock. How one can try to overcome it can be the topic of informatics in high schools. Systems such as ENIGMA uses the idea to change the encryption scheme after encrypting each particular letter and the roots of this approach one can find already in 15th century. The highlight of teaching cryptology are the public key cryptosystems and protocols based on them. All these advanced cryptosystems ask for really involved algebra and number theory. But following the genesis of the crucial ideas one can find a way to design public key cryptosystems by means of high school mathematics in such a way that the students understand why it is possible to make the encryption algorithm public and in spite of that only a person possessing a secret is able to decrypt. In this way pupils learn to imagine how the development of fundamental concepts of computer science (especially com-

plexity theory and algorithmics) offered a breakthrough in designing and implementing secure cryptosystems.

Following the above presented path students acquire a deep understanding of the genesis of cryptology as a scientific discipline. Moreover, the students see the whole history of developing secret writing as a process going from one product (achievement) to a next, better one. At the very end students gain competences in the sense of a true expertise, and so they are able to create new cryptosystems, and find methods to break the cryptosystems designed by analyzing their weaknesses. In this way we educate personalities who are able to contribute to society in a creative way.

5. Development of Informatics Curriculum

National and international efforts are dedicated to develop, support and evaluate curriculum development. From the recently growing number of publications, it can be seen that informatics curricula, both for primary and secondary schools, is currently an important subject and introduced in many countries (e.g. Bell *et al.*, 2014; Department…, 2021; Directorate…, 2019; Education Scotland, 2017; National …, 2018; Seehorn *et al.*, 2011). In some countries the informatics curriculum is well entrenched, however it is a relatively new phenomenon in others. This poses challenges especially in preparing and supporting teachers as they transition from initial teacher qualifications and experience in other learning areas to the teaching of informatics (Brown *et al.*, 2014).

Anja Balanskat and Katja Engelhardt (2014) have explored primary and secondary school informatics curricula initiatives across Europe. In 2011 an ITiCSE Working Group (Hubwieser *et al.*, 2011) provided research findings about informatics curricula of secondary education from different countries, and in the process developed a category system (Darmstadt Model) to support comparisons across regional and national boundaries.

A few years later, a 2015 Working Group applied the Darmstadt Model to analyse articles within two TOCE K-12 computer science education special issues (Hubwieser *et al.*, 2015). This work sought to understand informatics curriculum, aims, goals and competencies, programming languages, tools adopted, assessment practices and teacher training.

For example, informatics courses in Poland were divided into three phases (Sysło and Kwiatkowska, 2015). The first stage begins by training elementary school pupils in basic skills using information technology. In the second stage, secondary school pupils are trained in the ability of computing, understanding behind technology, and problem-solving. By the third stage, the informatics course is one of the important subjects for the high school final examinations. The main goal of these three stages is to help students understand and analyse problems, use computers or other computer equipment to solve problems, and also apply technology to society or to their own lives.

South Korea has also developed a new curriculum for schools. They started to promote computer education courses in 1971, with more than 34 h of computer courses in each grade of primary and secondary education (Heintz *et al.*, 2016). At first, they only

focused on teaching computing theory and the concepts of information science, but later they changed the curriculum to include the training of pupil's digital literacy, computational thinking, and programming skills.

In many countries the term curriculum is a high-level concept relating to specific learning objectives and measurable outcomes or benchmarks for learning levels. Educators in the US refer to curriculum as well-articulated bodies of courses, modules, and lesson plans. When educators from outside the US use the term curriculum, people from the US can understand their meaning as a computer science framework or standards.

In curricula we usually talk about big ideas that should be the focus of education for understanding. A big idea is a concept, theme, or issue that gives meaning and connection to concrete facts or skills. For example, in physics education the "big ideas" approach has a long tradition (e.g. Principles and Big Ideas of Science Education, `https://www.ase.org.uk/bigideas`). In informatics education Tim Bell, Paul Tymann, and Amiram Yehudai (2018) have presented a list of ten big ideas that have been distilled based on input from curriculum designers and computer science education experts around the world. However, while there is a consensus on this approach in the education research community, it has not yet become mainstream at the policy level.

Curriculum refers to the blueprint for learning that is derived from the desired results. Wiggins and McTighe in their excellent book on designing curricula "Understanding by Design" (1999) says "Curriculum takes content (from external standards and local goals) and shapes it into a plan for how to conduct effective and engaging teaching and learning. It is thus more than a list of topics and list key facts and skills (the "inputs"). It is map for how to achieve the "outputs" of desired student performance" (p. 6).

The curriculum development process systematically organizes

- what will be taught,
- who will be taught, and
- how it will be taught.

Each component affects and interacts with other components. For example, what will be taught is affected by who is being taught (e.g., their stage of development in age, maturity, and education). Methods of how content is taught are affected by who is being taught, their characteristics, and the setting. Considering the above three essential components, the following are widely used for curriculum development in formal education settings:

- Content is identified (what).
- Target audience (who).
- Intended outcomes/objectives (what the learners are able to learn).
- Methods to accomplish intended outcomes (how).
- Evaluation strategies for content and intended outcomes (what works).

Informatics curriculum development has some general components. We are going to focus on the content of informatics curriculum from primary education to secondary and high school education. The challenge in the curriculum development process is selecting content that will make a real difference in the lives of the learner and ultimately society as a whole. At this point, the primary questions are: If the intended

Table 2

Six levels of the revised Bloom's taxonomy

1. Remember	Learners can recognize (identify) concepts already learned, recall (retrieve) information and so are prepared to interconnect it with new knowledge.
2. Interpret (understand)	Learners can construct the meaning of instructional messages, use it to illustrate concepts by examples, to classify (categorize), to compare, to abstract.
3. Apply	Learners can apply own knowledge to solve tasks in different settings, execute algorithms (methods), implement strategies under different circumstances, simulate.
4. Analyze	Learners can compare objects (models) with respect to their attributes, structure, break down objects into components (model), compare and categorize.
5. Evaluate (judge)	Learners can use analysis to judge objects and products of human activities with respect to different criteria, choose an appropriate evaluation criterion with respect to goals, judge efficiency of algorithms / methods and the quality of their outputs, reflect on progress achieved, test hypotheses.
6. Create	Learners can design and develop their own, original products, generate hypotheses and verify them, plan activities, synthesize different parts into a new model, create knowledge.

outcome is to be attained, what will the learner need to know? What knowledge, skills, attitudes, and behaviours will need to be acquired and practiced? The scope (breadth of knowledge, skills, attitudes, and behaviours) and the sequence (order) of the content could also be discussed. For a more involved view on curriculum design see (Wiggins and McTighe, 1999).

In this paper we use the revised taxonomy of Benjamin Bloom (Bloom *et al.*, 1956; Anderson and Krathwohl, 2001) in order to find the right sequence of competences one aims to achieve in selected subareas of informatics (Table 2). The goal of using this approach of cognitive psychologists is to move the attention from the static notion of "educational objectives" to cognitive processes that are crucial to support the development of the intellectual potential and creativity of the pupils (learners). For us it is important not to see the Bloom's taxonomy as a hierarchy of cognitive activities (De Bruyckere *et al.*, 2015), but as a helpful instrument for designing educational objectives.

A similar approach was used for classification informatics tasks of the Bebras challenge on informatics and computational thinking (Dagienė *et al.*, 2020). The contribution of this classification of the Bebras tasks is a new concept for classifying tasks that also offers new ideas for generating tasks and which is used for creating spiral curricula for teaching informatics.

6. Examples of Designing an Informatics Curriculum

A curriculum in different countries may differ a lot, but there are some fundamental computer science topics such as programming, problem solving and algorithms, abstraction and data representation, data management and security that cannot be omitted if one does not want to miss the most basic competencies of computer scientists. "The vast

majority of any informatics curriculum will be scientific in nature, focus on the key concepts in the field and reflect the constructive aspect of the discipline. Attention should be given to a range of topics such as data, programming, algorithms, networks and the web, design and human computer interaction, security, privacy and ethical considerations. Moreover, the conceptual and practical elements should be blended in a way that reflects the multiple links between the two." (CECE-Report, 2017).

In this chapter we design curriculum for programming, algorithms, robotics, and networking in order to illustrate our approach. The detailed implementation of our design can be found in the textbook series covering all age groups starting with kindergarten and finishing with high schools (Barnett *et al.*, 2020; Hauser *et al.*, 2020; Hromkovič, 2018a, 2018b, 2018c; Hromkovič and Kohn, 2018; Hromkovič and Lacher, 2019, 2021).

The curriculum integrates the concepts of constructivism, critical thinking, viewpoint of competences, as well as the historical approach and combines these with the hierarchy of the revised Bloom's taxonomy (Anderson and Krathwohl, 2001).

We plan to continue with this project and design curricula for other informatics / computer science / computing areas and present them in future papers.

6.1. *Programming and Algorithms*

What do educators expect from teaching programming? Programming in narrow interpretation means talking to technology (computer, robot, etc.) in order to describe unambiguously an activity they have to execute. In that sense a program is a grammatically correct text in a programming language, that is built with mathematical rigor, i.e. every text has only one interpretation. Therefore, teaching programming in narrow interpretation means to master a programming language so far that one can write programs correctly describing the activities to be automated and can correct syntactic (grammatical) errors in programs. This is very similar to learning a natural languagea – the competences of being able to express oneself in an understandable way and grammatically correct are the main goals. Here we add the ability of developing a language by introducing new words. From this point of view programming in a narrow sense is very much related to learning how to control technology.

Programming in broad interpretation includes also problem solving. This is the way to support one more creative dimension and so the interpretation we prefer to use. From this point of view programming is on the intersection of algorithmic and technology. We aim to teach programming as:

- Formulating goals, creating motivations.
- Searching for solution methods (strategies).
- Describing solution methods by programs.
- Searching for errors in programs and correcting them.
- Modifying given programs in order to extend their functionality.

We start with programming in the narrow interpretation and continue investigating algorithms with the focus on problem solving. Then we combine programming with

robots. Finally, we discuss teaching communication in networks on the abstract level of graphs.

The choice of appropriate age group is based on one hand on the development of cognitive processes and topics taught in other subjects, and on the other hand by experimental teaching at more than 500 project schools involving approximately 20 000 pupils and 1 000 teachers.

The abbreviation **BT** stands for the revised Bloom's taxonomy following by appropriate levels. The numbers **BT 1-6** are not allowed to be related to a hierarchical view on cognitive activities, they are used to address educational objectives to be discussed.

6.1.1. *Programming*

1. **Executing programs as sequences of instructions (BT 1-3, Age 4+)**

   One starts with a very poor programming language consisting of a few instructions only (in the beginning represented by a symbol and later by one word) and the goal is to correctly interpret a given program, i.e., to take the role of a robot and to execute few commands. The usual starting point is a movement in two-dimensional space. For young children this strengthens their ability of orientation in space and planning with respect to time.

2. **Developing (writing) programs without inputs as sequence of unstructured instructions (BT 2-3, Age 4+)**

   A program without input (parameter) describes exactly one activity. The goal is to transform a description of an activity (for instance a trajectory in a landscape) to a program as another description of this trajectory. A classic example is writing programs to draw simple pictures.

3. **Searching for logical errors if a program does not execute the expected activity (BT 4, Age 6+)**

   We are still working with programs consisting of sequences of unstructured instructions. To train pupils to find and correct logical errors in own programs as well as in given programs is as important as to learn to write programs. Programming skills cannot be mastered without this competence. To support the training of pupils in achieving this competence one has to offer a programming environment in which the pupils can execute programs slowly and move forward as well as backward in the execution, while observing the effect of particular instructions of the program.

4. **Using loops without parameters (BT 2-4, Age 7+)**

   First pupils recognize repetition of patterns in programs and can shorten the program description by applying repeat-loop. Secondly pupils recognize possible repetitions in task descriptions (drawing pictures, running a regular trajectory) and design programs with loops.

5. **Understanding and applying modular design (BT 2-4, 6, Age 10+)**

   Pupils learn to partition a task into a couple of subtasks, develop and check the correct functionality of the programs for the subtask they are working on, and to

use these programs as building blocks for creating a program for the whole task. Pupils can use modularity to introduce new words (commands) to the programming language used.

6. **Working with parameters (BT 2-4, Age 12+)**

   Parameters are powerful concept of programming. To learn to work with parameters is one of the biggest jumps in teaching programming. One switches from programs executing exactly one activity to programs executing infinitely many activities depending on their input values (parameters). We speak here about parameters and not about variables, because in this first stage we do not allow to change the value of the parameters during the execution of programs. Pupils can use parameters to determine size, color or even form of pictures.

7. **Working with variables (BT 3-5, Age 13+)**

   The goal is to understand variables as names of places in computer memory and to learn to work with them to store and read information, but also to process information. Pupils can actively change the values of variables during the execution of the program.

8. **Recognizing different data types and using appropriate operations on them (BT 2-4, Age 14+)**

   Here pupils have to interconnect their knowledge about abstract representations of information by numbers, texts (symbol sequences) or tables and pictures and learn to handle different data types in a different way. Pupils can work on pattern recognition of various samples and get abilities to deal with abstraction. Note that the base for abstract representation is part of "information and data" in our curriculum and starts already at the age of 4.

9. **Understanding and applying conditional instructions (BT 2-4, Age 13+)**

   Pupils can branch programs with if-then-else and to use conditional loops such as while. Pupils touch the interface to formal language of logic and can work with a syntax of logical expressions.

10. **Correcting syntactic errors in text-based programs (BT 4, Age 10+)**

    This competence should not be taught in this sequence but should be an integral part of the whole process of teaching text-based programming. Using well-developed debuggers pupils have to learn to use the comments of the debugger and fix syntactic errors.

11. **Using data structures to work with data (BT 2-3, Age 15+)**

    Pupils can work with data structures of arbitrary size as arrays or lists, and to read, and process efficiently large amount of data. Pupils learn also to filter data and to merge two data sets.

12. **Programming functions and using them as subroutine (BT 4-6, Age 15+)**

    Pupils can design and implement programs that compute functions and their functions as building blocks of more complicated programs.

13. Designing and implementing recursive programs (BT 2-6, Age 17+)

Learners first understand and execute recursive programs and use trees to describe the execution of recursive programs. Then learners can use recursion as a strategy for problem solving (intersection with algorithmics) and develop implementations of recursive algorithms.

The next part focuses on problem solving related to algorithm design.

6.1.2. *Problem Solving and Algorithms*

Here we assume that the pupils already are familiar with abstraction (which is a part of the subarea "Recognizing different data types and using appropriate operations on them", 6.1.1 item 8) to describe information by numbers, texts and graphs and can interpret problem descriptions using these abstract objects.

1. Classifying solution proposals into feasible and unfeasible (BT 1-2, Age 8+)

Pupils understand the problem instance description and can interpret it correctly. They prove the competence of the correct interpretation by ability to classify solution candidates into feasible and non-feasible ones.

2. Searching for solutions for small problem instances (BT 2-3, Age 8+)

Pupils can search and find solutions for small instances of different kinds of problems by trial and error.

3. Listing all solutions of a problem instance or all objects with prescribed properties (BT 2-4, Age 8+)

For objects with simple, given properties pupils can use trees in order to find all such objects. Later they can use trees to find and systematically list all solutions of small problem instances.

4. Applying a given criterion to evaluate and compare solutions (BT 2-5, Age 9+)

Pupils can assign values (costs) to presented solutions to a problem instance by a given criterion (cost functions). They can use the costs of solutions to compare the solutions. Later they even can create cost functions (criteria) that enable to evaluate solutions for a given purpose.

5. Understanding descriptions of optimization problems (BT 2-3, Age 9+)

Pupils can interpret descriptions of optimization problems, and so distinguish between constraints (that have to satisfied) and optimization goal.

6. Solving instances of optimization problems (BT 3-5, Age 9+)

Pupils can solve small instances of optimization problems either by trial and error or by listing all solutions, evaluating them and choosing an optimal one.

7. Discovering and applying strategies for problem solving (BT 2-5, Age 13+)

Pupils can search systematically for solutions by applying general solving strategies like greedy or simply trying all possibilities (brute-force).

8. **Understanding concrete algorithms (BT 2-5, Age 14+)**

   Pupils understand and can successfully apply concrete algorithm solving all instances of a given problem. They understand (at least intuitively), why these algorithms work properly for any given instance of the problem considered.

9. **Modifying algorithms (BT 4-6, Age 14+)**

   Learners can modify (adapt) known algorithms to modified problem settings or new situations.

10. **Understanding and applying methods for design of algorithms (BT 3-6, Age 17+)**

    Learners understand how robust method for the design of efficient algorithms work. They know examples of greedy algorithms, local search, divide and conquer (divide et impera) and dynamic programming. Later they can use these design methods to develop algorithms for simple problems. Combining their knowledge from programming they can implement them, also using recursive programs for the implementation.

11. **Analyzing complexity of algorithms (BT 5, Age 17+)**

    Learners can analyze the computational complexity (amount of work executed) of algorithms working on concrete problem instances. For simple algorithms they can analyze its space complexity and its time complexity. Learners can compare two different algorithms for the same problem with respect to efficiency.

12. **Designing efficient algorithms (BT 5-6, Age 17+)**

    Learners are able for a given problem and a given complexity bound design algorithms solving the problem within the prescribed complexity.

13. **Testing and verifying algorithms and programs (BT 5-6, Age 17+)**

    Learners are able to test algorithms on a chosen set of inputs or to logically argue why algorithms designed and implemented work correctly for any input data.

14. **Applying machine learning to solve problems (BT 3-5, Age 15+)**

    Learners are able to develop programs that can learn from data sets to find solutions to given problems with high probability or to play games for which we do not know a winning strategy (e.g. chess).

6.2. *Robotics*

Robotics programs can be engaging learning environments for acquiring core informatics and computational thinking competencies. Several empirical studies evaluate the effectiveness of a robotics programming curriculum for developing informatics knowledge and skills.

Programming robots differs from programming computers. First one needs some knowledge of physics and engineering in order to build robots that are able to execute the aimed physical activities. Secondly one has to move from "executing commands"

to "moving robots from one state to another state" and fix by experiments how long the robot has to be in a state in order to finish a partial activity. The design of a curriculum for early childhood education on computing should include active research on what programming might be for that age (Chioccariello, *et al.*, 2004).

Designing playful programming construction kit is an interesting and challenging activity. One aspect, worth mentioning here, is what a designer thinks are children capable of master when programming their construction. This depends on both the cognitive development of the child, and how the construction kit, including its programming environment, is designed.

1. **Writing programs navigate robots from a to B (BT 2-4, Age 4+)**

   Pupils can design programs as sequences of commands in order to move robots from one position to another position. Pupils are able to check the correctness of their programs by executing them and pupils can correct them when they do not work as intended.

2. **Describing the state of a robot (BT 1-2 Age 10+)**

   Working with robots requires to think in states. Pupils have to be able to interpret state description correctly and explain what a robot is doing in a given state.

3. **Adjusting the robot to reach a state given by some parameters (BT 2-3, Age 10+)**

   Pupils know which commands or which sequence of instructions one has to apply to "move" the robot to a particular state.

4. **Using commands to move a robot from one state to another (BT 2-3, Age 10+)**

   Pupils can use the appropriate commands to move the robot form a given state to another one.

5. **Constructing a robot as a mechanical machine with sensors (BT 2-6, Age 12+)**

   Pupils can construct robots as a mechanical device able to move and work in their environment. Pupils can add sensors to robots and use them for different purposes.

6. **Using experiments to develop programs allowing robots to follow a trajectory (BT 2-5, Age 13+)**

   The difference between moving robots in real environment and on the screen is that one has to take into account the properties of the physical environment (e.g. friction). Pupils can determine by experiments how long the robot has to work in a given state in order to reach the goal.

7. **Using sensor to program autonomous behavior of robots (BT 2-6, Age 14+)**

   Pupils can use their knowledge about conditional commands from programming lessons to program robots working according to data offered by the sensors of the robot.

8. **Programming robots to learn (BT 3-4, Age 15+)**

Pupils can write programs for robots that enable robots to learn their environment and modify their own autonomous behavior with respect to "knowledge" acquired.

9. **Design, build and program robots for special purposes (BT 4-6, Age 17+)**

Starting from a given specification for automating a mechanical work, can design, build and program robots that are able to execute this work. This competence has to be combined with the ability to plan, coordinate and cooperate inside of a project group.

6.3. *Communication in Networks*

The goal here is to understand the design of communication networks that enable efficient transport (broadcast) of information on an abstract level.

1. **Understanding sequences of signals as information representation (BT 1-2, Age 4+)**

Pupils can correctly interpret sequences of physical signals and can mimic communication by using them.

2. **Creating codes as signal sequences (BT 3-6, Age 6+)**

Pupils can create own codes for representing different messages and use them in communication. Older pupils can even design codes that are resistant to small errors.

3. **Modelling interconnection networks by graphs (BT 1-2, Age 8+)**

Pupils can interpret graphs as models of communication networks and can describe communication structures by graphs.

4. **Understanding and applying strategies for information broadcast in networks (BT 2-3, Age 10+)**

Pupils understand strategies for disseminating information in networks and can apply and simulate them in different communication modes.

5. **Understanding and applying strategies for information accumulation and evaluation in networks (BT 2-3, Age 13+)**

Pupils understand strategies for accumulating information and for computing functions of arguments distributed in networks and can simulate them.

6. **Understanding and applying strategies for gossiping in networks (BT 2-3, Age 14+)**

Pupils understand strategies to complete exchange of information among all parties of a network and can simulate it in a concrete network.

7. **Measuring the time complexity of executing communication tasks in different networks (BT 3-4, Age 16+)**

Learners can analyze communication algorithms with respect to time needed to complete the communication strategies.

8. **Comparing different strategies for communication tasks in networks (BT 4-5, Age 16+)**

Learners can evaluate communication strategies in different networks, compare them with respect to their efficiency and choose an appropriate strategy.

9. **Designing networks with good communication properties (BT 4-5, Age 17+)**

Learners can design networks with very good communication properties with respect to information dissemination.

Conclusion and Discussions

Informatics should be recognized as a vital, important 21st century discipline. Considering that digital technology increasingly plays a pervasive role, informatics education is necessary to ensure sustainable and balanced development of the digital society."

Our contribution is to design curricula for teaching computer science in such a way. Our design is novel and it is to be noted that this is the case not only from the informatics point of view. Our approach, which is based on constructionism and critical thinking, offers a pattern other school subjects could use to improve their curricula. Our experiments with thousands of students showed that both the mastery of the subjects deepened a lot and the sustainability grew essentially. Combining thinking about the history of processes of developing basic concepts with the cognitive progress of students is the best way for the design and implementation of teaching sequences. It supports creativity and exploring of the intellectual potential of the students.

One could propose to extend our examples of curricula by appropriate tasks the students are able to solve for any of the competences listed. We omit to do this because we already have published a series of textbooks (also addressing specifically teachers) containing this in detail.

We do not consider the presentation of the competences in section 5 as a final product. It is a further step in developing more and more appropriate curricula for informatics. Everybody is invited to join us on this journey by contributing. Especially new missing competences could be added, or existing ones could be split into a sequence of more detailed ones.

Acknowledgement

References

ACARA (2014). Australian Curriculum Assessment and Reporting Authority. *Australian Curriculum: Digital Technologies*. `https://www.australiancurriculum.edu.au/`

Anderson, L. W., Krathwohl, D. R. (Eds. , 2001). A *taxonomy for learning, teaching, and assessing: a revision of Bloom›s taxonomy of educational objectives*. New York: Longman.

Balanskat, A., Engelhardt, K. (2014). Computing our future Computer programming and codinga – Priorities, school curricula and initiatives across Europe and Technology in Computer Science Education (ITiCSE'19). ACM, New York, NY, USA, 257–258. `https://doi.org/10.1145/3304221.3325535`

Barr, V., Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads* 2(1), 48–54. DOI:10.1145/1929887.1929905

Barnett, M., Hromkovič, J., John, A.L., Lacher, R., Lütscher, P., Staub, J. (2020). *Einfach Informatik Programmieren mit Robotern KG 1 / 2* (in German). Klett und Balmer.

Behr, H. (1996). Teaching Mathematics with Historical Componentsa – some Experiences and Ideas. In: Jahnke Hans-Niels, *History of Mathematics and Education: Ideas and Experiences*, Verlag Vandenhoeck & Ruprecht, Göttingen, 27–37.

Bellettini, C., Lonati, V., Malchiodi, D., Monga, M., Morpungo, A., Torelli, M., *et al.* (2014). Informatics education in Italian secondary schools. *ACM Transactions on Computer Science Education*, 14(2), 1–15.

Bell, T., Alexander, J., Freeman, I., Grimley, M. (2009). Computer science unplugged: school students doing real computing without computers. *New Zealand journal of Applied Computing and Information Technology*, 13(1), 20–29.

Bell, T. (2014). Establishing a nationwide CS curriculum in New Zealand high schools: providing students, teachers, and parents with a better understanding of computer science and programming. *Communications of the ACM*, 57(2), 28–30.

Bell, T., Tymann, P., Yehudai, A. (2018). *The Big Ideas in Computer Science for K–12 Curricula*. European Association of Theoretical Computing Science, 124, 2–12

Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*. New York: David McKay Company.

Brown, N.C.C., Sentance, S., Crick, T., Humphreys, S. (2014). Restart: The Resurgence of Computer Science in UK Schools. *ACM Trans. Comput. Educ.* 14(2), Article 9 (June 2014), 22 pages. `https://doi.org/10.1145/2602484`

Bruckheimer, M., Arcavi, A. (2000). Mathematics and its History: An Educational Partnership. In: V. Katz (Ed). *Using History To Teach Mathematics, Mathematical Association of America*, Washington DC, 135–145.

Bussi, M., Bartolini, G. (1996). History in the Mathematics Classroom. In: H.-N. Jahnke. *History of Mathematics and Education: Ideas and Experiences*, Verlag Vandenhoeck & Ruprecht, Göttingen, 39–66.

Caspersen M.E., Gal-Ezer, J., McGettrick, A., Nardelli, E. (2019). Informatics as a fundamental discipline for the 21st century. *Communications of the ACM*, 62(4), 58–63.

CECE (2017). *Informatics Education in Europe: Are we all in the Same Boat?* Report by the Committee on Europe Computing Education (CECE), ACM, `https://www.informatics-europe.org/news/382-informatics-education-in-europe-are-we-on-the-same-boat.html`

Chioccariello, A., Manca, S., Sarti, L. (2004). Children's playful learning with a robotic construction kit. In J. Siraj-Blatchford (Ed.). *Developing new technologies for young children* (pp. 93–112). London: Trentham Books Ltd.

Dagienė V., Hromkovič J., Lacher R. (2020). A Two-Dimensional Classification Model for the Bebras Tasks on Informatics Based Simultaneously on Subfields and Competencies. In Kori, K., Laanpere, M. (Eds) *Informatics in Schools. Engaging Learners in Computational Thinking. ISSEP 2020*. Lecture Notes in Computer Science, 12518. Springer. `https://`doi.org/10.1007/978-3-030-63212-0_4

Dagienė, V., Jevsikova, T., Stupurienė, G., Juškevičienė (2021). Teaching computational thinking in primary schools: Worldwide trends and teachers' attitudes. *Computer Science and Information Systems* 2021 OnLine-First Issue 00, 33–33. `https://doi.org/10.2298/CSIS201215033D`

Dasgupta, S. (2014). *It Began with Babbage: The Genesis of Computer Science*. Oxford University Press.

De Bruyckere, P., Kirschner, P.A., Hulshof, C. (2015). *Urban myths about learning and education*, London, Boston: Elsevier.

Denning, P.J., Rosenbloom, P.S. (2009). The profession of ITa – Computing: The fourth great domain of science. *Communications of the ACM*, 52(9), 27–29.

Denning, P. J., Tedre, M. (2019). Computational Thinking. The MIT press.

Dijkstra, E.W. (1972). The humble programmer. *Communications of the ACM*, 15(10), 859–866.

Department for Education (2021). *The National Curriculum in England*. Department for Education Government of UK, Crown, Cheshire. `www.gov.uk/government/collections/national-curriculum`

Directorate for Learning and Assessment Programmes (2019). SEC Syllabus: Computing.
`https://www.um.edu.mt/__data/assets/pdf_file/0017/292310/SEC09.pdf`

Education Scotland (2017). *Benchmark Technologies. education.*
`gov.scot/improvement/documents/technologiesbenchmarkspdf.pdf`

Erschov, A.P. (1972). *Communications of the ACM*. 15(7).

Ershov, A.P. (1981). Programming: The second literacy. *Microprocessing and Microprogramming*, 8(1), 1–9.

Gal-Ezer, J., Beeri, C., Harel, D. Yehudai, A. (1995). A High-School Program in Computer Science. *Computer*, 28, 10, 73–80.

Gal-Ezer, J., Harel, D. (1999). Curriculum and Course Syllabi for High-School Computer Science Program. *Computer Science Education*, 9(2), 114–147.

Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., Boyle, R., Drechsler, M., Mendelson, A., Stephenson, C., Ghezzi, C., Meyer, B. (2013). *Informatics Education: Europe Cannot Afford to Miss the Boat. Technical Report.* Association for Computing Machinery &, Joint Informatics Europe ACM Europe Working Group on Informatics Education, New York. 1–21 p.

Guzdial, M. (2015). *Learner-Centered Design of Computing Education: Research on Computing for Everyone (Synthesis Lectures on Human-Centered Informatics)*. Morgan & Claypool, 1st edition.

Harel, D. (1987). *Algorithmicsa – the Spirit of Computing*. Addison-Wesley.

Hauser, U., Hromkovič, J., Klingenstein, P., Lacher, R., Lütscher, P., Staub, J. (2020). *Einfach Informatik Rätsel und Spiele ohne Computer KG 1–2* (in German). Klett und Balmer.

Heintz, F., Mannila, L., Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. *Frontiers in Education Conference (FIE)*, 2016 IEEE (pp. 1–9).

Hromkovič, J. (2015). Homo Informaticus, *Bull. EATCS 115*.

Hromkovič, J. (2018). *Einfach Informatik. Daten 7–9* (in German). Klett und Balmer.

Hromkovič, J. (2018). *Einfach Informatik. Strategien Entwickeln. 7–9* (in German). Klett und Balmer.

Hromkovič, J. (2018). *Einfach Informatik. Programmieren 5–6* (in German). Klett und Balmer.

Hromkovič, J., Kohn, T. (2017). *Einfach Informatik. Programmieren 7–9* (in German). Klett und Balmer.

Hromkovič, J., Lacher, R. (2017a): How to convince teachers to teach computer science even if informatics was never a part of their own studies. *Bull. EATCS 123*.

Hromkovič, J., Lacher, R. (2017b). The computer science way of thinking in human history and consequences for the design of computer science curricula. *ISSEP 2017*, LNCS 10696, 3–11.

Hromkovič, J., Lacher, R. (2019). *Einfach Informatik. Lösungen Finden 5 / 6* (in German). Klett und Balmer.

Hromkovič, J., Lacher, R. (2021). *Einfach Informatik. 3 / 4* (in German). Klett und Balmer.

Hromkovič, J., Steffen, B. (2011). Why teaching informatics in schools is as important as teaching mathematics and natural sciences. *ISSEP 2011, LNCS*, LNCS 7013, 21–30.

Hubwieser, P., Armoni, M., Giannakos, M.N. (2015). How to implement rigorous computer science education in K-12 schools? Some answers and many questions. *Acm Transaction On Computing Education*, 15(2), 1–12.

Hubwieser, P., Schubert, S., Armoni, M., Brinda, T., Dagienė, V., Diethelm, I., Giannakos, M.N., Knobelsdorf, M., Magenheim, J., Mittermeir, R. (2011). Computer science/informatics in secondary education. In: *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Educationa – Working Group Reportsa – ITiCSE-WGR '11.*

Hubwieser, P., Armoni, M., Giannakos, M.N., Mittermeir, R.T. (2014). Perspectives and visions of computer science education in primary and secondary (K-12) schools. *Transaction on Computing Education*, 14(2), 7(1), 7–9.

Kert, S.B., Kalelioglu, F., Gulbahar, Y. (2019). A Holistic Approach for Computer Science Education in Secondary Schools, *Informatics in Education*, 18(1), 131–150,
`https://doi.org/10.15388/infedu.2019.06`

Kirschner, P.A., Sweller, J., Clark, R.E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2), 75–86.

Knuth, D.E. (1972). Ancient Babylonian algorithms. *Communications of the ACM*, 15(7), 671–677.

Kong, S.C. (2017). A framework of curriculum design for computational thinking development in K-12 education. *Journal of Computers in Education,* 3(4), 377–394.

Lau, W. (2018). *Teaching Computing in Secondary Schools: a Practical Handbook*. Taylor & Francis.

Man-Keung, S. (2000) The ABCD of Using History of Mathematics in the (Undergraduate) Classroom. In: V. Katz (ed.). *Using History To Teach Mathematics, Mathematical Association of America*. Washington DC, 3–9.

National Council for Curriculum and Assessment (2018). Computer Science Curriculum Specification. `https://www.curriculumonline.ie/Senior-cycle/Senior-Cycle-Subjects/Computer-Science`

Nygaard, K. (1986). Program development as a social activity. In: *Proceedings of the IFIP 10th World Computer Congress, Information Processing*. Dublin, Elsevier science publishers, 198–198.

Pacheco, J.A. (2012). Curriculum studies: What is the field today? *Journal of the American Association for the Advancement of Curriculum Studies*, 8, 1–18.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: BasicBooks.

Papert, S. (1993). *The Children's Machine: Rethinking School in the Age of the Computer.* New York: BasicBooks.

Papert, S., Harel, I. (1991). *Constructionism*. New York: Ablex publishing.

Passey, D. (2017). Computer science in the compulsory education curriculum: Implications for future research. *Education and Information Technologies,* 22(2), 421–443.

Piaget, J. (1950). *The Psychology of Intelligence*. Cambridge, MA: Harvard university press.

Seehorn, D., Carey, S., Futschetto, B., Lee, I., Moix, D., *et al.* (2011). *CSTA K-12 Computer Science Standards*. New York: ACM/CSTA.

Sentance, S., Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, 22(2), 469–495.

Singh, S. (1999). *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. London, 143–189.

Smith III, J.P., diSessa, A.A., Roschelle, J. (1994). Misconceptions Reconceived: a Constructivist Analysis of Knowledge in Transition. *Journal of the Learning Sciences*, 3(2), 115–163, `https://10.1207/s15327809jls0302_1`

Swetz, F. (2000). Problem Solving from the History of Mathematics. In: V. Katz (Ed.). *Using History To Teach Mathematics, Mathematical Association of America*. Washington DC, 59–65.

Sysło, M.M., Kwiatkowska, A.B. (2015). Introducing a new computer science curriculum for all school levels in Poland. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 141–154.

Tedre, M. (2014). *The Science of Computing: Shaping a Discipline*. CRC Press / Taylor & Francis, New York, NY, USA.

Webb, M., Davis, N., Bell, T., Katz, Y., Reynolds, N., Chambers, D., *et al.* (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Education and Information Technologies*, 22(2), 445–468.

Williams, M.R. (1985). A *History of Computing Technology*. Prentice-Hall, New Jersey, USA, 1st edition.

V. Dagienė is a principal researcher at Vilnius University Institute of Data Sciences and Digital Technologies. She has published over 300 scientific papers and 60 text-books in informatics education area. She is an Editor-in-Chief of two international journals "Informatics in Education" and "Olympiads in Informatics". She coordinated over 50 national and international projects on teaching coding to children and young-sters, technology enhanced learning, teacher training, development of education soft-ware, and software localization. She is acknowledged by honorary gold medal for contributions to school Informatics in Europe (ETH Zurich, 2011), the Informatics Europe Best Practices in Education Award (2015), Ada Lovelace Computing Excel-lence Award (2016), and the Cross of the Knight of the Order of the Lithuanian Grand Duke Gediminas (2016).

J. Hromkovič is professor of Information Technology and Eduation at the Department of Computer Science at ETH Zurich since January 2004. His research and teaching interests focus on informatics education, algorithmics for hard problems, complex-ity theory with special emphasis on the relationship between determinsm, random-ness, and nondeterminism. One of his main activities is writing textbooks which make complex recent discoveries and methods accessible for students and practitioners, and so contributing to the speed up of the transformation of new paradigmatic research results into educational folklore. In order to introduce the subject informatics to the school education, he founded the Centre for Computer Science Education in 2005. He is responsible for the master program Lehrdiplom Informatik at ETH devoted to the education of computer science teachers.

R. Lacher is the operational manager of the ABZ (Center for Computer Science Edu-cation at ETH) since 2014 and works at UBS (a global finance institute) since 2001 as an Operational Risk Manager. Regula has completed three educations: She is a ge-ographer (master in natural sciences of the University of Zürich, 1990), a Quality Manager (accredited by the European Organization for Quality Management in 1993) and a physics laboratory technician (apprenticeship plus professional baccalaureate, in 1982). All these different backgrounds together with the interest in the concept of constructivism proved to be useful in her work in informatics education. Regula en-joys contributing to computer science education as a co-author to a series of textbooks, research papers and creates tasks for the Informatics Beaver Competition.