Valentina Dagiene
Arto Hellas (Eds.)

LNCS 10696

Informatics in Schools
Focus on Learning Programming

10th International Conference on Informatics in Schools:
Situation, Evolution, and Perspectives, ISSEP 2017
Helsinki, Finland, November 13–15, 2017, Proceedings

Springer

Valentina Dagiene · Arto Hellas (Eds.)

Informatics in Schools

Focus on Learning Programming

10th International Conference on Informatics in Schools:
Situation, Evolution, and Perspectives, ISSEP 2017
Helsinki, Finland, November 13–15, 2017
Proceedings

Springer

Editors
Valentina Dagiene
Mathematics and Informatics
Vilnius University
Vilnius
Lithuania

Arto Hellas
University of Helsinki
Helsinki
Finland

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-71482-0 ISBN 978-3-319-71483-7 (eBook)
https://doi.org/10.1007/978-3-319-71483-7

Library of Congress Control Number: 2017960853

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Preface

The International Conference on Informatics in Schools: Situation, Evolution and Perspectives (ISSEP) celebrated its 10th anniversary in Helsinki, the capital of Finland. The year 2017 also marked the anniversary of Finland: One month after the conference, on December 6, Finland celebrated its 100th year of independence.

ISSEP is a forum for researchers and practitioners in the area of informatics education, in both primary and secondary schools. The conference provides an opportunity for educators and researchers to reflect upon the goals and objectives of this subject matter, its curricula, various teaching and learning paradigms and topics, as well as the connections to everyday life – including the various ways of developing informatics education in schools. This applies in an even larger extent to the didactics of informatics. ISSEP provides an opportunity to take stock of developments in the field and on the extent these developments are relevant for everyday school life.

This conference brings together delegates to address pressing issues in informatics education. On the basis of these challenging areas, participants consider how to cope with future demands of education in informatics and IT-related subjects. Teachers, those working in teacher education and in-service training as well as in school administration come to participate in workshops and to share their experience or vision on an international platform.

The ISSEP conference started in 2005 in Austria, Klagenfurt University (now Alpen-Adria University of Klagenfurt), where the conference was a part of the celebration of 20 years of teaching informatics in Austrian secondary schools. The following year, ISSEP took place in Vilnius, Lithuania, celebrating 20 years of informatics in Lithuanian secondary schools. The ISSEP conference expanded: In Vilnius there were over 200 researchers from 37 countries, and an additional day for local teachers was established and had over 250 attendants from all over Lithuania. In 2008, ISSEP was in Torun, Poland, and in 2010, ISSEP was organized in Zürich. Then followed Bratislava, Slovakia, in 2011, Oldenburg, Germany, in 2013, Istanbul, Turkey, in 2014, Ljubljana, Slovenia, in 2015 and Münster, Germany, in 2016.

In these proceedings, all research papers were blind reviewed by at least three reviewers. A final selection phase was conducted by the co-chairs who reviewed all reviews and recommendations before making their final decisions. From the 41 submitted articles, 18 were selected for presentation and inclusion in the proceedings.

This year, participants at ISSEP discussed a broad range of themes ranging from making informatics accessible to visually impaired students and computational thinking to context- and country-specific challenges as well as teacher development and training. The conference hosted several workshops where teachers and researchers were familiarized with, e.g., robotics and teaching programming as well as unplugged activities for computational thinking. Preliminary work and new ideas were presented as posters, creating a unique opportunity for the participants to discuss the ideas in a supportive environment. ISSEP also hosted CSERC (Computer Science Education Research Conference) as a special track at the conference.

October 2017

Valentina Dagienė
Arto Hellas

Organization

Conference Chairs and Co-chairs

Valentina Dagienė	Vilnius University, Lithuania
Arto Hellas	University of Helsinki, Finland

Program Committee

Erik Barendsen	Radboud University Nijmegen and Open Universiteit, The Netherlands
Andreas Bollin	University of Klagenfurt, Austria
Andrej Brodnik	University of Ljubljana, Slovenia
Michalis Giannakos	Norwegian University of Science and Technology, Norway
David Ginat	Tel Aviv University, Israel
Bruria Haberman	Holon Institute of Technology, Tel Aviv, Israel
Juraj Hromkovič	Swiss Federal Institute of Technology Zurich, Switzerland
Peter Hubwieser	Technical University of Munich, Germany
Petri Ihantola	Tampere University of Technology, Finland
Ivan Kalas	Comenius University, Slovakia
Tiina Korhonen	University of Helsinki, Finland
Peter Micheuz	University of Klagenfurt and Gymnasium Völkermarkt, Austria
George A. Papadopoulos	University of Cyprus, Cyprus
Sergey Pozdniakov	Saint Petersburg Electrotechnical University, Russia
Ralf Romeike	University of Erlangen (FAU), Germany
Maciej M. Syslo	University of Wroclaw, Poland

Local Organizing Committee

Antti Laaksonen	Aalto University, Finland
Juho Leinonen	University of Helsinki, Finland
Leo Leppänen	University of Helsinki, Finland

Contents

Keynote Paper

The Computer Science Way of Thinking in Human History and Consequences for the Design of Computer Science Curricula

Juraj Hromkovič(✉) and Regula Lacher

Department of Computer Science, ETH Zurich, Zurich, Switzerland
juraj.hromkovic@inf.ethz.ch

Abstract. Teaching computer science offers more than algorithmic thinking (or more general and as recently presented: computational thinking). To understand this claim, one has to have a more careful look at the development of human culture, science, and technology. This helps not only to recognize that the computer science way of thinking was crucial for the development of human society since anyone can remember, but it helps to make a good choice of topics for sustainable computer science education in the context of science and humanities. This leads to the creation of textbooks that do not focus on particular knowledge for specialists, but offer serious contributions in the very general framework of education.

1 Introduction

The recent technological development has led to a situation where many of our children will be working in a profession that does not yet exist, and they will be working on the solution of problems that have not even materialized yet. To prepare them for this, we need to change our approach to teaching. Our fundamental principle for designing educational curricula for any subject is the following:

Do not teach the final products of science, technology, and humanities, and do not consider it the highest goal to train to successfully apply them. For the latest knowledge may be found outdated with time. Teach the process of discovering new knowledge, teach the need to search for new solutions, teach the ways of collecting experience and formulating hypotheses, teach the ways of verifying hypotheses, teach how others can be convinced about the truth discovered, teach the constructive way of thinking in order to create new products and finally new technology, and teach the processes of testing and improving the products of our work.

We consider this principle to be the base of the so-called "critical thinking," becoming the fundamental goal of teaching science in leading universities. It is based on the understanding of the development of human societies, which was

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 3–11, 2017.
https://doi.org/10.1007/978-3-319-71483-7_1

only possible with the evolvement of science, humanities, law, and technology. We should teach those aspects that have the biggest impact on the development of our way of thinking. The whole of human history can be seen as a process of developing research instruments to understand the world around us and to find solutions for our problems. The goal of this article is to look at computer science as an integral part of science since ever and to use this view to recognize the main contributions and consequently to design how to teach them in the context of other educational subjects.

2 History

2.1 Computer Science, Languages, and Writing

Writing was developed in many cultures, and it became one of the most important human technologies. The oldest known writing was developed in Mesopotamia more than 5 000 years ago. The motivation for this development would be recently considered a "pure computer science task": The people of Mesopotamia needed to save (store) and process (update) tax and property data of some 1 000 000 people living between Euphrates and Tigris. The starting point during the development of a writing is the choice of symbols, called an alphabet, and then the representation of data (numbers, words, texts) as finite sequences of symbols. In this framework, we can speak about "coding" of information. The development of number representations is a wonderful story about different systems being in competition with each other (each to each) for many years with respect to understandability (transparency), the length of the description, and the efficiency of performing arithmetic operations on them.

As more and more people learned to read, the need to keep some of that written information secret, i.e., understandable for a selected group of people only, was formulated. The oldest traces of attempts to decrease readability are some 3 500 years old, where the order of some symbols was exchanged in Mesopotamia. 2 500 years ago, different secret writings were used in Palestine, India, China, Egypt and Greece. They used concepts of mixing the order of symbols and exchanging symbols (coding of symbols by other symbols) which are still used in modern cryptographic protocols.

The above mentioned activities are all strongly related to the expertise of computer scientists, some of them are even considered as a proper part of computer science, e.g., cryptography. Indeed, computer scientists are true experts in developing writings for specific purposes. Some examples for teaching this topic can be found in [1,5,8].

Another example is compression, which is strongly related to measuring the information content of texts. Coding texts as short as possible, without losing any information, in order to shorten communication messages, was developed even before computer science was established as an independent discipline (see [1,8] for an example of a teaching sequence).

Self-correcting codes [10] are robust writings which automatically recognize and correct mistakes in texts. Current e-commerce cannot work without such codes because any small misprint could cause a wrong money transfer or wrong online order.

Later on, computer scientists developed numerous writings for data representations for different purposes. Some focused on performing selected data operations (data structures), others focused on creating databases (information systems) with a very fast search capability (see [1,8]).

Driven by the construction of (physical) computers, the binary alphabet and many ways of representing data by sequences of bits were introduced[1] (see [1,8] for a teaching sequence).

Another strong relationship between languages and computer science is in the development of programming languages. Programming languages are languages with very well defined syntax and semantics, and looking at the development of formal language theory as a product of computer science one observes a big intersection with the development of linguistics. The formal concept of a grammar, context-sensitivity and context-freeness are all concepts developed in cooperation with linguistics (see [2] for a textbook for high schools on this topic).

2.2 Computer Science and Mathematics

If one wants to understand what mathematics is about, it is helpful to view mathematics as a language with the following two features:

(i) All sentences have an unambiguous meaning for everybody mastering the language of mathematics.
(ii) All argumentation in mathematics is verifiable.

The demand to create a language such as mathematics was obvious. There is no objectivity in science if one cannot communicate in such a way that everybody interprets each claim in the same way. Additionally, without verifiable argumentation the notion of truth is very relative (see [11] for more details).

This is how mathematics, together with experiments, became the main research instrument of humans. The relation with computer science lies mainly in abstraction and in the development of algorithms as a method for solving different tasks.

Computer science uses digital technology and so it cannot omit the formal language of mathematics that codes everything as sequences of symbols of a fixed alphabet. Abstraction in computer science means to be able to represent objects and real situations by mathematical concepts and finally as sequences of bits. All this is a part of the expertise of computer scientists as well. The contribution of computer science is using the formal language of mathematics to describe computation processes and to investigate them.

[1] It was technically much easier to build processors executing operations using binary numbers than using decimal numbers. Also, it was much easier to build stable physical systems for storing information with two states only (interpreted as 0 and 1) than developing a computer memory based on many states.

Algorithm is a key word of computer science, but this term is not new. Already Euclid formulated algorithms in his “Elements” [4], among them the famous *Euclid’s algorithm.* The term “algorithm” is due to al-Khwarizmi, who wrote a book about Indian digits around the year 825. Humans tried to develop algorithms as long as anyone can remember. People generated knowledge in order to understand the world around them and especially to apply this knowledge to develop “procedures” to reach their goals. This was key for the development of the first human societies because it allowed jumps in performance and efficiency increase. The point is that becoming an expert in performing a previously developed procedure was much easier than to learn to develop such procedures or to discover new facts. One nice example from ancient Greece is the Theorem of Pythagoras that became a procedure applied in the construction of buildings. We know due to Pythagoras that $3^2 + 4^2 = 5^2$ and that the triangle with sides of length 3, 4, and 5 units always contains a right angle. Therefore, to create a right angle, it is sufficient to take 3 ropes of the corresponding lengths (3, 4, and 5 units) to construct the right angle. There is a huge gap between the qualification of creating this triangle and the intellectual potential to discover the Theorem of Pythagoras, not to mention the capability to prove why it is true. In this way, many technologies became available to big parts of society in spite of the fact that only few people had the expertise to understand them or even to be able to develop them.

The above was also the reason why Leibniz described mathematics as a science of automation of human work. The idea of Leibniz was to translate the problems of the real world into the language of mathematics and then to solve them by formal calculations. His first dream was to develop a calculus for logical argumentation similar to formal arithmetics. This dream was satisfied 200 years later when logic was developed and became the formal system for automatic calculation of the correctness of mathematical proofs. The second dream of Leibniz was to create a type of mathematics in which all real problems can be described and solved.

During the time of technical revolution, Hilbert [7] specified Leibnizs second dream more precisely. He considered a problem as a collection of potentially infinitely many problem instances. An algorithm for solving the problem was therefore a method that was able to calculate the correct solution for any of the instances of the problem. The dream of Hilbert included finding an algorithm for each problem that can be formulated in the language of mathematics. However, in 1930, Kurt Gödel [6] proved that there exist claims in mathematics for which the truth cannot be decided inside of mathematics. In other words, Kurt Gödel proved that the description power of the language of mathematics is stronger than its argumentation power. One can formulate claims in mathematics for which there exist no proofs whether they are correct or not. This finding was the nucleus that started the founding of “computability” as the first sub-discipline of computer science with the goal to classify problems into algorithmically solvable and algorithmically unsolvable ones. The related formal definitions of the term “algorithm” offered by Turing [18] and Church [3] are now considered as the birth of computer science.

There are so many intersections between mathematics and computer science in the subsequent development, that we do not try to list them here.

2.3 When and Why Computer Science Became an Independent Discipline

Computer science became a distinguished discipline when the following two conditions were satisfied:

1. One was able to develop algorithms that were unambiguously described in the sense that no human intellect (no expert knowledge, no interpretation) was needed to execute them.
2. (Digital) technologies were available that enabled the delegation of execution (of precisely described procedures) to machines.

Hence, the two main components of computer science are:

1. To use mathematics to develop – in cooperation with specialists in the respective areas of science and practice – algorithms to automate more and more human activities. Many mathematicians consider this (or big parts thereof) a part of mathematics.
2. To contribute to the development of hardware and software technologies, allowing the automation of more and more complex tasks and increasing the performance of computing technologies. A part of this job is the development of programming languages enabling humans to communicate with machines to control them, and to instruct them to execute different activities. This component of computer science is assigned to engineering.

There is no strict boundary between these two and many tasks require the expertise of both components.

3 How to Teach Computer Science

One of the biggest mistakes in the past was to teach the use (application) of digital technologies (e.g., ECDL) instead of the foundations of computer science. If the focus is set on teaching the latest developments in computer science concepts and digital technology, this mistake is continued. Computer science has to learn from physics which typically is taught by following the historical development of our understanding of the physical world. Trying to teach quantum mechanics or relativity theory is of little value for most pupils and tends to end in frustration for most. One has to start with Galileo Galilei and Newton in order to get a first intuition about how the physical world is functioning and in order to learn how to design experiments that enable to verify hypotheses or disprove them. One has to follow the genesis of science, learn from mistakes and especially learn to develop research instruments, namely experiments and mathematics.

If we want to enable future generations to contribute to science and technology, we have to follow the development of basic concepts of computer science step by step. In this way we can contribute to

(i) better understanding mathematics and languages and their development,
(ii) understanding computer science as a research instrument in science and humanities, and
(iii) understanding our technical world and being able to control and develop it.

Rather than dedicating the space of this article to explaining in detail the spiral curriculum of teaching computer science developed at the Center for Computer Science Education of ETH, we give some selected examples to illustrate the approach we followed during the development of our textbooks.

3.1 Data Representation

The first goal is to learn that there is freedom of choice regarding the alphabet used to code information. One can start with different number representations and compare them with respect to transparency, length of the description, and efficiency in working with them. One learns to recognize what they have in common and what are the differences. Finally, one does not only learn the binary representation of numbers and its advantages and disadvantages, but also to design new number representations (see [1,8] for teaching examples).

Data compression is another topic that is best taught following the historical development. One is not allowed to push to learn the best compression method currently known. The focus must be on learning from examples in order to be able to propose own compression methods and to compare them with other methods presented. One can distinguish between compression methods without and with loss of information. In the latter case, one can play with the tradeoff between the size of the data representation achieved by a compression on one hand, and the amount of information lost on the other hand (see [1,8] for teaching examples).

Teaching self-verifying codes is very well motivated by the use of digital technologies. There are plenty of codes that can be presented as examples and there is a lot of freedom to design own codes with some required properties. Starting with suitable examples, the main idea is designing efficiently computable and short self-verifying codes. This can be successfully explained without any sophisticated mathematics. Again, the main goal is to learn to design own codes with required properties for small sets of data.

Cryptography as a theory of secret writings is probably the most gratifying topic of data representation. One is not allowed to focus on the correct usage of current cryptographic protocols based on deep knowledge of algebra and number theory. Again, one has to follow the historical development step by step. One should present historic examples of secret writings and cryptosystems, let the pupils find their weaknesses and use these to break them. Following the development of cryptosystems, it becomes natural to understand the concept of the security of a cryptosystem up to the current public-key cryptography. Again, the main achievement is the ability to design new cryptosystems and to break them, and so to discover a demand to design better and better cryptosystems (see [1,5,8] for teaching examples).

There are many other data representation topics. An important one deals with suitable data representation for building databases or for special algorithmic tasks. A key point is that there are many levels of deepness and one can create a challenging spiral curriculum for this topic with many creative and constructive tasks.

3.2 Algorithm Design

Algorithms as methods that solve all infinitely many instances of a problem are a very abstract concept and should not be introduced too early. For pupils up to class 4, one is allowed to find solutions to particular problem instances and to develop general strategies on a very intuitive level only.

But even for older pupils, for classes 7 to 9, we do not recommend to teach specialized, concrete algorithms that do not work if one changes the specification of the problem just a little bit. The idea is to teach robust strategies that successfully work for many problems. Our teaching approach starts with training to recognize feasible solutions for a new problem instance, to represent it as a sequence of symbols and to be able to list and count all feasible solutions for concrete instances. Then one can introduce criteria to judge the quality of the solutions in order to choose an appropriate one or even the best feasible solution. This imposes a lot of understanding of the problem setting and introduces the trouble with an exponential explosion of the number of solutions. The next step is to teach to make compromises when looking for a good solution in situations where one cannot list and compare all solutions because there are too many. Again, simple robust strategies as greedy or local algorithms can be explained and discussed. It is interesting to discover that, for some problems, they compute optimal solutions, and for other problems they may sometimes fail by offering solutions that are far from optimal. There is a lot of creative work in discovering problem instances for which certain strategies behave poorly.

Again, this topic is excellent for developing a spiral curriculum starting with collecting experience by solving simple, particular problem instances, and finishing by developing clever algorithms for concrete problems in high school.

3.3 Programming

For a more detailed presentation of this topic, see [9,12–17]. The starting point is not to teach a programming language, but rather to combine problem solving with the communication of the designed algorithm to the computer. Start with as few instructions as possible and force the pupils through modularity to introduce new instructions and to "teach" the computer to understand them. This way, on one hand, the pupils learn how to develop a language in order to make the communication more efficient, and on the other hand they learn modular design, which is one of the most fundamental concepts of engineering. Simultaneously, they follow the genesis of the development of programming languages to some extent and therefore naturally do not see programming languages as final products that must be taken as they are.

The main focus is on the following contributions:

1. To learn problem solving by building up experience with solving particular problem instances and finally developing solving strategies.
2. To learn to describe the strategies (algorithms) developed first in a meta language and then in exact programming language (learning fundamental concepts of programming).
3. To discover the demand of introducing new words to a language in order to simplify communication and to increase the understandability.
4. To learn the modular design method as a fundamental concept of engineering and to be able to use it in various situations.
5. To learn to verify by reasoning and to test by experiments the products (programs) of own making.

4 Conclusion

To prepare our children for the future, we need to change our approach to teaching fundamentally: Rather than teaching the current products of science, technology and humanities, we should teach the process of making discoveries. We should teach collecting experience and data, formulating and verifying hypotheses, how to convince ourselves and others about the truth discovered, how to think in a constructive way that leads to the creation of new products and new technology, and we should teach the processes of testing and improving the products of our work. The example of Pythagoras' Theorem illustrates not only how a product of science has been applied in processes, but also shows the difference of skills needed for the discovery of new knowledge and the application thereof.

When applying this strategy for teaching computer science, one obtains the following principles:

1. Do not teach computer science as an isolated subject, but teach computer science as a part of science and technology offering a deep contextual view. Take care on contributing to knowledge transfer to other disciplines. Build and use bridges to the development of languages and mathematics.
2. Do not teach the latest products of IT and the latest scientific discoveries. Follow the genesis of fundamental concepts and improve them step by step. Create the need for these concepts and their improvements and discover the new ideas by doing experiments, proposing and verifying solutions and evaluating their feasibility and quality. Try to discover, for all age groups, to which extent which concepts are feasible and how to apply learning by doing to master them. Computer science topics such as data representation, programming, and algorithm design have many levels of deepness, therefore lending themselves perfectly for a spiral curriculum.
3. Teach to view IT as an enabling technology. Teach to control computers by programming and to automate well understood activities in order to make society more efficient. Teach to work in a constructive way, to test the functionality of own products, and the modular design approach. Simply teach the way of thinking and working of technical disciplines (engineering).

References

1. Bell, T., Witten, I.H., Fellows, M.: Computer Science Unplugged. Creative Commons Attribution 2.0 (2015)
2. Böckenhauer, H.-J., Hromkovič, J.: Formale Sprachen: Endliche Automaten, Grammatiken, Lexikalische und syntaktische Analyse. Springer, Wiesbaden (2013). https://doi.org/10.1007/978-3-658-00725-6
3. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
4. Euclid: Elements, books I-XIII
5. Freiermuth, K., Hromkovič, J., Steffen, B., Keller, L.: Einführung in die Kryptologie: Lehrbuch für Unterricht und Selbststudium. Springer, Wiesbaden (2014). https://doi.org/10.1007/978-3-8348-2269-7
6. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik, pp. 173–198 (1931)
7. Hilbert, D.: Mathematische Probleme. Nachrichten von der königl. Gesellschaft der Wissenschaften zu Göttingen. Mathematisch-Physikalische Klasse, pp. 253–297 (1900)
8. Hromkovič, J.: Einfach Informatik. Datendarstellung. Klett (2018, to appear)
9. Hromkovič, J., Kohn, T.: Einfach Informatik. Programmieren. Klett (2018, to appear)
10. Hromkovič, J., Keller, L., Komm, D., Serafini, G., Steffen, B.: Entdeckendes lernen am beispiel fehlerkorrigierender codes. Login **168**, 50–55 (2011)
11. Hromkovič, J.: Homo Informaticus: why computer science fundamentals are an unavoidable part of human culture and how to teach them. Bull. EATCS **115**, 112–122 (2015)
12. Hromkovič, J., Kohn, T., Komm, D., Serafini, G.: Combining the power of python with the simplicity of logo for a sustainable computer science education. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 155–166. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_13
13. Hromkovič, J., Kohn, T., Komm, D., Serafini, G.: Examples of algorithmic thinking in programming education. Olympiads Inform. **10**, 111–124 (2016)
14. Hromkovič, J., Kohn, T., Komm, D., Serafini, G.: Algorithmic thinking from the start. In: Bulletin of the EATCS 121, The Education Column (2017)
15. Hromkovič, J., Serafini, G., Staub, J.: XLogoOnline: a single-page, browser-based programming environment for schools aiming at reducing cognitive load on pupils. In: Dagiene, V., Hellas, A. (eds.) ISSEP 2017. LNCS, vol. 10696, pp. 219–231. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71483-7_18
16. Hromkovič, J., Steffen, B.: Why teaching informatics in schools is as important as teaching mathematics and natural sciences. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 21–30. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_3
17. Serafini, G.: Programmierungerricht für Kinder und deren Lehrpersonen: Unterrichtsmaterialien, didaktische Herausforderungen und konkrete Erfahrungen, GI-Edition. Lecture Notes in Informatics, vol. 249, pp. 267–272 (2015)
18. Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. **42**(2), 230–265 (1936)

Accessibility, Visualizations and Physical Computing

Adolescent and Adult Student Attitudes Towards Progress Visualizations

Onni Aarne(✉), Petrus Peltola, Antti Leinonen, Juho Leinonen, and Arto Hellas

University of Helsinki, Helsinki, Finland
{onni.aarne,petrus.peltola,antti.leinonen, juho.leinonen,arto.hellas}@helsinki.fi

Abstract. Keeping students motivated for the duration of a course is easier said than done. Contextualizing student efforts with learning progress visualizations can help maintain engagement. However, progress can be visualized in many different ways. So far very little research has been done into which types of visualizations are most effective, and how different contexts affect the effectiveness of visualizations. We compare the effects of two different progress visualizations in an introductory programming course. Preliminary results show that older students prefer a visualization that emphasizes long-term progress, whereas younger students prefer a visualization that highlights progress within a single week. Additionally, students perform better and are more motivated when their visualization matches their age group's preferred visualization. Possible explanations and implications are discussed.

Keywords: Progress visualization · Student engagement · Growth mindset · Motivation

1 Introduction

The effort that we expect from our students can be overwhelming to the extent that students temporarily lose track of their long-term goals and original motivations – seeing the forest from the trees can be hard. Progress visualizations can be a valuable tool for showing students the metaphorical forest, and helping them focus on the work that they are doing. Certain visualizations can improve students' performance in a measured task; for example, a progress bar that measures a specific activity can increase students' engagement with that activity [8]. These types of motivational tools can be important in labour-intensive courses, such as introductory programming courses, and can provide additional incentives for the students to complete the tasks they are given.

Learning progress visualization tools intended for students themselves have mostly been researched as part of broader attempts at gamifying education. However, this research rarely looks at specific elements of gamification with

O. Aarne and P. Peltola contributed equally to this work.

any rigor. In this work, we compare two types of progress visualizations that emphasize progress at different course granularities, and study whether specific visualizations are more suitable for specific course sub-populations. More specifically, we explore how adult and adolescent students respond to visualizations that highlight weekly progress and course-long progress.

This article is structured as follows. First, in Sect. 2 we provide an overview of students' orientations and mindsets as well as discuss point and progress visualizations. Then, in Sects. 3 and 4 we explain the details of the study and its results, respectively. In Sect. 5 we discuss the results and lastly, in Sect. 6 we conclude this work and outline possible future research.

2 Related Work

In this section, we visit two relevant streams of research. First, we briefly discuss self-regulation in adults and adolescents. Then, we discuss previous work on motivational visualizations.

2.1 Students' Mindsets as Factors of Success

This stream of research aligns with human self-regulatory processes, which broadly encompasses the ability of an individual to control and evaluate their behavior during learning. Caprara et al. [3] studied how changes in self-regulatory efficacy affects students' grades. They found that as the efficacy declined, the students' grades lowered. This in turn led to a higher chance of dropping out from studies. Their dataset consisted of 412 students, and the study was conducted as a longitudinal study over a period of ten years – the participants were followed between the ages of 12 and 22.

In her research, Dweck observed that individuals who believe that intelligence is fixed, or that success is based on innate ability, are more likely to stop working during setbacks or when struggling than those who believe that intelligence can grow through effort [4]. Dweck popularized this belief using the term "mindset". Later, different types of interventions that emphasize growth of knowledge over fixed knowledge have been developed.

For example, O'Rourke et al. [12] developed two versions of a math video game called "Fractions" to study the effects of growth mindset. Their experimental version of the game promoted students' learning with commentary that praised their intelligence, whereas the control group got more neutral feedback praising their mathematical skills. They found that the experimental group were more persistent as they played for a longer period of time and completed more levels than the control group. They also started to display behavior coherent with a growth mindset by using strategies they learned from the game. Another result of the study was that lower performing students were more encouraged to keep playing in the experimental version than in the control version.

The goal is to provide the students visualizations that best support self-regulation. This in turn would lead the students to the correct mindset which prevents quitting.

Steinberg et al. [15] found that young adults' future orientations and delay discounting continued to develop from childhood all the way into their mid-twenties. The research compared 10 to 30 year old individuals and indicated, when compared to their counterparts, that younger individuals are more likely to choose a smaller reward sooner than a larger reward later. This supports our study design.

2.2 Point and Progress Visualizations

Overall, point and progress visualizations fall under multiple umbrellas. They are an integral part of gamification research that seeks to bring game-like elements into educational contexts as well as more generic educational principles of transparency where students' progress should be made visible.

Leppänen et al. [8] studied the effect of progress visualization on student engagement. They found that, at least initially, a simple progress bar significantly increased the quantity of completed exercises. Their system displayed the bar at the bottom of the screen, and the bar became full when the user had answered three consecutive multiple choice questions correctly. However, the arbitrary nature of this goal meant that their bar simply created an illusion of progress – which was sufficient for increasing students' effort – rather than depicting anything more meaningful.

Loboda et al. [9] developed and studied a form of open learner modelling which is comparable to our progress visualizations, though more clearly split up by subject. However, in their study, the use of the system was entirely optional so any apparent effect might simply be due to better students being more likely to use the system.

Another study of open learner modelling was conducted by Bull et al. [2]. They used multiple visualization types for the same student, and the students could choose which visualizations were visible to them. The available visualizations included skillometers, tables, stars, smileys, gauges, histograms, networks, radar plots, word clouds and treemaps. These visualizations had information about students' knowledge and skills on the subject. They found that students prefer simpler visualizations such as skillometers and stars over complex visualizations such as treemaps and word clouds.

Santos et al. [14] studied similar use of visualizations. Their research had a few differences from ours. Their study was iterative. In the first phase they collected feedback from teachers. The teachers valued simplicity and intuitivity, which is in line with Bull et al.'s results [2]. After modifications between evaluations, they provided the visualizations to students. They found that their demographic, which consisted of 18 to 20 year old students, valued the visualizations. Another difference to our research was that they showed the students their peers' progress, which the students found positive.

Visualizations have also been used in education to present students' progress to teachers [5,7]. Park and Jo [13] studied different usages of learning analytics dashboards for students, teachers or both. Based on the information they collected from previous studies of dashboards, they developed their own dashboard for both students and teachers, with the focus on students' self-regulation in the variables shown in the dashboard. They developed their dashboard iteratively similarly to Santos et al. [14] by interviewing students and improving the dashboard based on the feedback. Their dashboard also allowed students to see each other's progress.

Gamification in education [6,10,11] utilizes visualization of students' progress. For example Holman et al.'s [6] learning management system GradeCraft shows students their course scores as a progress bar. Additionally, they can get badges as they learn new concepts which are also shown in the progress bar. GradeCraft shows useful visualizations for teachers as well. They can see which badges have been completed, and which are in progress. They also see how the students predicted their success during the course and if their progress corresponds the prediction.

3 Methodology

In this section, we first explain the context of our study in Sect. 3.1. Then, we describe the survey the students were given and the students' answers, i.e. the data for our study in Sects. 3.2 and 3.3. Lastly, our research questions are outlined in Sect. 3.4.

3.1 Context

The data for this study comes from a CS1 course series split into two seven-week courses conducted in the spring of 2016 at the University of Helsinki. The course series is an introduction to object oriented programming with Java with content similar to most CS1 courses globally. Since most CS majors take these courses in the fall, our sample mostly consists of non-CS majors.

The course uses a blended online material with interspersed theory sections and assignment sections. For the purposes of this study, a visualization component was added to the online material.

In the study, the students were randomly divided into two groups, which were shown separate visualizations of their progress. One group was shown a series of bars that corresponded to the percentage of completed exercises for each week of the course, and the other group was shown a line plot that emphasized the growth of students' skills throughout the course. The visualization aggregated students' scores over the course, showing a steady growth as students' progressed throughout the course.

From this point onward, we refer to the visualization with the weekly bars as the *Weekly visualization*, and the visualization with an emphasis on course-long growth and effort as the *Growth visualization*. Two examples of the Weekly visualization are shown in Fig. 1, while Fig. 2 shows two examples of the Growth visualization.

The visualization was implemented as a floating element that was constantly available in the lower right corner of the course material. As the students worked on the course, they were able to hide the visualization temporarily. Regardless of whether the student had previously hidden the visualization, the visualization was visible to the student when the material page was opened again.

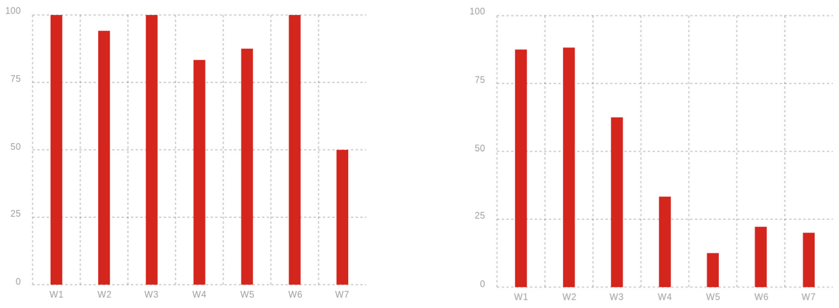


Fig. 1. Examples of the Weekly visualization, where the percentage of completed assignments for each week of the course are shown as a bar chart. The student on the left did fairly well, whereas the student on the right could not quite keep up but did not give up.

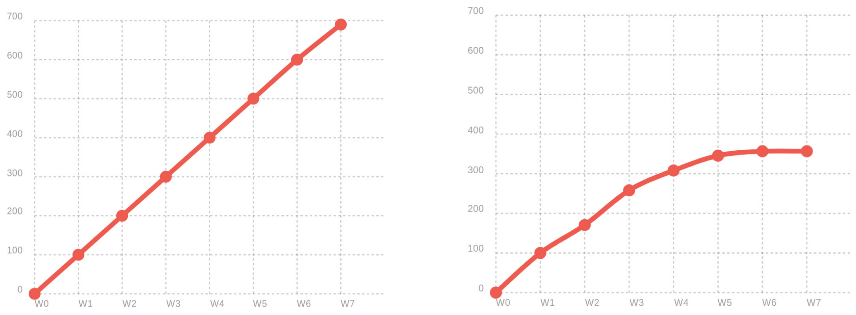


Fig. 2. Examples of the Growth visualization where aggregate assignment score is displayed on the x axis and time is shown on the y-axis. On the left is a perfect score, and on the right the student slowly fell behind and gave up.

3.2 Survey

At the start of the final exam of the course, the students were asked to fill out a survey about the course. As an incentive, they were offered an extra point toward the exam score. All of the survey questions relevant to our study were Likert items scaled from 1–7, where a low number indicated agreement and vice versa. A zero option was also included if the students felt the item was not applicable to them. The items for this study included statements about whether the students saw the progress visualization, whether they liked the visualization they saw and whether they felt motivated by it. The distributions of the responses to the questions are shown in Figs. 3, 4, 5 and 6.

3.3 Data

Our study uses data from two subsequent programming courses that are typically taken together. In our dataset each student in a course makes up a single data point. Therefore students who took both courses contributed two points of data. Our survey from the course exams provided data from 89 and 83 students for each course respectively. For the study, we only included students who agreed that they saw the visualization, i.e. whose responses were between 1 and 3 on the 7-point Likert scale. This narrowed our dataset down to a total of 118 data points. We had to exclude 27 data points due to missing information on the participants' ages, leaving us with a final dataset of 91 data points.

We divided the data into two groups: adolescents and adults. We consider adults to be students born before the year 1995, i.e. students who turned 22 the year of the study. This cutoff was chosen since it formed quite even groups and kept the age of the adolescents reasonably low. A later cutoff year would have raised the adolescent group's mean age to over 20. With our chosen cutoff the adolescents were 18–20 year olds, while the adults' age ranged from 21 to 34 years. The medians, means and standard deviations of the ages for the different groups are reported in Table 1.

There were 34 adolescents and 57 adults. Of all the individual students, 17 only took the first course, 20 only took the second course and 27 took both. For the students who took both courses, the visualization was switched between courses, which means that 27 students reported their experiences with both visualizations. The number of students in different age groups for the two visualizations can be seen in Table 1.

Table 1. The group sizes for the different age and visualization groups, along with the medians, means and standard deviations of the groups' ages.

Age group	Weekly vis.	Growth vis.	Median age	Mean age	σ of age
$\geq$21	27	30	23	24.11	3.64
<21	11	23	20	19.56	0.56
Total	38	53	21	22.41	3.64

3.4 Research Questions

We have the following two research questions for this work:

- RQ1. Does the type of visualization affect student scores?
- RQ2. Do participants of different ages respond differently to the different progress visualizations?

With the first research question, we are interested in studying whether the two visualizations affect students' scores differently, that is, whether the type of visualization matters. With the second research question, we examine whether a student's age affects how the student is affected by the visualization.

4 Results

To get a general overview of the results and distributions of the relevant survey responses, we visualize them in Sect. 4.1. We then analyze the results in greater detail, and seek to answer the research questions in Sects. 4.2, 4.3 and 4.4.

To answer research question 1, we normalized the total assignment scores for each student and calculated the means and standard deviations for each age group by visualization, reported in Table 4.

For research question 2 we calculated means and standard deviations of the responses to the survey questions "Did you like the visualization?" and "Did the visualization motivate you to complete more assignments?", which are reported in Tables 2 and 3 respectively.

4.1 Questionnaire Responses

The distributions of the questionnaire responses for each age and visualization group are reported above in Figs. 3, 4, 5 and 6.

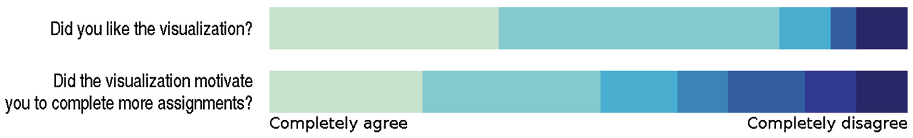


Fig. 3. The questionnaire response distribution from the adults who saw the weekly visualization.

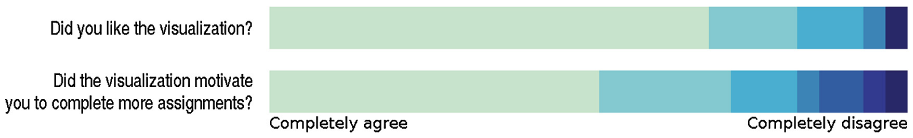


Fig. 4. The questionnaire response distribution from the adults who saw the growth visualization.

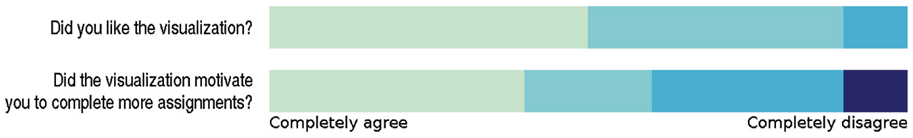


Fig. 5. The questionnaire response distribution from the adolescents who saw the weekly visualization.

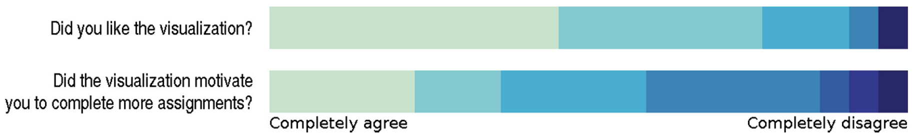


Fig. 6. The questionnaire response distribution from the adolescents who saw the growth visualization.

4.2 Preference

The different age groups seemed to prefer different visualizations. When asked if they liked the visualization, the mean response level of adolescents who saw the Weekly visualization was 1.55, whereas the adolescents' opinion on the Growth visualization was only 2.00. This indicates that the adolescents collectively preferred the Weekly visualization to the Growth visualization. Conversely, the mean response levels of adults were 2.22 and 1.65 for the Weekly and Growth visualizations respectively, i.e. the adults liked the Growth visualization more than the Weekly visualization.

There is very little difference between the standard deviations for the Growth visualization. For the Weekly visualization the standard deviations are significantly different between the two age groups: 0.69 for the adolescents and 1.72 for the adults. These values are reported in Table 2.

Table 2. Likert scale responses to the question "Did you like the visualization?". The responses were between 1 (completely agree) and 7 (completely disagree).

Age	Weekly vis.		Growth vis.	
	Mean	σ	Mean	σ
$\geq$21	2.22	1.72	1.65	1.28
<21	1.55	0.69	2.00	1.38

4.3 Motivating

The age groups were also more motivated by their preferred visualization. Adults reported being more motivated by the Growth visualization. When asked to report if the visualization affected their motivation to complete assignments, the mean response levels were 3.04 for the Weekly visualization and 2.19 for the Growth visualization. Similar to the preference question, the effect was reversed for adolescents. Their mean response levels were 2.27 for the Weekly visualization and 2.96 for the Growth visualization.

The differences in standard deviations follow the same trend that appears in the preference subsection, only to a lesser extent. The Growth visualization's

Table 3. Likert scale responses to the question "Did the visualization motivate you to complete more exercises?". The options were between 1 (completely agree) and 7 (completely disagree).

Age	Weekly vis.		Growth vis.	
	Mean	σ	Mean	σ
$\geq$21	3.04	2.01	2.19	1.64
<21	2.27	1.79	2.96	1.75

standard deviations are not significantly different, whereas the Weekly visualization's standard deviations are 2.01 for the adults and 1.79 for the adolescents, i.e. there is a slight difference. Although the standard deviations of the Weekly visualization group differ for both of the questions, the difference is significantly higher for the preference question. These values are reported in Table 3.

4.4 Points

The normalized assignment scores for the two different visualizations and age groups are reported in Table 4. We see a similar effect, albeit weaker, as with motivation and liking: younger students complete slightly more assignments when shown the Weekly visualization (86% completed vs 80% completed) and older students complete marginally more assignments when shown the Growth visualization (84% completed vs 81% completed).

Table 4. Average scores for the different age and treatment groups normalized to be between 0 and 1.

Age	Weekly vis.		Growth vis.	
	Mean	σ	Mean	σ
≥ 21	0.80	0.13	0.84	0.10
< 21	0.86	0.08	0.81	0.09

5 Discussion

Our results indicate that the long-term focused visualization received more positive responses from adult students and was associated with higher scores compared to the alternative. Similarly, the visualization which emphasizes weekly progress received more positive responses and resulted in higher scores in the younger group, compared to the alternative.

Therefore, the answer to our first research question, "*Does the type of visualization affect student scores?*", is affirmative. Moreover, as younger and older students preferred different visualizations, a more specific answer that also answers our second research question, "*Do participants of different ages respond differently to the different progress visualizations?*", is the following: "*Depending on the age of the student, different progress visualizations affect students' scores differently*". However, the differences between the visualizations were subtle. This is especially the case when comparing our results to Leppänen et al. [8], who found a much more intense effect. A number of relevant differences may explain this. Their study used simple multiple choice questions, and kept the progress bar always visible – the progress bar also indicated when the student had "completed" a specific task, even though in reality their completion was arbitrary. In our study, the progress was slower as the assignments were more complex, and the visualization was only visible in the web-based course material and not in programming environment where students worked on most of the course tasks.

It is possible that a greater effect could be achieved if the visualization was shown after every completed assignment, also within the programming environment in which the students completed their course work.

We found that older students like the Growth visualization that emphasizes long-term gains more than the Weekly visualization that highlights short-term progress. These results are in line with those reported by Steinberg et al. [15] who found that students' future orientation continues to develop until their mid-twenties. This suggests that there might be a benefit to adjusting motivational tools according to the students' ages, even among legal adults.

Generally, students both liked and were motivated by the two visualizations. For all groups, the mean response level was under 4 on a scale from 1 (completely agree) to 7 (completely disagree). Thus, our results indicate that students like both visualizations, and the differences in our results are only in the magnitude of the liking. This is in line with previous work by Bull et al. [2], who found that students enjoy simple visualizations, as both of the visualizations in this study can be considered simple. On the other hand, it is possible that students who did not like the visualization turned the visualization off, and therefore answered that they did not see the visualization – such students were excluded from this study.

Another reason for the younger students thinking more favourably of the Weekly visualization might be that it is more familiar to them. Progress bars are commonly used for depicting various elements such as skill proficiency and enemy health in video games, and presumably younger students spend more time on them on average compared to older students.

In a broader context, our results indicate that in order to increase students' learning, it might be more effective to personalize visualizations depending on students' demographic factors such as their age. Even subtle improvements in students' success could be beneficial in e.g. programming courses that suffer from notoriously poor pass-rates [1].

5.1 Limitations

In this subsection, we acknowledge the limitations of our work. As our study is based on a small population with rather small differences in the outcomes, none of the results are statistically significant, despite the visual analysis that indicates otherwise. Therefore, this study should be treated more as qualitative first insight into this topic rather than a quantitative examination of the effect of different visualizations on learning. Comparisons between adult and adolescent learners are always difficult as the age is rarely the only difference; older students are often at very different points in their lives. Due to our method of splitting the students into only two age groups of under 21-year-olds and at least 21-year-olds, the younger group is much more homogeneous than the older one. For example, there is likely a larger difference between a 22-year-old student and a 32-year-old student compared to a 22-year-old student and a 20-year-old student, but in our study the 22-year-old would be in the same group with the 32-year-old student.

Our study also lacked a control group of students who saw no visualization at all. As such, we can not say how much the existence of a visualization affected students performance over a baseline – we can only compare the effect between the two visualizations. However, because most of the students we studied said they liked and were motivated by the visualization they saw, we can assume that the visualizations did not have a negative impact on the students' scores. That is, we may assume that a control group would have performed more poorly.

6 Conclusions

In this article we studied how different age groups responded to being shown different visualizations of their course progress – overall, students liked the visualizations and they can be a useful tool for educators in keeping students motivated and engaged. However, our preliminary results suggest not all such visualizations are made equal. Out of our two progress visualizations, the one that emphasized long term growth was better liked and promoted better scores among the older age group, whereas the younger age group performed better with a bar plot visualization that emphasized each week's progress individually. The differences were rather subtle, however, and were not statistically significant.

In the future, we would like to further test our findings on a course with more students. It would also be valuable to have a control group who were not shown any visualization to study how these visualizations compare to not having a visualization at all. In addition, we want to study how increases in the number of completed exercises translates to actual acquired skills, for example whether there are differences in exam scores between the different visualization groups.

References

1. Bennedsen, J., Caspersen, M.E.: Failure rates in introductory programming. ACM SIGCSE Bull. **39**(2), 32–36 (2007)
2. Bull, S., Ginon, B., Boscolo, C., Johnson, M.: Introduction of learning visualisations and metacognitive support in a persuadable open learner model. In: Proceedings of the Sixth International Conference on Learning Analytics & Knowledge, pp. 30–39. ACM (2016)
3. Caprara, G.V., Fida, R., Vecchione, M., Del Bove, G., Vecchio, G.M., Barbaranelli, C., Bandura, A.: Longitudinal analysis of the role of perceived self-efficacy for self-regulated learning in academic continuance and achievement. J. Educ. Psychol. **100**(3), 525 (2008)
4. Dweck, C.S.: Mindset: The New Psychology of Success. Random House Incorporated, New York (2006)
5. Dyckhoff, A.L., Zielke, D., Bültmann, M., Chatti, M.A., Schroeder, U.: Design and implementation of a learning analytics toolkit for teachers. J. Educ. Technol. Soc. **15**(3), 58 (2012)
6. Holman, C., Aguilar, S., Fishman, B.: Gradecraft: what can we learn from a game-inspired learning management system? In: Proceedings of the Third International Conference on Learning Analytics and Knowledge, pp. 260–264. ACM (2013)

7. Jacovina, M.E., Snow, E.L., Allen, L.K., Roscoe, R.D., Weston, J.L., Dai, J., McNamara, D.S.: How to visualize success: presenting complex data in a writing strategy tutor. In: EDM, pp. 594–595 (2015)
8. Leppänen, L., Vapaakallio, L., Vihavainen, A.: Illusion of progress is moar addictive than cat pictures. In: Proceedings of the Third ACM Conference on Learning@Scale (L@S 2016), pp. 133–136, New York, NY, USA. ACM (2016)
9. Loboda, T.D., Guerra, J., Hosseini, R., Brusilovsky, P.: Mastery grids: an open-source social educational progress visualization. In: Proceedings of the 2014 Conference on Innovation & #38; Technology in Computer Science Education (ITiCSE 2014), p. 357, New York, NY, USA. ACM (2014)
10. Morrison, B.B., DiSalvo, B.: Khan academy gamifies computer science. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education, pp. 39–44. ACM (2014)
11. O'Donovan, S., Gain, J., Marais, P.: A case study in the gamification of a university-level games development course. In: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, pp. 242–251. ACM (2013)
12. O'Rourke, E., Haimovitz, K., Ballweber, C., Dweck, C., Popović, Z.: Brain points: a growth mindset incentive structure boosts persistence in an educational game. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 3339–3348. ACM (2014)
13. Park, Y., Jo, I.-H.: Development of the learning analytics dashboard to support students' learning performance. J. UCS **21**(1), 110–133 (2015)
14. Santos, J.L., Govaerts, S., Verbert, K., Duval, E.: Goal-oriented visualizations of activity tracking: a case study with engineering students. In: Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, pp. 143–152. ACM (2012)
15. Steinberg, L., Graham, S., O'Brien, L., Woolard, J., Cauffman, E., Banich, M.: Age differences in future orientation and delay discounting. Child Dev. **80**(1), 28–44 (2009)

3D Printing as Medium for Motivation and Creativity in Computer Science Lessons

Petra Kastl, Oliver Krisch, and Ralf Romeike(✉)

Computing Education Research Group,
Friedrich-Alexander-Universität Erlangen-Nürnberg,
Martensstraße 3, 91058 Erlangen, Germany
{petra.kastl,oliver.krisch,ralf.romeike}@fau.de

Abstract. 3D printers are a modern technology which is new to most students and can be used to motivate them. However, if the use of a 3D printer is limited to the downloading and printing of prefabricated models, the students will learn little about computer science. In this article, we present an approach for learning basic programming skills, which is based on the programming of three-dimensional objects using turtle graphics in Beetle Blocks. In addition, there should be an interdisciplinary link to the functions taught in mathematics teaching. The motivation was the creation of a 3D figure based on mathematical functions, its creative elaboration and its 3D printing at the end of the sequence. The experience shows that students are inspired by the idea, but the link with mathematical foundations is a challenge.

Keywords: 3D printing · Programming · Junior high · Beetle Blocks · Creativity

1 Introduction

For a long time, technological progress with respect to school classes was primarily driven by better graphics performance and was used for motivating students in the school, e.g. in the field of computer graphics. In recent years, 3D printers have been a new technological development, usually not (yet) found in the typical household. Nevertheless, it can fascinate students, their colleagues and parents alike with amazing results. Therefore, 3D printers are purchased at various schools to provide the impressive possibilities of 3D printing to students. In practice, however, the use of 3D printers is often limited to configuring the printer and printing out prefabricated models downloaded from the Internet, which is supported by [2]. As of yet, few convincing examples have been published that make use of the interdisciplinary potential of 3D printing in computer science lessons. In the following, we present a teaching example in which students are motivated by creating and printing algorithmically-generated 3D models as an introduction to programming. The students experiment with concepts they know from their mathematics lessons. Furthermore, the application

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 27–36, 2017.
https://doi.org/10.1007/978-3-319-71483-7_3

of these mathematical concepts in combination with simple algorithmic structures makes possible the creation of impressive figures and thus opens up the application context outside mathematics.

2 Geometric Shapes and Algorithmic 3D Models

2.1 Turtle Graphics as the Basis for Algorithmic 3D Models

In the 1980s, Seymour Papert introduced Logo, a programming environment that allows children to explore geometric shapes by drawing them on the screen using simple programming instructions. In his learning theory called constructionism, Papert [5] emphasizes the role of creating concrete artifacts that are personally meaningful and can be shown, tested and admired by others. The following examples are based on the corresponding turtle graphics approach, but extended into three-dimensional space: The output of the 3D printer, or its virtual representation, corresponds to the traced path of the turtle in Logo or to the pen of an object in Scratch. Many everyday objects can be constructed from mathematical objects and shapes and are suitable as a basis for turtle graphics. The starting point in our teaching example was a saltshaker. Its shape is equivalent to a hyperboloid with the formula $\frac{x^2}{5.5^2} - \frac{y^2}{2.5^2} = 1$ (see Fig. 1). The idea was to build similar three-dimensional objects, which are based on mathematic functions, and print them with a 3D printer. To implement the code, the block-oriented programming language Beetle Blocks [3] was used.

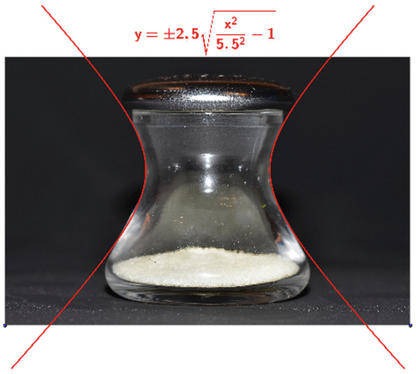


Fig. 1. Saltshaker - hyperboloid as a template for 3D objects.

The saltshaker is based on the underlying shape of a circle, which serves as the starting shape in our example. The pupils generated the circle line with the help of the formula $r^2 = x^2+y^2$. The formula was transformed to $y = \pm\sqrt{r^2 - x^2}$, which divided the circle by drawing two semicircles. The pupils could now draw a cylinder by slowly increasing the height z. In a second step, they then learned to change the radius of the individual circles in relation to the height. This was firstly achieved exemplarily by simple mathematical operations, which were then replaced by more complex operations during the course of the lesson. The advantage of building the object from individual layers is that the 3D printer also builds its objects by adding individual layers. Thus, the pupils were able to see in advance how the whole object emerges from individual layers (cp. Fig. 2).

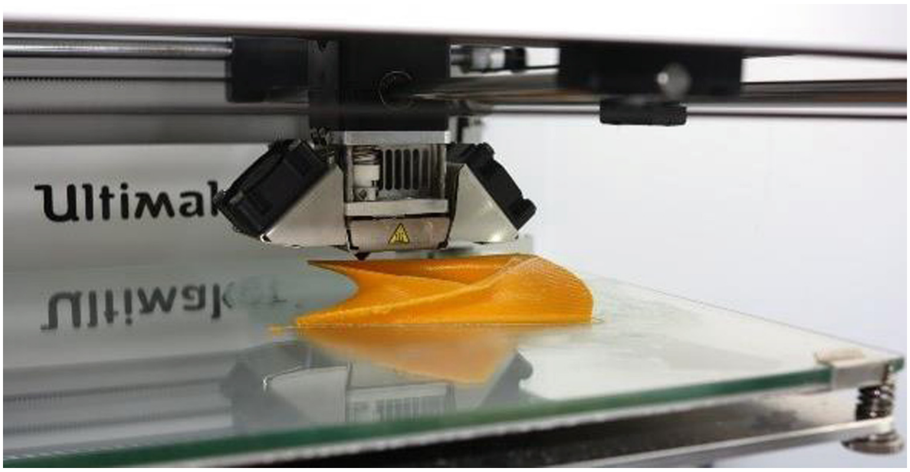
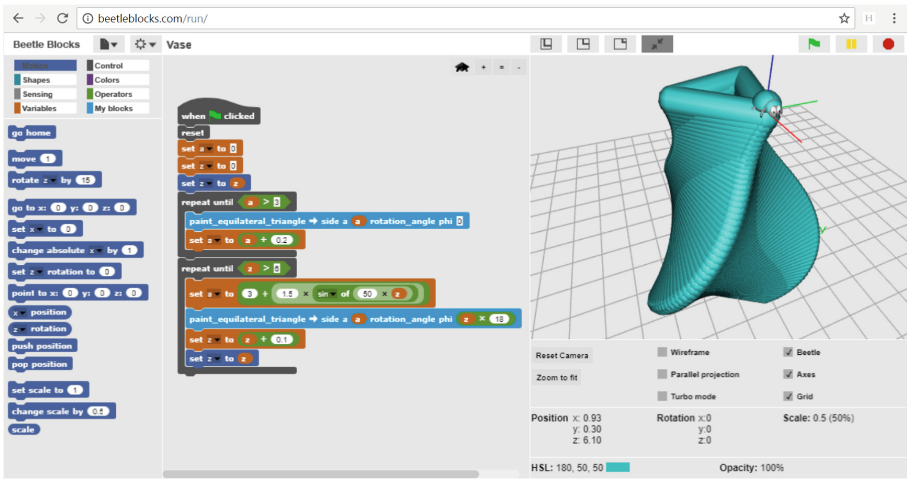


Fig. 2. Printing of a Beetle Block 3D model with an ultimaker 2.

2.2 Visual Programming with Beetle Blocks

With the introduction of Scratch [4], the entry-level for beginners has been significantly reduced, which is particularly beneficial for pupils at junior high. By visualizing the programmed objects (sprites) on the stage, "programming", in the sense of Paperts' constructionism, is immediately experienced, the multimedial bandwidth allowing the pupils to implement personally significant ideas and to directly follow the consequences of their programming decisions. The representation of the instructions as blocks also avoids syntax errors and the available programming constructs are immediately available. Just as Strecker [6] reported on the use of graphical programming languages in high school, due to its intuitive and easy handling, the visual access is also appealing in junior high. Koschitz and Rosenbaum [3] now transfer the concepts and user interface of Scratch, as well as the idea of turtle graphics, into three-dimensional space using Snap [1]. With Beetle Blocks, three-dimensional objects can be programmed based on algorithmic structures (see Fig. 3). The results can subsequently be exported as a 3D model and can be printed out using a 3D printer. Algorithmic, mathematical and artistic objects are thus becoming directly tangible. In the classroom, we have had good experiences with designing jewelry, everyday objects such as vases and bowls, as well as artistic figures. The potential of a lesson aiming towards these goals is that even those students who are not primarily interested in programming can be motivated by the inventive creation of products typically not expected to be encountered in computer science.



```
when 🏁 clicked
reset
set a to 0
set z to 0
set z to z
repeat until a > 3
    paint_equilateral_triangle ➜ side a a rotation_angle phi 0
    set a to a + 0.2
repeat until z > 6
    set a to 3 + 1.5 × sin of 50 × z
    paint_equilateral_triangle ➜ side a a rotation_angle phi z × 18
    set z to z + 0.1
    set z to z
```

☐ Wireframe
☐ Parallel projection
☐ Turbo mode
✓ Beetle
✓ Axes
✓ Grid

Position x: 0.93 y: 0.30 z: 6.10
Rotation x:0 y:0 z:0
Scale: 0.5 (50%)
HSL: 180, 50, 50
Opacity: 100%

Fig. 3. Beetle Blocks.

3 Implementation

3.1 Context and Background

In tenth grade at Bavarian junior high, at the latest, students will learn about the basic structures of sequence, conditionals, and loops in the "Modeling and Coding Algorithms" module of the curriculum for the subject of "information technology", which includes computer science. Another point of the curriculum is the implementation of algorithms with a suitable programming tool. The motivation of the students is a major concern in these lessons. Correspondingly, the programming languages used for learning have changed over the course of time. For example, the Pascal programming language was previously used at our school. In order to increase motivation, we moved to Delphi. With Delphi, the students programmed a pocket calculator or a vending machine at the end of the teaching sequence, applying the newly acquired skills in a more complex context. Such a self-programmed pocket calculator motivated some, but did not inspire enthusiasm in the student group. The reasons for this were twofold: on the one hand the very static interaction with the environment, and on the other hand the high error rate due to syntactic errors. This has changed with the introduction of the visual programming language Scratch. With this programming language, the students were able to achieve an appealing result very quickly and without syntactic errors. The first small games were programmed very quickly and could be taken home. Working with Beetle Blocks was similar to Scratch. With a few commands, the students could quickly move the beetle over the screen. This caused amusement and curiosity in the group.

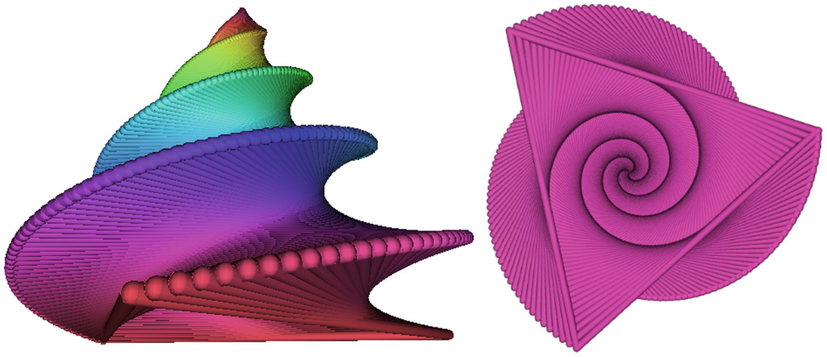


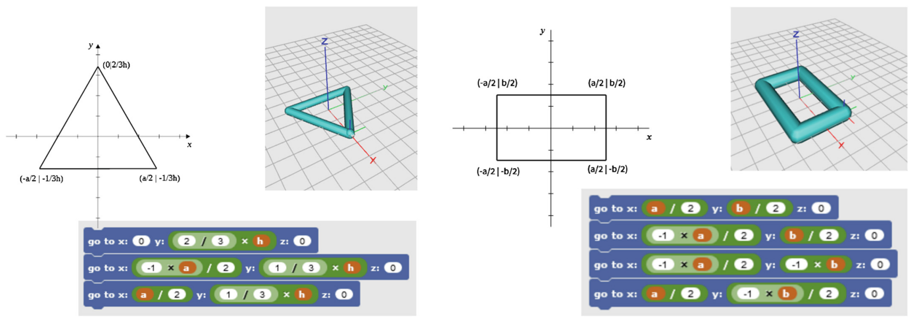
Fig. 4. Snail of equilateral triangles

3.2 Teaching Objectives and Lesson Structure

The aim was for students to use simple basic mathematical forms, e.g. circle, square or triangle, to develop three-dimensional objects and to program them. As a template, the students were able to use familiar subjects from their daily lives, while being able to freely choose their 3D objects, since they had different ideas ofwhich objects to create. One of the goals was to give the students the opportunity to use their own creativity independently and implement their own ideas. For this purpose, the students had to draw on their knowledge of geometry. In addition to the well-known theorem of Pythagoras, the basic forms also included other basic knowledge, for example, the division ratio of the heights in an equilateral triangle. With this knowledge, the basic shapes were created very quickly (cp. Fig. 5). For the third dimension, mathematical functions were used: With the help of sine, cosine and other functions, the size of the basic shapes was changed as a function of the height to give the objects a nicely curved shape. In Fig. 4 we see a colored snail, which is composed of equilateral triangles. In the snail, the side length was gradually reduced in size.

The lesson is divided into three phases:

1. Learning the algorithmic basic structures of sequence and loops and application/exercise with the programming language Beetle Blocks. In this case, the pupils produce simple geometric figures, such as e.g. rectangles or squares (3 double lessons)
2. Expansion into the third dimension, by the students building the simple geometric figures as a tower (1 lesson)
3. Mathematical modification of the tower (3 double lessons, cp. Fig. 7))



```
go to x: 0 y: 2 / 3 × h z: 0
go to x: -1 × a / 2 y: 1 / 3 × h z: 0
go to x: a / 2 y: 1 / 3 × h z: 0
```

```
go to x: a / 2 y: b / 2 z: 0
go to x: -1 × a / 2 y: b / 2 z: 0
go to x: -1 × a / 2 y: -1 × b z: 0
go to x: a / 2 y: -1 × b / 2 z: 0
```

Fig. 5. Programming of basic geometric shapes.

3.3 Course of Instruction and Observations

The example in focus was taught in a 10th grade class with pupils from the economic branch of a junior high school. Therefore, the initial enthusiasm of the students to learn programming was lacking. Linking the content with the "beloved" subject of mathematics did not contribute to the motivation. This was shown, for example, in the statement of student Clara: "I don't understand math, and now I have to learn how to program it." Correspondingly, the start of the lesson was difficult. The students were to learn to use the term "sequence". For this purpose, we programmed simple geometric figures, for example, a rectangle. The programming of an equilateral triangle was somewhat more elaborate, since the coordinates of the third corner had to be calculated with the help of the Pythagorean theorem. The fourth lesson was somewhat easier for the pupils because only the variable z, which had not yet been used, was added. Up until this point, they still had not used repetition. During the construction of the tower, the pupils programmed the height of their tower by repeatedly increasing the variable z with a counting loop.

Starting from the fifth lesson, the motivation of the pupils increased significantly. The teacher brought a 3D-printed object and promised the students their objects would also be printed with the 3D printer. After the constructive introduction, in the sense of a bottom-up approach, further teaching now pursued an experimental-deconstructing approach. For this purpose, different methods were provided to the pupils in the form of given blocks, which, in addition to the

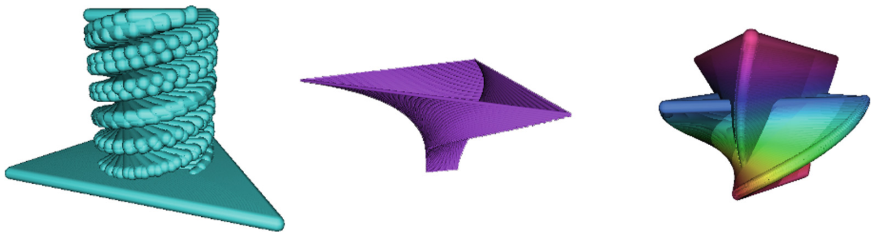
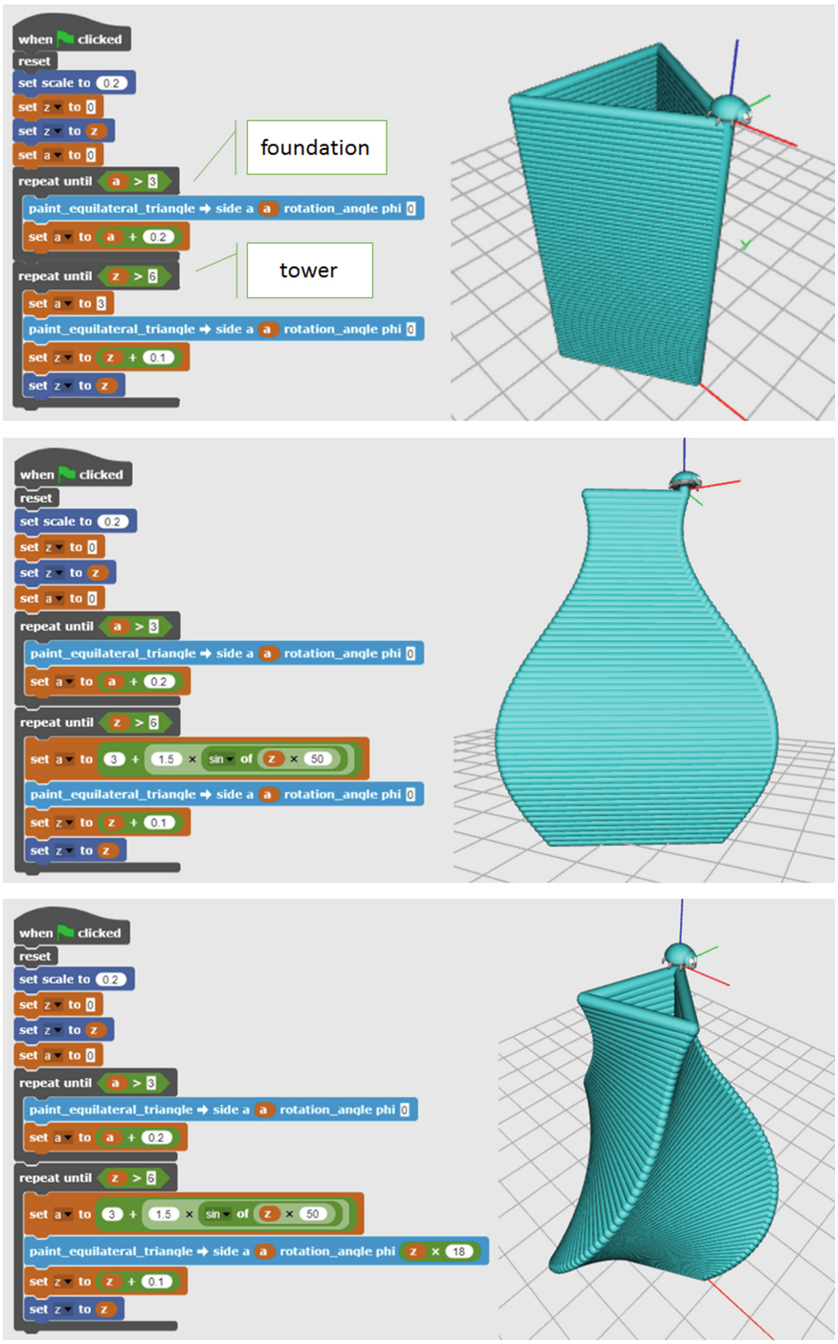


Fig. 6. Student results: Variations of the towers.



```
when clicked
reset
set scale to 0.2
set z to 0
set z to z
set a to 0
repeat until a > 3
    paint_equilateral_triangle ➜ side a a rotation_angle phi 0
    set a to a + 0.2
repeat until z > 6
    set a to 3
    paint_equilateral_triangle ➜ side a a rotation_angle phi 0
    set z to z + 0.1
    set z to z
```

```
when clicked
reset
set scale to 0.2
set z to 0
set z to z
set a to 0
repeat until a > 3
    paint_equilateral_triangle ➜ side a a rotation_angle phi 0
    set a to a + 0.2
repeat until z > 6
    set a to 3 + 1.5 × sin of z × 50
    paint_equilateral_triangle ➜ side a a rotation_angle phi 0
    set z to z + 0.1
    set z to z
```

```
when clicked
reset
set scale to 0.2
set z to 0
set z to z
set a to 0
repeat until a > 3
    paint_equilateral_triangle ➜ side a a rotation_angle phi 0
    set a to a + 0.2
repeat until z > 6
    set a to 3 + 1.5 × sin of z × 50
    paint_equilateral_triangle ➜ side a a rotation_angle phi z × 18
    set z to z + 0.1
    set z to z
```

Fig. 7. Construction and variation of a triangle-based vase.

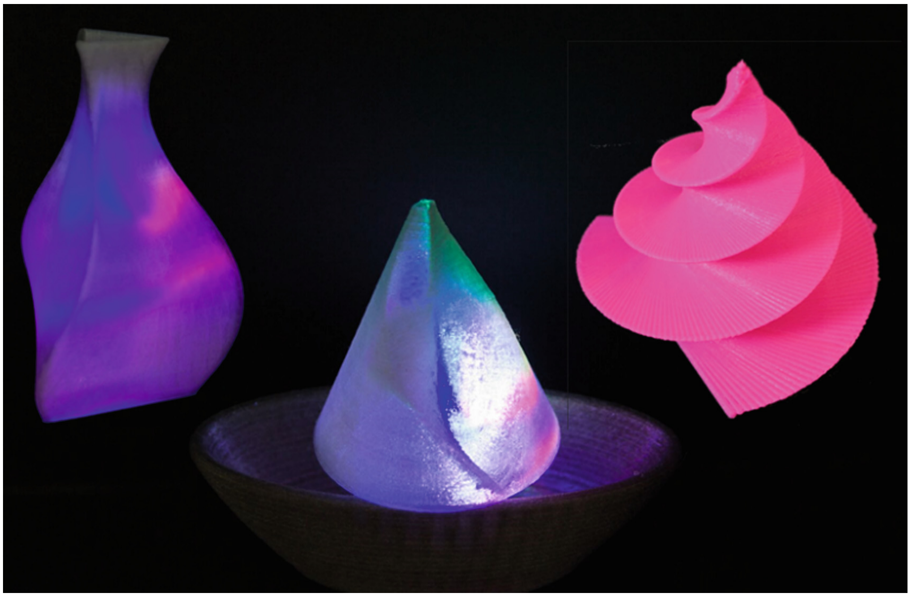


Fig. 8. Vase, art object and snail from the 3D printer.

blocks developed by the pupils, provided the possibility for the geometric figures to still be rotated by an angle α.

The reason for providing the blocks was that the students had not yet learned the underlying mathematical concepts in their mathematics lessons. In addition, the time given was seven-hours, which can be considered short, and the students should be given the opportunity to spend more time developing their own 3D objects. In doing so, they elaborated the previously learned method of building the individual objects from layers. In the course of time, the attitude of Clara also changed. With statements like "Is there another mathematical function that I can try?" Clara wanted to try ever more variations of her tower. Thus, not only were simply turned towers produced, but also, for example, small bowls. Other students built nested objects (see Fig. 6).

After all, one of the most beautiful models was created by Clara. A snail-shell, which consisted of turned, equilateral triangles with a decreasing side-length towards the top (see Fig. 8). All results were then printed with a 3D-printer and could be taken home by the students.

Summarizing, the students achieved the following milestones:

1. Creation of two-dimensional geometric shapes using sequences and variables as the basis.
2. Creation of uniform three-dimensional shapes as an aggregate of two-dimensional shapes (slices) using loops and procedures.
3. Experimenting with varying parameters by applying trigonometric functions to the side length of the basic geometric shapes, using a given block.
4. Experimenting with rotation of the basic geometric shapes.
5. Inclusion of additional parts, such as a foundation.

4 Conclusion and Outlook

As a conclusion, the students' motivation increased when it became clear that the 3D objects they were programming were to be printed three-dimensionally. Their involvement in learning programming was comparable to that of the students learning to program with Scratch. Also, regarding the speed of learning, there was no difference to the Scratch group. With the first basic geometric figures, things became more difficult. The students were able to calculate the coordinates of the corners of a square without any problems, but with an equilateral triangle, some problems arose. Here, a different approach should be found and tested with simpler mathematical methods. The use of mathematics was significantly different from the way students are used to solving problems in their mathematics lessons. Perhaps, students from the mathematical branch, or grammar school, have fewer difficulties. A group of two students did something different: Instead of calculating the coordinates, they have constructed an equilateral triangle on a piece of paper, placed it in the origin of the coordinate system, and read the coordinates of the equilateral triangle from the scales. However, in the third dimension, the two students could not change the size of the sides. For the others, the transition to the third dimension was easier: Since the basic figures were programmed in such a way that their size depended on the length of the side, this could be easily changed using mathematical functions. The pupils' creativity was given free rein. They were able to apply the functions they had learned so far. In this step, a few students varied the thickness of the lines drawn by the beetle. Another group of students also changed the color of the drawn lines with the height. Two students provided their 3D object with a floor plate to get a vase. This part gave the pupils lots of fun. A link to other modules from the curriculum, for example with Arduinos (module: micro-controller) as a control for an artistic lamp (see Fig. 8), would be a suitable next step.

For the repetition of the teaching sequence, it is recommended to try creating the basic figures with simpler mathematical methods. The linking of height with mathematical functions worked well. Here, the students have the opportunity to see that mathematics is not only an abstract, theoretical subject, but that it also has an application to give space to one's own creativity and to develop new objects. Another new starting point would be to subdivide real objects (see saltshakers) into mathematical basic objects and to reprogram them.

References

1. Harvey, B., Mönig, J.: Bringing "no ceiling" to scratch: can one language serve kids and computer scientists. In: Proceedings of Constructionism, Paris (2010)
2. Shewbridge, R., Hurst, A., Kane, S.K.: Everyday making: identifying future uses for 3D printing in the home. In: Proceedings of the 2014 Conference on Designing Interactive Systems, pp. 815–824. ACM (2014)
3. Koschitz, D., Rosenbaum, E.: Exploring algorithmic geometry with "Beetle Blocks": a graphical programming language for generating 3D forms. In: 15th International Conference on Geometry and Graphics Proceedings, Montreal (2012)

4. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The scratch programming language and environment. ACM Trans. Comput. Educ. (TOCE) **10**(4), 16 (2010)
5. Papert, S.: Mindstorms-Kinder, Computer und Neues Lernen. Birkhäuser Verlag, Basel (1982)
6. Strecker, K.: Graphical programming languages in high school (Grafische Programmiersprachen im Abitur). In: Gallenbacher, J.(Hrsg.): INFOS 2015: Informatik allgemeinbildend begreifen (16. GI-Fachtagung Informatik und Schule, Darmstadt). Bonn: Köllen (2015)

Teaching Materials in Informatics for Lower Secondary School Blind Students

Ľudmila Jašková(✉) and Mária Stankovičová

Comenius University, Bratislava, Slovakia
jaskova@fmph.uniba.sk, stankovicova@cezap.sk

Abstract. This paper deals with teaching materials in informatics designed for lower secondary school blind students. First, we describe our motivation to develop special teaching materials for blind students. The second chapter provides an overview of the available computing textbooks for primary and lower secondary education. In the third chapter we explain the way of using computers by the blind to justify the need for teaching materials specially designed for this category of students. Further, we offer a detailed description of the form and content of teaching materials developed by the CEB project (Computing Education for Blind). In conclusion, we describe our research aimed at the verification of suitability of the created materials.

Keywords: Informatics education · Blind · Teaching material · Programming · Accessibility

1 Introduction

In recent years, many countries have been directing their efforts towards introducing computer science to pupils at all levels of education in the appropriate volume and form. In exposing students to computing at school it is important to provide them with some basic information about the discipline, as is common with other scientific disciplines, so that later on they will be able to choose whether to pursue this discipline further in their future education [1]. The CSTA K-12 standards [2] describe **computer science** as **the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society**.

In Slovakia, informatics became a standard compulsory subject for almost all grades of primary and secondary schools in 2008 [3, 4]. The content of this subject includes: basic ICT literacy, information and data handling, communication via ICT, algorithms, problem solving, user applications, working with the Internet, text, tables and images, principles of hardware, social aspects of using ICT. Those themes are adapted to the age of the pupils, with separate curriculums aimed at primary schools and secondary schools. Since 2011, we have had an opportunity to teach informatics at a special school for visually impaired pupils, and in recent years our research has been aimed at the development of suitable teaching materials for teachers of blind pupils. Research is under way in the framework of the **CEB project** (Computing Education for Blind). This project is scheduled for 2016-2018 and funded by our national agency.

2 Textbooks of Computing

Since the beginning of this millennium, we have been observing efforts in the education systems of many countries that the content of informatics as a subject has increasingly evolved from the narrow concept of using a computer as a tool to a more complex concept of understanding its principles [5]. The subject has been gradually updated, enabling teachers to focus on the fundamentals and on sustainable knowledge [6]. This trend is also evident in the changing content of the books for children.

The authors initially focused on the proper use of computers [7–9] (Fig. 1).

Internet & Computer Ethics for Kids
COMPUTERS DON'T BYTE 2ND EDITION
A BEGINNER'S GUIDE TO UNDERSTANDING COMPUTERS
A Kid's Guide to CREATING WEB PAGES for Home and School

Fig. 1. Books about proper use of computers

Later on, there were more publications devoted to programming and computer science [10–12] (Fig. 2).

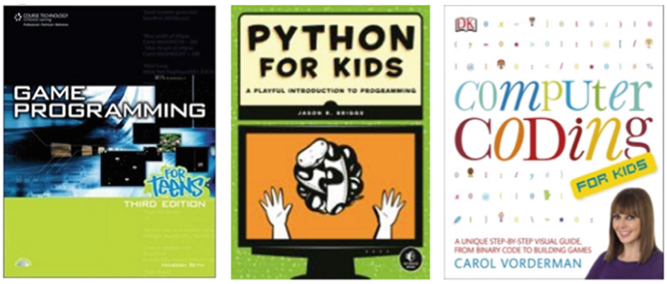

GAME PROGRAMMING for teens THIRD EDITION
PYTHON FOR KIDS
A PLAYFUL INTRODUCTION TO PROGRAMMING
COMPUTER CODING FOR KIDS
CAROL VORDERMAN

Fig. 2. Books about programming

A collection of textbooks titled **Creative Informatics** was created in Slovakia between 2005 and 2010. Lower secondary schools (age 11 to 15) were provided with exercise books on work with graphic and text editors, spreadsheets and the Internet [13], but also with a textbook on the principles of computer science [14] and workbook about programming in Comenius Logo environment [15]. Between 2010 and 2013 textbooks were created also for primary schools (age 6 to 10), e.g. [16]. While in our view, many of these textbooks present a good tool for the teachers of mainstream students, they are not suitable for the teachers of blind students.

3 Textbooks for Blind Students

Blind people use computers in a totally different way than sighted users. They do not use a mouse, and all the input is entered via a keyboard. Since they use a screen reader, which reads the content of the screen in a linear way, they cannot get an instant view of the screen content, as sighted users do.

A different way of using a computer requires a different way of teaching computer skills and computer science. Unfortunately, our country suffers from a shortage of qualified informatics teachers of the blind students. While regular IT teachers do not have the experience of teaching blind learners, teachers from special schools do not have the experience of teaching informatics. For both, high-quality teaching material is highly desirable. Since the IT knowledge and computer skills are essential for blind learners, so that they can continue at higher levels of study and succeed in the labor market, informatics is the key subject for them.

Since 1993, Sarah Morley Wilkins has written a series of books on Microsoft Windows, helping blind and visually impaired computer users to make effective use of their PC in the Windows environment (http://www.sarahmorleywilkins.com/). These books, together with accessible images (Fig. 3), inspired us to create our educational materials.

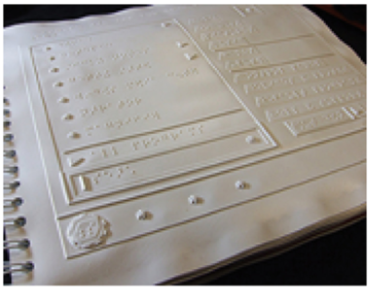


Fig. 3. A tactile image showing the Windows 7 Start Menu (http://www.sarahmorleywilkins.com/books/win7/)

Another source of inspiration was tutorials for the blind from companies that develop software for the blind (e.g. Freedom Scientific) or from organizations defending the rights of the blind (e.g. American Foundation for the Blind). All of these materials, however, were designed for adult users, who are highly motivated to learn how to work with a computer. They do not need a collection of interesting tasks, but a list of keyboard shortcuts and step-by-step instructions. We have not found textbooks for so young students (age 11 to 15) with suitable and at the same time engaging tasks to build and develop their computer skills. Therefore, our goal was to create teaching materials that offer a set of interesting tasks with graded difficulty to teach pupils in an enjoyable way.

Within the CEB project we focus on the following areas of informatics:

– **Text processing** – the ability to create a well-structured and clearly formatted text document should be considered part of basic digital literacy. Blind people should be able to imagine what is happening with the text when they use various formatting tools. They should understand the impact of formatting on the understanding of text content by the reader.

- **Sound processing** – blind people live mostly in the world of sounds. They have aural associations with many objects, feelings and situations. When reading text on the screen, they hear synthetic speech, which is not as natural as recorded sounds. Understanding synthetic speech is sometimes demanding and tiring. Activities that allow blind pupils to listen to natural sounds are a pleasant diversion for them. These activities stimulate their creativity and are motivating and attractive.
- **Programming and problem solving** – the ability to perform an activity following a sequence of instructions as well as the knowledge of how to divide a problem into partial problems are skills that are indispensable for study and a successful life in 21st century.
- **Working with mathematical formulas** – the ability to understand and use mathematical formulas is the key to understanding mathematical concepts and fundamentals of computer science. Most of the mathematical tasks in the lower grades of secondary education cannot be solved without the ability to use mathematical formulas. Braille mathematical notation is different from regular mathematical notation. If a math teacher does not manage Braille mathematical notation, a barrier may arise when he or she teaches blind pupils. This barrier may be overcome by means of Lambda software, which may be especially helpful if blind students intend to proceed with their studies.
- **ICT for communication and information retrieval** – the ability to search for information and its subsequent use is important for learning various subjects as well as for life in modern society. The same can be said about the ability to communicate effectively with classmates and teachers using digital technologies.

The creation of teaching materials is based on our experience gained during teaching informatics to blind and partially sighted pupils [17]. It is our ambition to allow students to learn in a constructionist way in the spirit of Papert [18]. Constructionism means giving students appropriate tasks, topics, problems that would facilitate the learning process. We provide graded tasks, which each student can solve at his or her own pace. Apart from the tasks themselves, the data files that students work with (modify, supplement, remove bugs and the like) present an integral addition to the material. We believe that creating tasks with respect to the cognitive abilities of blind students is of utmost importance [19] and that tactile perception may significantly aid in knowledge acquisition and generation of ideas. Hence, tactile images are also included in the materials (e.g. tactile pictures illustrating the appearance of different font formats, tables, windows and the like).

All created teaching materials will be freely available on the web portal, so IT teachers from both special and regular schools will be able to download and use them.

Teaching materials for thematic areas **Text processing** and **Sound processing** have been completed and are ready for use. We will gradually complete the other materials so that all of them will be finished by the end of 2018.

3.1 Text Processing

The aim of the teaching material is to familiarize students with basic typographic principles so that they will be able to use them when editing and formatting text.

The teaching material contains almost 60 tasks complemented with a few dozens of data files that pupils should use to solve these tasks. Some tasks are intended for the initial stage of acquiring skills and gaining experience. These tasks also include explanations of basic concepts, keyboard shortcuts and step-by-step procedures. They require MS Word 2007 or higher. In addition, the material also includes tasks designed to practise and automate skills. Such tasks are mainly included in the last chapter, which contains ideas for larger projects.

A series of 24 tactile images (Fig. 4) show various font, paragraph, and text formatting. These images were created for a printer that combines embossed graphics with inkjet printing.

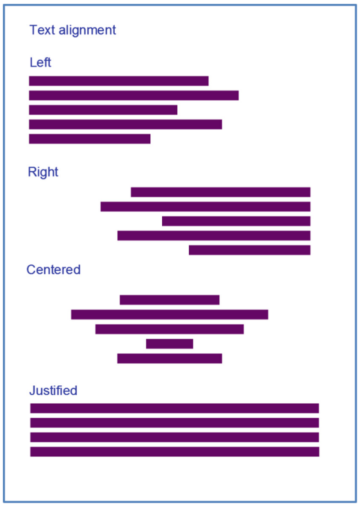


Fig. 4. A printed version of a relief image illustrating text alignment [20]

Now, let us take a sample task to explain the difference between educational texts designed for blind students and for sighted students. The aim of the task is to set the correct heading styles in the document. The screen reader offers keyboard shortcuts to move between headings in a document, but the heading styles must be set properly. If so, blind users can get an overview of the text more easily.

Table 1 lists the task assignment for the blind and the instructions for solving it. Note that the text contains detailed explanation of the task entry, while the instructions for solving it contain information on how to perform the required operations using the keyboard.

If the material were intended for sighted pupils, it would contain less text and would be completed with a picture of the finished text (Fig. 5).

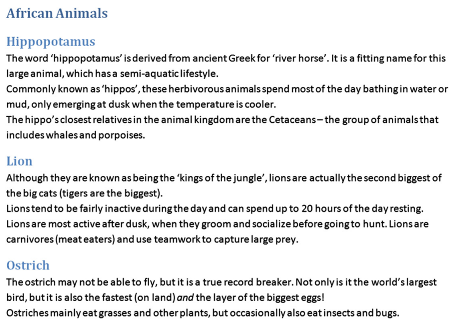
Table 1. Assignment of the task for the blind students

Open document *Headings2.docx*. The document mentions various African animals. The headlines do not have the correct headline style so we cannot move between them. The first line contains the text *African animals*. Set this line to the Heading 1 style. The names of the animals that appear on separate lines should be set to the Heading 2 style. Find them and set them to the correct headline style.

When you set the correct styles to all the headings, go to the top of the document and move over the headings.

Hints

- Set the headline style by moving to the line you want to set and press **Alt + Shift + Left Arrow** or **Alt + Shift + Right Arrow**.
- Set the normal style by moving to a line that you want to set and press **Ctrl + Shift + N**.
- We can find out the style of the current text by pressing **Insert+F**.



African Animals

Hippopotamus

The word 'hippopotamus' is derived from ancient Greek for 'river horse'. It is a fitting name for this large animal, which has a semi-aquatic lifestyle.
Commonly known as 'hippos', these herbivorous animals spend most of the day bathing in water or mud, only emerging at dusk when the temperature is cooler.
The hippo's closest relatives in the animal kingdom are the Cetaceans – the group of animals that includes whales and porpoises.

Lion

Although they are known as being the 'kings of the jungle', lions are actually the second biggest of the big cats (tigers are the biggest).
Lions tend to be fairly inactive during the day and can spend up to 20 hours of the day resting. Lions are most active after dusk, when they groom and socialize before going to hunt. Lions are carnivores (meat eaters) and use teamwork to capture large prey.

Ostrich

The ostrich may not be able to fly, but it is a true record breaker. Not only is it the world's largest bird, but it is also the fastest (on land) *and* the layer of the biggest eggs!
Ostriches mainly eat grasses and other plants, but occasionally also eat insects and bugs.

Fig. 5. Picture of the finished text

Instructions for solving the task would contain information on how to work with the text using the mouse and would be supplemented by a picture of Style ribbon (Fig. 6).

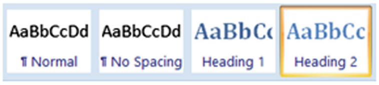


Fig. 6. Style ribbon in MS Word

3.2 Sound Processing

The aim of the teaching material is to help pupils acquire basic skills for their creative work with sounds. Specifically, this means acquiring skills to play audio files, to record and play sounds, to combine two or more sounds, to modify sounds and to learn the basic terminology needed to process audio information using a computer. The material contains almost 30 tasks. Instructions for solving these tasks are tied to the use of the AudaCity editor (http://www.audacityteam.org/). This editor is fully manageable via keyboard (http://manual.audacityteam.org/man/keyboard_shortcut_reference.html) and works well with the screen reader.

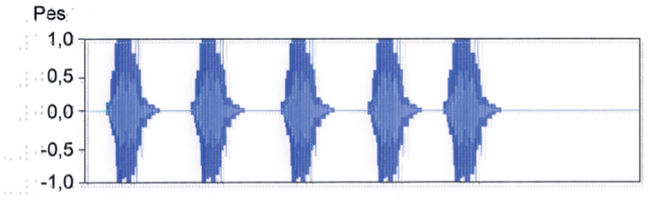


Fig. 7. Sound of barking dog represented as waves in AudaCity program (tactile image)

The material also includes tactile images illustrating particular environment elements (such as a volume slider), as well as tactile pictures illustrating the sound as waves (Fig. 7).

3.3 Programming and Problem Solving

Since we believe that algorithmic thinking is of great importance for the blind, we find that the programming profession is very suitable for them. Currently, a significant percentage of blind programmers are working in software companies. We realize that the sooner the pupils' interest in programming is appropriately captured, the more easily they will be able to decide about their further study needed to obtain the necessary qualification. The aim of the teaching material is to acquire basic programming skills and to understand concepts such as **command**, **sequence of commands**, **cycle**, **conditional command**, **procedure**, **variable**.

The teaching material will also include unplugged tasks that do not require a computer. Some of such tasks will require using robotic toys such as the Bee-Bot, Constructa Bot or a tactile table [21]. We also plan to include algorithmic tasks from the Beaver contest or their modifications [22, 23].

However, learning programming requires an appropriate programming environment. The existing languages intended for children are highly visual in nature, both in manipulating the code (e.g. drag and drop) and in the effect that the code has (e.g. animation, moving robot). As such, they are not appropriate to students with visual impairment. With this in mind, our tasks will be tied to the Alan audio programming environment developed by one of our students of informatics [24]. This environment allows users to program audio stories (Table 2), but also helps to develop creativity and to assimilate all the required programming concepts. The Alan environment will be freely available along with the teaching material.

Table 2. Program in Alan environment

```
Say(What animal makes this sound?)
Repeat 3 times
   Play(donkey)
End of Repeat
Question(Is that a donkey?) Answer: Yes
   Say(Excelent!)
Else
   Say(No, it is a donkey)
End of Question
```

Microsoft Research Cambridge is currently developing a physical programming language Torino (https://hxd.research.microsoft.com/work/torino.php), which is friendly to blind pupils (Fig. 8). The beta version should be available in autumn 2017. We plan to buy this language and create teaching material for this product as well.

According to the current timetable, this teaching material should be completed by the end of 2017. In line with our ambitions to support multiple platforms, this will be the most extensive teaching material.

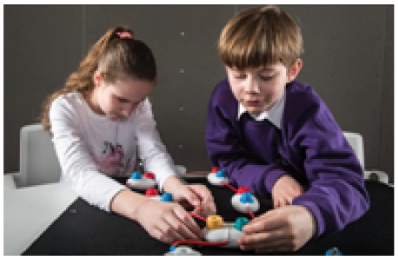
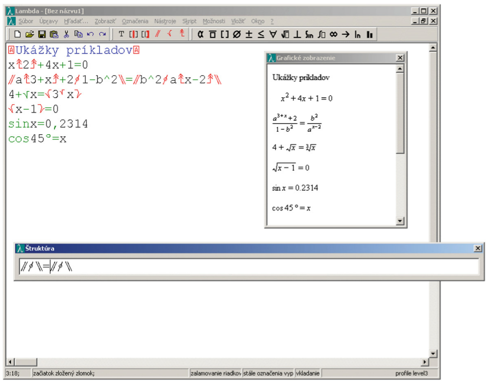


Fig. 8. A physical programming language Torino (https://hxd.research.microsoft.com/work/torino.php)

3.4 Working with Mathematical Formulas

This teaching material will provide practical instructions on the effective use of the Lambda editor (Linear Access to Mathematics for Braille Device and Audio-synthesis). This editor was developed as an outcome of the Lambda project (http://lambdaproject.org/) and was translated into Slovak language at our university in 2006–2007. As a follow-up, several courses for teachers and blind pupils were held at our Support Centre. The Lambda editor is a very useful tool because it helps pupils develop their understanding of mathematics and informatics [25]. Unfortunately, however, there is a lack of compact teaching material.

The first part of the teaching material under preparation will be aimed at understanding the difference between linear text view and graphical math view. In either of them, the editor responds to keyboard inputs in a different way.


Lambda - [Bez názvu1]
Súbor Úpravy Hľadať... Zobraziť Označenia Nástroje Skript Možnosti Vložiť Okno ?
Grafické zobrazenie
Štruktúra

```
Ukážky príkladov
x⇡2⇡+4x+1=0
//a⇡3+x⇡+2/1-b^2\\=//b^2/a⇡x-2⇡\\
4+√x=√3√x⟩
√x-1⟩=0
sinx=0,2314
cos45°=x
```

Ukážky príkladov

$x^2+4x+1=0$

$\frac{a^{3+x}+2}{1-b^2}=\frac{b^2}{a^{x-2}}$

$4+\sqrt{x}=\sqrt[3]{x}$

$\sqrt{x-1}=0$

$\sin x=0.2314$

$\cos 45°=x$

```
//∕\\=//∕\\
```

```
3:18; | začiatok zložený zlomok; | zalamovanie riadkov | stále označenia vyp | vkladanie | profile level3
```

Fig. 9. An example of a series of compensatory functions (linear text view, graphical math view, compact view)

The following sections will describe a series of compensatory functions (Fig. 9), whose aim will be to help students learn the basics of how to use these functions effectively (line duplication, compact structure of expression, calculator, calculation result insertion, multiple memory buffer, graphical visualization).

One of the major skills is the skill of keeping the right order of keyboard shortcuts. This skill requires students to understand the meaning of the notation. Keeping the right order of keyboard shortcuts is necessary for correct graphic visualization and for the right interpretation by the screen reader.

The last part will deal with exporting files to various formats, especially XHTML.

Based on our previous experience, the best time to start with the Lambda editor is in the lower secondary school. Later, when pupils acquire basic skills, they will be able to use the editor at math lessons as an appropriate exercise tool [26].

3.5 ICT for Communication and Information Retrieval

This teaching material will be finished at the end of 2018. The aim will be to teach pupils how to use tools for communication over the Internet safely and ethically. In addition to this, they should learn how to look up and use the information needed for their personal life, study and work in accordance with the copyright law.

A large number of websites are hard to use for blind learners because they do not meet accessibility standards (https://www.w3.org/WAI/intro/wcag). We plan to use a training site that satisfies the conditions of accessibility, which will be developed for this particular purpose. Search-based tasks will focus on verified locations such as Google search engine, YouTube sound sources, and Wikipedia sources of information.

4 Verification

As we have already mentioned, we are building on our long-term research observations, which we made at the elementary school for visually impaired pupils. Most of the tasks used in our materials have already gone through several iterations consisting of the design, implementation, verification and modification phases. Similar iterations will be done with each teaching material as a whole. We will verify them in cooperation with the teachers of blind pupils in the form of qualitative evaluation study [27]. Two independent reviewers will produce testimonials for each material, and we will incorporate their comments. In addition, we will analyze a detailed description of computing lessons, interviews with students, interviews with teachers. Next, we will analyze the tasks solved by students (predominantly given as projects or tests).

The material on text processing was verified at the beginning of this year during 10 informatics lessons. It can be said that both students and teachers have found the material easy to use. On the other hand, we have received a number of interesting responses from students. Based on this feedback, we will have to make minor adjustments to the text, to tactile images and to data files.

– Some texts in data files are too large. In order to practice the required operation, **shorter texts would be enough**.
– It is necessary **to modify the content of some data files** as they contain terms that blind pupils do not understand or cannot imagine (such as the names of flowers they have not encountered yet).
– Some task instructions refer to keyboard shortcuts containing numbers on the **alphanumeric keypad**. If pupils press a number on the **numeric keypad**, this does not have the desired effect. It is therefore necessary to specify exactly which keys should be pressed.

We have found that tactile images must form an integral part of the teaching material. Blind students could not imagine some objects without them, such as a bulleted list or more intricate nested lists.

Thanks to tactile images, we have noticed some misconceptions. For example, some students interpreted the double line setting as doubling of the letter size. Similarly, the centered alignment (Fig. 10) was interpreted as lines starting in the middle of

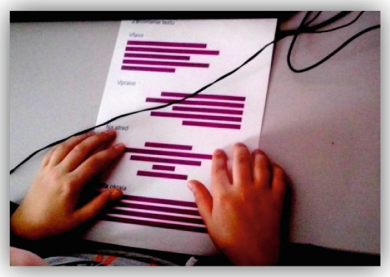


Fig. 10. A blind student encounters a tactile image illustrating text alignment [20]

the page. After using the tactile image, they realized that the distance between the beginning of the line and the left edge of the page is the same as the distance between the end of the line and the right edge of the page.

It was a valuable experience for us as well, because we entered the perception of blind pupils.

5 Conclusions

As all materials will be available on the web portal, we also expect feedback from IT teachers from different schools. Once all the materials have been created, we plan to organize a workshop for teachers of blind pupils and experts in the field of education of the blind both from our country and from the neighboring countries. This seminar will be yet another platform for exchanging experience and getting feedback.

Acknowledgements. This paper was created as part of a project Computing Education for Blind (Kega 014UK-4/2016) funded by KEGA grant. Special thanks to all participating students and teachers for their help and input into the research.

References

1. Armoni, M., et al.: Early education – what does computing have to do with it, and in what ways? In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education, London, UK. ACM DL, New York (2015). ISBN 978-1-4503-3753-3
2. Seehorn, D., et al.: CSTA K–12 Computer Science Standards. Computer Science Teachers Association, New York (2011)
3. Kabátová, M., et al.: Robotic activities for visually impaired secondary school children. In: 3rd International Workshop, "Teaching Robotics, Teaching with Robotics", Integrating Robotics in School Curriculum, Riva del Garda (TN), Italy (2012)
4. The national curriculum in Slovakia (in Slovak). http://www.statpedu.sk/clanky/statny-vzdelavaci-program
5. Jones, S.P., et al.: Computing at school in the UK (2013)
6. Hromkovič, J., Steffen, B.: Why teaching informatics in schools is as important as teaching mathematics and natural sciences. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 21–30. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_3
7. Schwartau, W., et al.: Internet & Computer Ethics for Kids: (and Parents & Teachers Who Haven't Got a Clue.). Interpact Press, Seminole (2001)
8. Gimmotty, S.: Computers Don't Byte: A Beginner's Guide to Understanding Computers, 2nd edn. (2004)
9. Selfridge, B., et al.: A Kid's Guide to Creating Web Pages for Home and School. Zephyr Press, Chicago (2004)
10. Sethi, M.: Game Programming for Teens. Course Technology, Boston (2008)
11. Briggs, J.R.: Python for kids, 3rd edn. No Starch Press Inc, San Francisco (2013). ISBN 978-1-59327-407-8
12. Vorderman, C.: Computer Coding for Kids. Dorling Kindersley Ltd., UK (2014). ISBN 9780241010433

13. Varga, M., Hrušecká, A.: Creative informatics. First exercise book on the Internet 1. SPN, Bratislava (2006). (in Slovak), ISBN 80-10-00648-3
14. Kalaš, I., Winczer, M.: Creative informatics. Informatics around us. SPN, Bratislava (2007). (in Slovak). ISBN 978-80-10-00887-2
15. Blaho, A., Kalaš, I.: Creative informatics. First exercise book in programming. SPN, Bratislava (2005). (in Slovak), 80-10-00019-1
16. Blaho, A., et al.: Informatics for the second grade of primary school. Aitec, Bratislava (2010). (in Slovak)
17. Jašková, Ľ.: How to start with learning informatics at elementary school for blind pupils. In: Trajteľ, Ľ. (ed.) Didinfo 2013, UMB, Banská Bystrica (2013). (in Slovak), ISBN 978-80-557-0527-9
18. Papert, S.: Constructionism vs. Instructionism: speech to an audience of educators in Japan (1980). http://www.papert.org/articles/const_inst/const_inst1.html
19. Jašková, Ľ., Kaliaková, M.: Cognitive abilities of blind pupils of lower secondary education and their use in teaching computer science. In: Trajteľ, Ľ. (ed.) Didinfo 2016, UMB, Banská Bystrica (2016). (in Slovak), ISBN 978-80-557-1082-2
20. Vrábľová, M.: Tactile graphics to support teaching informatics to blind elementary school pupils. Diploma Thesis, FMFI UK, Bratislava (2017). (in Slovak)
21. Jašková, Ľ., Kaliaková, M.: Programming microworlds for visually impaired pupils. In: Futschek, G., Kynigos, C.: Constructionism and Creativity 2014, Proceedings of the 3rd International Constructionism Conference, Östereichische Computer, Gesellschaft, Vienna (2014). ISBN 978-3-85403-301-1
22. Jašková, Ľ., Kováčová, N.: Bebras contest for blind pupils. In: Proceedings of the 9th Workshop in Primary and Secondary Computing Education, London, UK. ACM DL New York (2015). ISBN 978-1-4503-3753-3, http://dl.acm.org/citation.cfm?id=2818324&dl=ACM&coll=DL&CFID=6147908
23. Jašková, Ľ., Kováčová, N.: Contest for blind pupils – universal design of tasks. In: Proceedings of the Conference Universal Learning Design, Linz 2016, pp. 79–97. Masaryk University, Brno (2016). ISBN 978-80-210-8295-3
24. Kováč, M.: Programming environment for visually impaired pupils. Bachelor Thesis, FMFI UK, Bratislava (2016). (in Slovak)
25. Horňanský, M.: Making math accessible for blind in Slovakia through Lambda editor. Diploma Theses, FMFI UK, Bratislava (2009). (in Slovak)
26. Kaliaková, M., Stankovičová, M., Vitálišová Z.: Informatics competences of visually impaired pupils and their use in teaching mathematics and physics with Lambda editor. In: Trajteĺ, Ľ. (ed.) Didinfo 2015, UMB, Banská Bystrica (2015).(in Slovak), ISBN 978-80-557-0852-2
27. Hendl, J.: Qualitative Research. Basic Methods and Applications. (translated into Slovak) Portál s.r.o, Praha (2005). ISBN 80-7367-040-2

Teachers' Expectations and Experience in Physical Computing

Mareen Przybylla[1(✉)], Finn Henning[1], Carla Schreiber[1], and Ralf Romeike[2]

[1] University of Potsdam, Potsdam, Germany
przybyll@uni-potsdam.de

[2] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
ralf.romeike@fau.de

Abstract. In recent years, physical computing has grown in popularity for school education. Children and teenagers learn basic concepts of embedded systems in a creative and motivating manner. We investigate teachers' expectations towards and first experiences with physical computing to gain a better understanding of their needs in professional development. We developed, conducted and evaluated a qualitative interview study, identified typical benefits (e.g. motivation, direct feedback and tangibility), topics (e.g. algorithms and data structures) and challenges (e.g. acquisition of suitable hardware and necessary preparations) that teachers associate with physical computing and derive recommendations for professional development and supporting measures.

Keywords: Physical computing · Interview study · Expectations and attitudes · Classroom experience · Computer science education

1 Introduction

"Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user." [23] *Ubiquitious Computing*, as it was described by Weiser in the early 1990s, has become reality. Embedded and cyber-physical systems are pervasive in our society, computers are everywhere and often we don't even notice their presence. They communicate with their environment and people by using sensors and actuators and are connected to each other and services over the Internet. They enhance our lives in many ways: as little helpers in the household, as control systems in transportation or medicine or as robots to explore areas that are inaccessible for humans. Both, in industry and science, the development and advancement of embedded systems embrace a large and important area of research. The creation of such systems is supported by the large availability of suitable hard- and software tools for all purposes and experience levels, also for non-professional use. Many tools, such as Arduino, Raspberry Pi or the BBC Micro: Bit are aimed at inexperienced and creative developers who use the main concepts of embedded systems and computer science (CS) in different contexts,

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 49–61, 2017.
https://doi.org/10.1007/978-3-319-71483-7_5

for example, for the production of interactive clothing, installations or works of art. The creative design and development of interactive, physical objects and systems is also referred to as *physical computing*. This development is particularly attractive for the educational sector, as it allows for an introduction to embedded systems with low barriers. Also, as creative art and design processes are connected with CS, the common image of computer scientists as 'nerdy' and 'geeky' can be broken through. Physical computing brings together hardware and software components and thus connects the virtual world of computers with the physical world of human experience. In the last few years, various empirical reports and promising research results on physical computing have been published. For example, motivational aspects and the appeal of the subject area for all learners, regardless of age or gender, are often fostered and investigated (e.g. [8,12,17]).

With the integration of physical computing into CS curricula, e.g. England's national curriculum [4] or the new CS curriculum of Berlin/Brandenburg, Germany [20], the demand for professional development for CS teachers has grown significantly. Thus, we developed design principles for professional development and conducted workshops accordingly [15]. From our experience and discussion with workshop participants, we had the impression that they were interested in physical computing and saw positive aspects for their classrooms. However, in follow-up activities, we found that the teachers were very hesitant to apply what they had learned. Therefore, in this study, we pursue the reasons for their reluctance and are particularly interested in the challenges that keep them from implementing projects in their classroom. We describe the development, implementation and evaluation of an interview study that was conducted in a two-day workshop with participants with and without physical computing teaching experiences. We present results about benefits and advantages that teachers attribute to physical computing and topics that they teach using physical computing tools. In particular, we identify challenges and concerns that deter them from comprehensive implementations in the classroom, discuss possible solutions and deduce recommendations.

2 Embedded Systems and Physical Computing in School

Physical computing derives its basic concepts from the field of embedded and cyber-physical systems. *Embedded systems (ES)* combine hard- and software components, are embedded into a technical context and fulfill predefined tasks. They usually control, regulate or monitor a system continuously. For this purpose, in addition to possible user inputs and outputs, sensor data are recorded and commands are sent to actuators. Embedded systems often have to meet real-time requirements (cf. [1,22]). *Cyber-physical systems (CPS)* are networked ES that communicate among themselves and over the Internet [9, p. 5]. ES and CPS are the technological basis for many application areas, such as *robotics*, *wearable computing* or *electronic textiles* and disciplines, such as *physical computing*, *interaction design* or *art* that make use of these core technologies in various, often non-technical contexts (cf. [16]).

Programmable microcontrollers as tools in CS education are an attractive and promising approach to physical computing. This discipline has evolved along with an ever-growing community of artists, designers and hobbyists, who use microcontrollers to create interactive objects and prototypes for these systems. Sensors, such as brightness or temperature sensors and actuators such as servo motors or speakers, are used to continually interact with their environment. *Physical computing* covers the creative design and realization of interactive objects and installations, which are programmed, tangible media that communicate with their environment using sensors and actuators [13]. In physical computing, many methods and ideas of embedded systems design are used.

2.1 Representation in Curricula, Standards and Recommendations

The ACM CSTA K–12 CS standards [18] range from kindergarten to twelfth grade and are well-known in and accepted by the international community. Here, robotics is mentioned as a suitable context for young learners that engages constructionist learning. For teaching algorithms in K–6, among other activities, they suggest to let pupils give algorithmic descriptions to find ways out of a maze, e.g. using toy robots. From a societal perspective, the impacts of ubiquitous and pervasive computing in daily life are discussed. In later stages, robotics is a topic when talking about computers as models of intelligent behavior. In the 2016 revision, the creative and design-oriented parts of CS no longer focus on software development only, but also include the development of physical objects, e.g. prototypical embedded systems [19]. Internationally, also England's national curriculum with the "Computing programmes of study" gained a lot of attention in recent years. All children from ages 5–16 (key stages 1–4) now have mandatory CS education in schools [21]. Already in primary school (key stages 1 and 2), "pupils are equipped to use information technology to create programs, systems and a range of content" [4]. In key stage 2, specifically, they "design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems" [4]. In the accompanying teachers guideline, Computing At School (CAS) recommends programmable toys (e.g. Bee-Bots, Big Traks) in key stage 1 and programmable bricks (e.g. LEGO WeDo) in key stage 2 to let pupils control physical systems and work with sensors, lights and motors.

In Germany, there are recommendations by the German Informatics Society (GI), the so-called educational standards, which today are used in many federal states as a basis when revising curricula [5,6]. For the development of computing systems in class, the GI standards for the upper secondary level recommend identifying and modeling scenarios from real-world contexts. Typical application areas that are listed include *robotics*, *process regulation* and *process control* [6, p. 11]. This development is also evident in recent curricula of some federal states. For example, the Berlin-Brandenburg CS core curriculum suggests the realization of a *physical computing project* and the use of external hardware within the subject area *computing systems* [20, p. 23]. In the field of physical computing, the *characterization of embedded systems*, the *processing of electrical quantities* or the *use of microcontrollers* in different contexts is called for [20, p. 27].

2.2 Realization in the Classroom

Due to its occurrence in the newer curricula, many practical reports cover experience in physical computing activities. Often, however, they lack creative aspects of physical computing and rather fall into the category of robotics. Topics that were earlier done using LEGO Mindstorms robots (e.g. obstacle avoidance or line following) are now realized with different hardware, such as Arduino-based robots (e.g. mBot). The creative potential offered by modern tools, such as Arduino or Raspberry Pi, is often neglected (e.g. in [2,11]). On the other hand, there are also many examples from all over the world, where physical computing is practiced in a creative way—unfortunately, often still in out-of school contexts only (e.g. [3,8]).

3 Professional Development in Physical Computing

We developed a concept for professional development in physical computing to qualify CS teachers in very limited time. They should be empowered to not only implement existing projects, but also to plan their own physical computing lessons tailored to their particular needs and classroom situations, as described in [15]. The concept consists of six design principles for professional development. Important aspects include networking and follow-up experiences to allow teachers to share and discuss their ideas and experiences with colleagues and to offer help with the implementation of pilot projects, support them with hard- and software issues, give advice when needed and, from a research perspective, to gain information about their experience. One year after the first two-day training, we met for an extended workshop where the teachers could connect back to each other and exchange their experience. We also encouraged inexperienced teachers to participate, so that they could learn from their peers.

4 Investigating Teachers' Expectations and Experience

From investigations in our early workshops we found that teachers were strongly interested in physical computing as a topic for their CS classes. They seemed exceedingly motivated despite concerns that they might not have enough time or that technical failure occurred. Interestingly, our impression was that teachers who had done projects with their classes, mostly did so using new tools to teach familiar content (e.g. introduction to programming) – it seemed as if physical computing was used as 'just another teaching method' rather than a new content area. Our experiences from those workshops are described more detailed in [15].

4.1 Aims and Scope of the Study

In the interview study described in this article, the aim was to investigate teacher's expectations, beliefs and attitudes in physical computing (in particular aims, fears, hurdles they see coming in the classroom, mental reservations) and their relationship to teachers' classroom experience to better understand how they can be supported in translating their initial enthusiasm for the subject into lessons with new learning objects. Our research questions are:

- Why are teachers interested, which benefits do they see?
- Which topics and content do they associate with physical computing?
- Which concerns and challenges do they (fear to) face in the classroom?
- What kind of support (teaching aids, materials, education) do they want?

4.2 Setting and Participants

The workshop was held in Brandenburg, Germany, on two consecutive days. Our 15 participants (seven female and eight male) came from different German federal states and Switzerland. Their teaching experience ranged from less than two years to more than twenty years, with approximately one half of the participants having more than ten years of teaching experience. Most teachers came from different types of secondary schools, two participants were university teachers. Concerning their prior physical computing experience, we had a well-mixed group. Eight teachers had already held their own lessons, seven of them gained their basic knowledge in a professional development workshop. The other seven teachers had no classroom experience. Four of them had until then only worked independently on the subject or had taken part in a short workshop. Three teachers had never worked with physical computing tools before, neither in the classroom, nor in any other context. Thus, through the composition of the group of participants, we also had the possibility to draw comparisons between teachers with and without practical physical computing experience (in the following abbreviated to *pE-teachers* and *nE-teachers* respectively, Fig. 1).

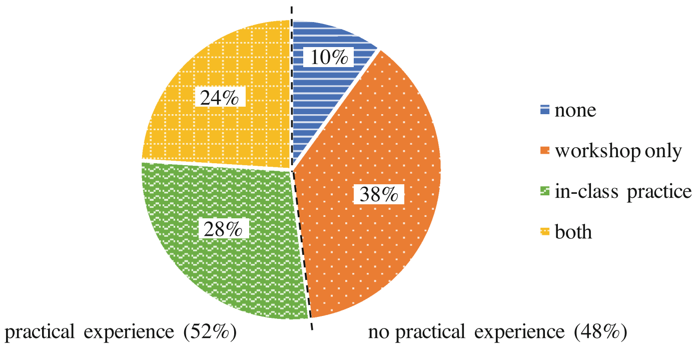


Fig. 1. Participants' physical computing experience

4.3 Methodology and Implementation

We conducted semi-structured interviews in order to react flexibly during the survey with questions of clarification or when new, unforeseen aspects occurred. Each participant was interviewed once during the workshop, starting with nE-teachers to avoid that they were influenced by the pE-teachers' reports. We also collected final impressions in a group interview at the end of the second day. With the design of the interview guideline we adhered to Hussy and Mayring [7,10] and followed a multi-step process from the drafting of a first set of questions, subsequent discussion of this catalog in our working group, a telephone test interview, the revision and re-discussion of the questions, and eventually the design of the final question catalog. In total, about 30 questions and sub-questions were developed and used as interview guide. We evaluated the study using qualitative data analysis methods, as we were interested in the variety of expectations and experiences. We then clustered our participants based on their experience to identify trends of change in their attitudes. During the survey design, we also developed the category system for the data analysis and later, inductively, added categories for items that were not yet represented in the system.

5 Results and Interpretation

5.1 Interest and Benefits

One aspect of our survey was to investigate, why teachers are interested in physical computing and which benefits and advantages they find promising for their teaching. The survey included questions about their general expectations of physical computing in CS education and, more specifically, expectations regarding the possible effects on their pupils, particularly in relation to motivation and gender-specific issues. Our hypothesis that teachers perceive didactic rather than content-related aspects behind physical computing, has been confirmed in the evaluation of responses to the first question set. In general, the vast majority of participants considered it an advantage that through physical computing, pupils got *direct feedback* and CS became *tangible*. It is interesting to see the shift between inexperienced and experienced teachers: some aspects, such as *interdisciplinary teaching* or *motivation*, were mentioned considerably more often by nE-teachers. In contrast, *practical relevance* was more often regarded as a major benefit of physical computing by the pE-teachers. Additional singular responses included *the creation of tangible products*, *social skills* and *creativity*. The imbalance between teachers with and without physical computing teaching experience shows that, e.g. through the experience they have or lack, teachers weigh the importance of different aspects differently (see Table 1).

5.2 Contents and Topics

When we asked the teachers, which contents they consider relevant for teaching physical computing, the overall impression was that they connect quite a variety of topics with it, e.g. programming and algorithms, modeling, sensing and

Table 1. Teachers' expectations concerning benefits in physical computing depending on whether they have practical experience (pE, 8 total) or not (nE, 7 total)

General benefits	# nE-teachers	# pE-teachers	Sum
Creativity	1	0	1
Social competence	0	2	2
Interdisciplinarity	3	0	3
Direct feedback and tangiblity	5	6	11
Practical relevance	2	5	7
Product	1	2	3
Motivation	5	1	6
Project goals	1	0	1

actuation, technical aspects of CS or the Internet of things (IoT). However, our data also shows that in their teaching, teachers mainly connect familiar content with the new tools physical computing offers. This can be inferred when looking at the differences between the two teacher groups, as depicted in Table 2. For example, it is noticeable that CS teachers who have already gained practical experience with physical computing, are more likely to name basic content of CS such as modeling, programming and algorithms (about 85%). Teachers without this experience proportionally more often mention specific content that is more closely linked to physical computing, such as sensing and actuation or the IoT (about 50%).

Table 2. CS content relevant in physical computing teaching as mentioned by teachers depending on whether they have practical experience (pE, 8 total) or not (nE, 7 total)

CS content	# nE-teachers	# pE-teachers	Sum
Computer engineering	0	3	3
Embedded systems	0	0	0
Internet of Things	1	0	1
Interactive systems	0	0	0
Sensing and actuation	3	4	7
Measurement, regulation and control	1	0	1
Modeling	0	2	2
Programming and algorithms	4	9	13
Other	1	4	5
sum	10	22	32

In a second question, we examined, to what extent teachers would integrate the following topics into their lessons: *embedded systems*, *sensing and actuation*, *interactive systems* and *IoT and smart objects*. When counting the topics that teachers consider important and want to talk about in their lessons, interestingly, sensing and actuation is the only content area, which is mentioned more often by pE-teachers than by nE-teachers. All other topics are dominated by the other group. Given the small number of interviewees, these findings should not be overrated. However, it might be concluded that teachers – although willing to integrate new topics in the beginning – find it difficult to prepare those contents for their pupils or have other obstacles that prevent them from doing so. When looking more closely into their statements[1], it becomes evident that the main reasons for their hesitance are:

– *Lack of knowledge:* (e.g. "Embedded system is still like a foreign word for me, I have no idea what that is.")
– *Topic not in focus:* (e.g. "Well, so far, sensing and actuation and interactive systems. I never thought about the others.")
– *More suitable in different context:* (e.g. "The Internet of things, for me, is more connected to home technologies than physical computing.")
– *Not in curriculum:* (e.g. "Well, these are no topics that can directly be found in the curriculum.")

As most of the existing curriculum documents are recommendations or roughly delineated, teachers require more concise descriptions and explanations of relevant contents and their interrelations. In other subject areas, there are textbooks and collections of instruction material that have been created over the years. For physical computing, such aids are not yet available. Thus, teachers need support in the concrete design and implementation of teaching scenarios.

5.3 Concerns and Challenges

One of the main goals of the survey was to find out what problems CS teachers expect to occur in physical computing teaching, which challenges they actually face in the classroom and how they cope with these hurdles. Here, it is particularly interesting to look at differences between pE- and nE-teachers.

Among the general answers to this question, *time* was seen as a major issue by the nE-teachers. Among the pE-teachers, this issue was mentioned only once. However, it is not always clear if teachers fear a lack of time in the classroom or high preparation times. In the first case, anchoring physical computing in the curricula might help. In both groups, *organizational complexity* is seen as a challenge when planning physical computing projects. pE-teachers often report how they coped with issues such as finding storage room for the unfinished projects or receiving funds to buy necessary hardware components. *Technical difficulties* were mentioned equally often by both groups and there seems to be a tendency that pE-teachers experience hardware issues more often than other technical problems (Table 3).

[1] All statements were translated from German by the author.

Table 3. Teachers' concerns and perceived challenges in physical computing teaching depending on whether they have practical experience (pE, 8 total) or not (nE, 7 total)

Perceived challenges	# nE-teachers	# pE-teachers	sum
Technical difficulties	4	3	7
Organizational complexity	3	5	8
Unrealistic project goals	0	1	1
Financial difficulties	2	1	3
Time	5	1	6
Too much crafting	1	2	3
None	0	1	1
sum	15	14	29

We posed additional questions for some specific problems that we were confronted with in our previous workshops: When inquired in more detail, nE-teachers also mentioned several times that they were afraid their pupils might not be experienced enough to use the necessary hardware tools. Only one of the pE-teachers confirmed this worry. Many teachers of both experience groups fear that components of construction kits might break or get lost in class. Not all pE-teachers support this – pupils, according to one teacher "[...] have a natural respect for things that are expensive. It's the same with a laptop, they wouldn't throw it on the floor." However, there was a tendency that teachers do not regard this a problem in Gymnasium[2], but rather in other secondary school types that do not focus on academic learning (Fig. 2). Other challenges were only mentioned once or twice and seemed to be of minor importance (e.g. *unrealistic project goals* or *too much crafting* in the classroom).

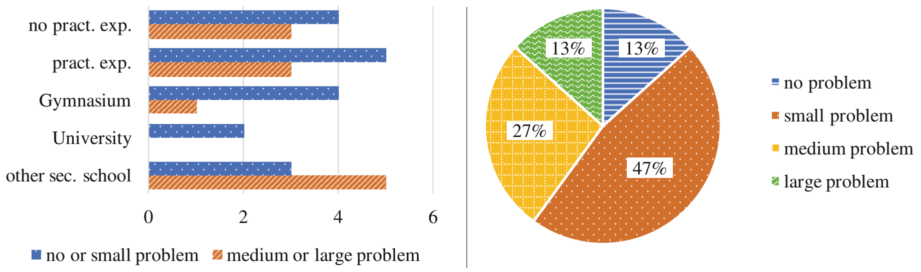


Fig. 2. Teachers' concerns that students loose or break parts of construction kits

[2] Gymnasium: advanced secondary school that leads to Abitur (A-Level equivalent), compares to grammar schools (UK) or preparatory high schools (US).

Table 4. Support strategies and means for teachers depending on whether they have practical experience (pE, 8 total) or not (nE, 7 total)

Category	Code	# nE-teachers	# pE-teachers	sum
Strategies	collaboration	4	0	4
	professional development	1	0	1
	networking	2	0	2
	detailed planning	2	4	6
	other	3	1	4
Means of support	financial support	2	1	3
	best practice examples	5	2	7
	lesson plans	2	4	6
	textbooks for teaching	3	1	4
	other	2	4	6

5.4 Support

We also tried to figure out how teachers can be supported in avoiding problems in the future. They gave different answers depending on the particular problems. While nE-teachers strive for *networking* opportunities, *professional development* and *collaboration* both with colleagues and external partners, in contrast, pE-teachers also emphasize the need for detailed *descriptions and manuals* and *elaborate lesson plans* (Table 4). Well-developed, tested and *reliable tools* seem to be really important for successful teaching. If teachers have to deal with technical difficulties, this is often perceived as problematic. However, there were also many of the teachers' initial concerns that were not confirmed by the pE-teachers. Thus, networking meetings, where experiences are exchanged and experts are available for questions are important, especially for inexperienced teachers. Guidelines and teaching aids can help during the implementation phase, scaffolding the process of physical computing lesson planning. In the last question, we asked teachers what they would find helpful with the implementation of physical computing projects. In general, the answers reflect the findings from the previous sections pretty well: example projects and lesson plans are mentioned most frequently, directly followed by text books. Financial support and networking were mentioned twice, each. Additional aspects were, for example, *classroom-suitable construction kits*, more workshops and professional development opportunities, *tutorial videos* and ideas how to deal with *grading* (category "other" in Table 4).

6 Conclusions

In this study we analyzed teachers expectations, attitudes and experience with physical computing to gain a better understanding of their aims and needs in teaching. Our key findings are:

1. Interest in the subject mainly stems from pedagogical considerations: Benefits of physical computing are seen in motivation, direct feedback and tangibility of CS and not primarily in new contents related to the field. This is, partly at least, due to the fact that many publications focus on the aforementioned aspects only.
2. Related to the first finding, teachers mainly associate traditional topics of CS with physical computing: pE-teachers reported they were using physical computing tools to teach in content areas such as algorithms and programming and not, as one might expect, topics such as embedded systems or IoT. The reasons given suggest that more professional development opportunities are required in the new subject areas and that these topics must also be anchored in curricula.
3. Participants suspected and experienced a high organizational effort: nE-teachers saw most of the challenges in the acquisition of hardware and materials, the greater complexity of the necessary preparations, and the overall effort involved in the development and adoption of new processes and routines. The pE-teachers added challenges in project management, organization of the classroom, as well as the storage of the components and unfinished projects. There is a need for practical solutions to these problem areas, in particular to reduce the initial barriers for hesitant teachers.
4. Teachers need support in many areas: above all, they call for example projects that give orientation, but also the development of suitable tools, classroom materials and manuals are required, as well as networking opportunities.
5. Interdisciplinary teaching is hardly practiced: collaborations are primarily pursued with art and physics teachers, often combined with the aim of outsourcing the craft work on the projects from computer science classes. Again, suggestions and practical examples are required.

Some work has already been done in all of these sectors. There are tools available on the market, some of them more and some of them less suitable for education (cf. [14]), many teachers piloted physical computing projects and developed materials. In general, however, more needs to be done in all the mentioned areas: curricula need revision, tools need improvement and professional development is required that goes beyond an introductory course on tool operation and especially focuses on deeper examinations of the new content. Only then we can provide successful education in the field of embedded systems, which enables our pupils to appropriately participate in the digital society.

References

1. BITKOM: Eingebettete Systeme – Ein strategisches Wachstumsfeld für Deutschland [Embedded Systems – A stratgic growth area for Germany] (2010). https://www.bitkom.org/noindex/Publikationen/2010/Leitfaden/Eingebettete-Systeme-Anwendungsbeispiele-Zahlen-und-Trends/EingebetteteSysteme-web.pdf. Accessed 07 Sept 2017

2. Brinkmeier, M., Kalbreyer, D.: A case study of physical computing in computer science education. In: Vahrenhold, J., Barendsen, E. (eds.) Proceedings WiPSCE 2016, pp. 54–59. ACM, New York (2016)
3. Buechley, L., Eisenberg, M., Elumeze, N.: Towards a curriculum for electronic textiles in the high school classroom. ACM SIGCSE Bull **39**(3), 28 (2007)
4. Department for Education: Computing programmes of study: key stages 1 and 2. National curriculum in England (2013)
5. Gesellschaft für Informatik e.V.: Grundsätze und Standards für die Informatik in der Schule [Principles and standards for computer science at school]. LOG IN 28(150/151), supplement (2008)
6. Gesellschaft für Informatik e.V.: Bildungsstandards Informatik für die Sekundarstufe II [Educational standards for computer science in upper secondary education]. LOG IN 36(183/184), supplement (2016)
7. Hussy, W., Schreier, M., Echterhoff, G.: Forschungsmethoden in Psychologie und Sozialwissenschaften für Bachelor. Springer, Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-34362-9
8. Kafai, Y.B., Lee, E., Searle, K., Fields, D.: A crafts-oriented approach to computing in high school: introducing computational concepts, practices, and perspectives with electronic textiles. ACM T Comput. Ed. **14**(1), 1–20 (2014)
9. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach, 2nd edn. MIT Press, Cambridge (2017)
10. Mayring, P.: Qualitative Content Analysis. Theoretical Foundation, Basic Procedures and Software Solution. Klagenfurt, Austria (2014). http://nbn-resolving.de/urn:nbn:de:0168-ssoar-395173
11. Neutens, T., Wyffels, F.: Teacher professional development through a physical computing workshop. In: Proceedings WiPSCE 2016, pp. 108–109, no. 6 (2016)
12. Przybylla, M., Israel, P., Streichert, J., Romeike, R.: Bridging motivation gaps with physical computing in CS education. In: ISSEP 2016, p. 53 (2016)
13. Przybylla, M., Romeike, R.: My interactive garden-a constructionist approach to creative learning with interactive installations in computing education. In: Kynigos, C., Clayson, J.E., Yiannoutsou, N. (eds.) Constructionism: Theory, Practice and Impact. Proceedings of Constructionism 2012, pp. 395–404 (2012)
14. Przybylla, M., Romeike, R.: Key competences with physical computing. In: Proceedings of Key Competencies in Informatics and ICT (KEYCIT) 2014. Universitätsverlag Potsdam, Potsdam (2014)
15. Przybylla, M., Romeike, R.: Teaching computer science teachers - a constructionist approach to professional development on physical computing. In: Sipitakiat, A., Tutiyaphuengprasert, N. (eds.) Proceedings of Constructionism 2016 (Constructionism 2016, Bangkok, Thailand), pp. 265–274. Suksapattana Foundation (2016)
16. Przybylla, M., Romeike, R.: Social demands in ubiquitous computing : contexts for tomorrow's learning. In: Webb, M., Tatnall, A. (eds.) 11th World Conference on Computers in Education. Springer International Publishing, Dublin (2017)
17. Richards, M., Petre, M., Bandara, A.K.: Starting with UbiComp: using the senseboards to introduce computing. In: 43rd ACM Technical Symposium on Computer Science Education, pp. 583–588. ACM, Raleigh (2012)
18. Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., Owens, B.B., Stephenson, C., Verno, A.: CSTA K–12 Computer Science Standards. Technical report, New York (2011)
19. Seehorn, D., Primann, T., Lash, T., Lee, I., Moix, D.: [INTERIM] CSTA K-12 Computer Science Standards. Technical report, New York (2016)

20. SenBJW Berlin, MBJS Brandenburg: Teil C Informatik Wahlpichtfach [Part C Computer Science Elective Course] (2015)
21. Sentance, S., Humphreys, S.: Online vs face-to-face engagement of computing teachers for their professional development needs. In: Brodnik, A., Varenhold, J. (eds.) Informatics in Schools. Curricula, Competences, and Competitions, vol. 9378, pp. 69–81. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_7
22. Vahid, F., Givargis, T.D.: Embedded System Design: A Unified Hardware/Software Introduction. Wiley, New York (2002)
23. Weiser, M.: Some computer science issues in ubiquitous computing. Commun. ACM **36**(7), 75–84 (1993)

Teacher Development and Assessment

Investigating Informatics Teachers' Initial Pedagogical Content Knowledge on Modeling and Simulation

Nataša Grgurina[1(✉)], Erik Barendsen[2,3], Cor Suhre[1], Klaas van Veen[1], and Bert Zwaneveld[3]

[1] Teaching and Teacher Education, University of Groningen, Groningen, The Netherlands
{n.grgurina,c.j.m.suhre,klaas.van.veen}@rug.nl

[2] Radboud University, Nijmegen, The Netherlands
e.barendsen@cs.ru.nl

[3] Open University, Heerlen, The Netherlands
zwane013@planet.nl

Abstract. Computational science, comprised of modeling and simulation, is a new theme in the new 2019 Dutch secondary education informatics curriculum. To investigate the pedagogical content knowledge (PCK) on modeling and simulation, we interviewed ten informatics teachers and analyzed their PCK, distinguishing its four elements - knowledge of goals and objectives, students' understanding, instructional strategies and assessment - and investigated potential differential features of their PCK in order to typify teachers' individual PCK. We charted the teachers' PCK in terms of these four elements and found differential features related to knowledge of goals and objectives and related to knowledge of assessment, dividing these teachers into four distinct groups. However, these differential features do not lead to distinct types of PCK. Our findings will be used to explore the future development of teachers' PCK and they will contribute to the development of teaching materials, assessment instruments and teacher training courses on modeling.

Keywords: Pedagogical content knowledge · Computational science · Modeling and simulation · Informatics · Secondary education

1 Introduction

Modeling plays a significant role in the development and learning of science [13] and informatics provides the means for students to actively engage in learning science by providing tools and techniques to engage in modeling. The new 2019 Dutch secondary education informatics curriculum recognizes this and includes an elective theme comprised of modeling and simulation, together called *Computational Science*. It is described by the high-level learning objectives: *"Modeling: The candidate is able to model aspects of a different scientific discipline in computational terms"* and *"Simulation: The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field."* Modeling itself will be a part of the compulsory core curriculum, described as *"Modeling: The candidate is able to use context to analyze a relevant problem, limit this to a manageable problem, translate this into a model, generate and interpret model results, and test and assess the model. The candidate is able to use consistent reasoning"* [2, 8]. The curriculum does not provide further details about these objectives, instruction or assessment. In line with the Dutch tradition, this is left to educators and authors of teaching materials. The elaboration of this learning objective, the development of teaching materials, assessment tools and teacher training courses are already taking place and we intend to monitor these developments.

This study is a part of a larger research project on teaching Computational Science in the context of informatics in Dutch secondary education, investigating pedagogical aspects and teachers' pedagogical content knowledge (PCK) about modeling[1]. Following Magnusson et al. [15], we distinguish four elements of content-specific pedagogy: (M1) goals and objectives, (M2) students' understanding and difficulties, (M3) instructional strategies, and (M4) assessment. Previously, we refined the CSTA definition of computational thinking (CT) [5], made initial explorations of teachers' PCK [6, 7] and of the computational modeling process [9], obtained an operational description of the intended learning outcomes of the learning objective Computational science – thus focusing on Magnusson's element M1, observed students working on modeling tasks - focusing on Magnusson's element M2, and established what data sources were suitable for assessment - Magnusson's element M4 [8].

Aim of this study. In this study, we focus on teachers' PCK on modeling. We address the following research questions:

1. How can the teachers' PCK be portrayed in terms of the four elements of PCK?
2. What differential features of PCK can be used to typify teachers' individual PCK in terms of the four elements of PCK?

Our findings will serve as input for the subsequent studies on designing teaching materials and assessment instruments, the development of teachers' PCK, and they will contribute to the development of teacher training courses.

Related Work. The construct of PCK has proven to be a powerful one to help capture teachers' views and knowledge on teaching various topics in STEM disciplines such as models and modeling in science [13] and as a part of public understanding of science [11], programing in informatics [17] and designing digital artifacts in informatics [16]; to expose the relation between the quality of teachers' PCK and their subject matter knowledge [17, 18] to explore the PCK repertoire of beginning teachers [14] and to chart the development of teacher's PCK as the experience with teaching a particular topics increases [10].

We investigate PCK through the lens of the operational description of the intended learning outcomes of the learning objective Computational science obtained in our previous study, describing the modeling cycle for simulation modeling through its

[1] In this paper, the terms *modeling*, *simulation modeling* and *computational science* all refer to the learning objective computational science.

elements *purpose*, *research*, *abstraction*, *formulation*, *requirements/specification*, *implementation, verification/validation, experiment*, *analysis*, *and reflection* [8].

Context of the study. In the Netherlands, informatics is an elective subject in grades 10 and 11 of the senior general secondary education spanning grades 7 through 11 (in Dutch: HAVO) and in grades 10 through 12 of the pre-university education spanning grades 7 through 12 (in Dutch: VWO). In grades 10 through 12, all students follow either one of the two humanities tracks or one of the two science tracks and they can all elect to follow informatics if their school offers it. There are two routes to teachers' qualification for informatics. When informatics was introduced in secondary education, in the early 2000's, teachers of other subjects could get qualified through in-service training. As of 2006, qualification entails an MSc degree in informatics education [4].

2 Method

We conducted individual semi-structured interviews with ten informatics teachers from the local informatics teachers' network who replied to our invitation to be interviewed. Four of the teachers have an informatics background and three of these are qualified teachers with an MSc degree in informatics education while the fourth one is still studying to get this qualification. Out of the other six teachers, one has no teacher qualification, one is qualified for mathematics only, and other four are qualified teachers for other subjects - such as physics or history - who additionally attained teacher qualification for informatics through in-service training. The teaching experience of these ten teachers ranges from a few months to several decades. Eight of these teachers were about to take a course on agent-based modeling. Interviews lasted between half an hour and an hour, depending on the extent of a particular teacher's PCK. In our interview protocol, we first enquired about the teachers' educational and professional background and whether they had already taught modeling. We then asked detailed question arranged around the four PCK elements described by Magnusson, that were inspired by [11–13, 16].

- M1: Goals and objectives
  What comes to mind when you hear the word 'model'? In which context(s) does this word make sense to you? [12] What are models for? In which circumstances are they used? [12] What do you expect to be your main objective in teaching modeling and simulation in informatics? [11] Why do you intend to teach this to your students in informatics? What do expect you will like or dislike about modeling projects by your students? [16]
- M2: Students' understanding
  Do your students need any specific prerequisite knowledge to be able to learn about modeling and simulation in informatics? [11] What sorts of skills do students need to acquire in order to be able to develop models and run simulations? [16] What do you expect to be successful for your students? [11, 16] What do you expect to be difficulties for your students? [11, 16] What do expect your students to actually learn from their modeling (and simulation) projects? [16]

- M3: Instructional strategies
  In what activities and in what sequence do you expect your students to participate in the activities of learning modeling and simulation? What to teach students to achieve the modeling objectives? [16] How to teach students to achieve the modeling objectives? [16] What do you expect to be your role as a teacher when teaching about modeling and simulation? [11] What do you expect are going to be the teaching difficulties/problems concerned with the modeling projects in your classroom? [16] What technological tools do you intend to use in your classroom? [16]
- M4: Assessment
  How do you intend to assess your students' learning and achievement during their modeling projects? [16] How do you intend to establish whether your students reached the learning goals with regard to modeling and simulation? How would you know? [11]

The interviews were recorded and transcribed verbatim. We first coded the transcripts using the coding categories derived from our operational description of modeling [8]. We then classified the interview transcripts using Magnusson's four elements of PCK [15] as main coding categories. Within these categories, we applied inductive coding to characterize the teachers' responses. In an axial coding process [3], the codes were grouped and merged where necessary. We used the codes to describe the teachers' PCK. In a subsequent analysis, we tried to identify differential features with the purpose of typifying teachers' individual PCK.

3 Results

In this section, we first present the results of our characterization of PCK organized around the four elements of PCK. We then explore differential features in order to distinguish types of teachers' PCK.

3.1 Knowledge About Goals and Objectives (M1)

No teacher has taught computational science as a separate topic in the context of the informatics course yet and only one of them taught system dynamics modeling in a physics course. Since we did not enquire about the modeling process explicitly, we performed a detailed analysis of the interviews through the lens of our theoretical framework on the modeling cycle and thus reconstructed the teachers' content knowledge (CK) pertaining to the aspects of modeling, as shown in Table 1.

Concerning the nature of models, six teachers said that models were a simplified representation of reality, three reported that models were something to be used for calculations, two saw models as something for visualization and communication, and one also mentioned physical models. Regarding the contexts for the use of models, the teachers mentioned information systems, scientific research and cited several examples (e.g. exploring group forming on ethnical basis). Several teachers stressed the fact that informatics served other disciplines - an idea that persistently permeates their thinking about modeling and teaching modeling.

Table 1. Teachers CK on modeling cycle

Aspect	Teacher									
	1	2	3	4	5	6	7	8	9	10
Purpose		✓		✓			✓	✓	✓	✓
Research									✓	
Abstraction	✓								✓	✓
Formulating	✓			✓			✓	✓	✓	✓
Requirements/specification	✓			✓						
Implementation				✓		✓	✓		✓	
Verification/validation	✓	✓	✓		✓	✓		✓		✓
Experiment	✓	✓		✓					✓	✓
Analysis				✓			✓	✓	✓	
Reflection	✓	✓		✓		✓	✓		✓	✓

According to the teachers, the objectives of teaching modeling are twofold:

- **Conceptual objectives:** these objectives emphasize learning to master skills associated with informatics subject matter. Teachers mention CT aspect automation, software design cycle, the necessary research skills, analysis of the world the students live in, linking and translating reality to model through abstraction, building a model and, in words of teacher 9, *"using it [...] so that you can predict things or test things, or ... ehm ..., understand things"*.
- **Motivational and practical objectives:** these objectives focus on students' engagement, motivation, attitude, skills, practical benefits, insight and awareness about relevance. Teachers mention perseverance, building confidence about students' own ability, developing self-reliance, and realizing that models are useful tools that can be used in specific situations. In this light, learning to model helps to enhance students' insight, it helps develop cooperation and communication skills, it is interesting or fun, it serves as preparation for the future, and it concerns "soft informatics". Furthermore, modeling lays a connection between informatics and other disciplines and models could be useful for students' subsequent studies. In words of teacher 10, *"... that you need to be able to design a test setup with informatics, ehm, digital means so that you can get a lot of results, but processing these results is no more informatics, that's physics."*

3.2 Knowledge of Students' Understanding (M2)

The prerequisite knowledge needed to learn modeling and skills needed to make models are related to:

- **Student's characteristics:** such as age and development, mindset, attitude, and other skills. For example, teacher 6 believes *"it is all very complex, they [students] are only 15 years old"* and teacher 7 is reminded of Bloom's taxonomy and fills in *"it gets easier as they get older"*. Fantasy, creativity and an analytical mindset also play a role. Attitude is important: several teachers stress the significance of perseverance. Some mention differences among students: teacher 6 expects to see different skill levels when it comes to tackling practical assignments.

- **Student's knowledge pertaining:** the other discipline, programming or computational thinking, and modeling aspects. Teachers 4, 5, 6 and 8 expect the students to know something about the phenomenon from other discipline they are modeling. Almost all the teachers expect their students to be familiar with programming before they embark on modeling. However, teacher 9 also sees the possibility to use modeling as a vehicle to teach programming. Teachers 4, 8 and 10 expect their students to be familiar with several modeling aspect such as being able to explain how a particular model works, abstraction, and in words of teacher 8, *"you need to learn to recognize the actors and you need to be able to see the relations among them, to ...ehm ... to describe them, to translate them into a model".*
- **The chosen teaching approach:** teacher 8 mentions the interplay between the teaching materials used and the necessary prerequisite knowledge and says *"but you need to sense what extras you should offer"* and adds, *"It's quite abstract so I think about the 12th grade, but if you present it simply enough, then you could teach it in the 10th grade as well, without ever having taught any informatics before that".* He also stresses the importance of re-activating the knowledge the students already possess.

The issues teachers regard as successful:

- **Relevance:** through modeling, learning about familiar phenomena encountered in other courses (e.g. biology, economy) rather than distant ideas. As teacher 6 puts it: *"You concoct nice stories about nuclear power plants, but how many children end up in there?"*
- **Perception:** attitude, experiencing success, interest and fun, and student's characteristics. Several teachers expect students' confidence would grow through their perseverance when facing problems and experiencing success by making a model on their own. Teacher 2 admits that achieving this with his students poses a challenge. Numerous teachers expect these aspects, together with the relevance experienced by students and the possibility to come up with creative solutions and implementation of students' own ideas, to make modeling interesting and fun. Teacher 10 mentions Gardner's multiple intelligences to explain why he expects students in science tracks to perform differently than students in humanities tracks.
- **Skills:** programing and computational thinking. Most teachers expect that programming the models would not pose a problem.
- **Organizational issues:** teaching strategies to meet students' needs. Several teachers describe how to contribute to the students' success by choosing a suitable teaching approach. A number of teachers who intend to use NetLogo to teach modeling prize its user-friendliness and see this as a success factor.
- **Interest and fun:** Teacher 5 says, *"even if I never teach this, I still find it fun for myself"* and goes on to reflect how to transfer this enthusiasm to his students.
- Finally, teachers 1, 3, 4 and 9 do not know what to say. As teacher 4 puts it, *"I've never taught modeling, so I wouldn't know what would be a success."*

The issues teachers regard as difficult:

- **Technical issues:** programing and computational thinking, aspects of modeling and development. For example, the meaning of the term *parameter* in physics or mathematics differs from the meaning in a simulation model and that might be confusing. Students who follow physics course are already familiar with models, as opposed to students in humanities tracks. Teacher 2 expects that in the beginning, students will have difficulties understanding existing models and how their components interact. Teacher 4 expects her students to have problems getting used to the programming language and subsequently to have problems programming. Numerous teachers expect that modeling aspects such as abstraction – deciding what is relevant for a model and what to leave out - will be difficult. Some teachers expect problems with implementation – translating conceptual model into program code. Teacher 7 expects his students could be too ambitious with the models they want to make and suggests keeping an eye on his students all the time to be able to intervene and help in case they encounter any of these problems.

- **Perception:** attitude, skills, interest or fun, relevancy, age and development. Teacher 9 expects problems with motivation if the students do not see the relevance of modeling. He also mentions lack of perseverance and inability to go on after getting stuck. Teacher 6 believes that abstract aspects of modeling are difficult for the students of this age – 16 years old. He also mentions students not using common sense.
- **Approach**: work method, possibility to work on their own case. Some teachers expect problems with students who dive right into building their models without giving it sufficient thought first, and, with students who lack oversight and do not know where to begin. According to teacher 9, the last problem could be alleviated by good teaching material. Several teachers believe some students would have difficulties coming up with a suitable case to model.
- Again, some teachers do not know what to say.

3.3 Knowledge About Instructional Strategies (M3)

When asked about their teaching approach, some teachers are cautious about replaying due to lack of experience, but in the end all the teachers have similar ideas that can be summarized as follows: during a period ranging from six to twelve weeks, scaffold learning by beginning with an introduction about models in general, show and explain the working of few models and their code, then give students several assignments to expand existing models or develop simple models from scratch, possibly differentiate to account for students interests, level (HAVO or VWO) or the track they follow (humanities or science), and finally, engage them in a large (group) project where they develop a model from scratch during several weeks. While describing their teaching approach with various degree of detail, the teachers report about:

- **Their role as a teacher:** to coach, to explain, to show, to help, to encourage, to keep an eye on students' progress, and simply to be there. Teacher 9 says, *"and then you come in there to steer. And if it gets stuck, not just to answer the question, but to help them find the answer themselves"*. Teacher 5 is cautious: *"well, when they're working on a new model, I don't always have a ready-made answer."* Teacher 7 would have his students recreate a small program to check their understanding before they embark on the large project. During the project, he would have a double role, both as the teacher and as a client. As the teacher, he would keep an eye on the progress of the whole project. As the client, he would come in every two weeks and tell his students things like *"hey, you have things that make no sense"* and have students fix the problems themselves.

- **Assignments the students work on:** open and closed problems and making models. Teacher 6 would have a number of closed problems together with answers for students to practice before embarking on the open problem they have to work on as their final project. For the final project, most teachers would let their students come up with their problems themselves, but would also have a list of problems available for those who cannot think of something themselves.
- **Student's characteristic to take into account:** level and the track they follow. Teacher 8 would have different assignments for students to practice on, catering to their needs and preferences, depending on the educational track they follow. Teacher 2 would not require his HAVO students to develop a model from scratch, but would rather have them expand and adjust an existing model.
- **Organizational aspects:** the daily teaching practice, planning, organizing, SCRUM, rapid prototyping and playing the role of the client. Teacher 3 teaches 60-minutes lessons and would start each lesson with a short central instruction and let the students work for themselves the rest of the time. He would use Trello boards for planning and, like teacher 9, employ SCRUM to organize the work. Teacher 10 would send his students to a teacher of a different subject who would then pose as a client, while he himself would be a process supervisor. Teacher 9 would pose as a client himself.
- **Difficulties and problems:** technical problems, problems related to teaching materials and other problems. Teacher 9 warns that no matter how simple the software used is, there is always a possibility of getting an error message, not understanding it and then getting stuck.

3.4 Knowledge About Assessment (M4)

When asked about assessment, all teachers agree that the large practical assignment the students worked on to learn modeling could serve for the assessment purpose as well. Teacher 9 could possibly give his students a written exam instead. Teacher 8 would use a small written exam to ascertain the students learned enough about modeling before they are allowed to start working on a large project. When talking about assessment, the teachers report on:

- **Assessment form:** written exam – formative and summative, project to be done individually or in groups.
- **Problems to work on:** given by the teacher or provided by the students themselves, open or closed, opportunity for differentiation.
- **Organizational issues**: SCRUM and rapid development.
- **Assessing the work**: quality of the end product and project documentation, aspects of modeling cycle, teacher's impression about the students' activities in the lessons during work on the project. When assessing the results of the final project, most teachers want to look into the quality of the model - and some of them the code too - as described by the students in the project documentation they are required to turn in. The teachers mention various aspects of the modeling process as relevant for the

assessment. However, no clear quality criteria are elaborated and teacher 1 would simply estimate the quality of the project. Teacher 8 adds that technical aspects are easy to assess, but it is difficult to see if the students realize the full spectrum of possibilities the modeling offers to them. Some teachers would rely mainly on their observation of students while they work on the projects, again without elaborating on specific quality criteria. Additionally, teacher 10 would talk to his students to ascertain whether they understand what they are doing. Teacher 6 would also assess the quality of the report the students wrote for the customer. Teacher 7 would have his students present their models, he would assess the product (i.e. model and project documentation) and additionally the process (SCRUM) and he stresses the importance of reflection, together with number of other teachers. He adds, *"even if they didn't succeed, I find you can't say they don't know what modeling is"*. Teacher 5 is not sure what to say: "*if the model works, is that sufficient?"*

3.5 Differential Features and Typification

We found two characteristics that distinguish among teachers:

- **Knowledge of goals and objectives (M1).** Teachers 1, 2, 4, and 6 stress the importance of *conceptual objectives* such as learning how to employ programming or how to make models, while teachers 3, 5, 7, 8, 9 and 10 put more emphasis on *broader motivational and practical objectives* such as enhancing insight or preparing for the future.
- **Knowledge of assessment (M4).** Teachers 1, 2, 4, 8, 9 and 10 predominantly put to use *product-based assessment* stressing the quality of the students' product, while teachers 3, 5, 6 and 7 prefer *process-based assessment* stressing the importance of the employed working procedures.

Combining these two differential features leads to four groups of teachers as shown in Table 2:

Table 2. Distinct groups of teachers

		M1	
		Conceptual objectives	Practical and motivational objectives
M4	Product-based	1, 2, 4,	8, 9, 10
	Process-based	6	3, 5, 7

Our analysis revealed, however, that the knowledge of students' understanding (M2) and instructional strategies (M3) varies within each of these four groups, so the above differential features do not give rise to a typification of the teachers' overall PCK.

4 Conclusion

In answering the **first research question** - *How can the teachers' PCK be portrayed in terms of the four elements of PCK?* - we portrayed each of these elements:

Concerning teachers' **knowledge about goals of objectives** on teaching modeling, we charted teachers' content knowledge and described learning objectives in terms of conceptual, and motivational and practical objectives.

Concerning the **knowledge about students' understanding,** the teachers reported on the prerequisite knowledge, attitudes, skills, abilities and various approaches students have to learning, in line with Magnusson et al. [15]. However, when talking about the difficulties, they mentioned problems due to the abstract nature of modeling and inefficient students' strategies, but no teacher mentioned misconceptions.

Concerning the **knowledge about instructional strategies**, we see an agreement about subject-specific strategies [15] – scaffolding learning with a final project which serves both to give students the opportunity to learn how to develop a model from scratch and as assessment.

Concerning **knowledge of assessment**, there is an agreement about a suitable form – a large practical assignment. Teachers mentioned a range of assessment criteria focused on the quality of students' products and teachers' impressions of the students work process. In contrast to the uniformity regarding the form of assessment, there is a great variation in granularity and depth of their description of assessment criteria. We saw similar diversity regarding the knowledge of dimension to assess [15], i.e. the aspects of modeling process.

In answering the **second research question** - *What differential features of PCK can be used to typify teachers' individual PCK in terms of the four elements of PCK?* - we found two characteristics that distinguish among teachers: their focus on conceptual versus motivational and practical objectives (M1) and their emphasis on product-based versus process-based assessment (M4) leading to four distinct groups of teachers. However, none of these differential features leads to an overall typification of the teachers' PCK.

5 Discussion

Reflections on findings. As a possible explanation for the variations found in the teachers' PCK, we explore the relation with the teachers' background. Not surprisingly, we saw that teachers 1, 2, 4 and 9, who all have a background in informatics, displayed rich knowledge of modeling. Teachers 1 and 4 - both young teachers - had limited idea about what to expect from their students beyond the general remarks about students needing to plan before acting and lack of perseverance. On the other hand, teachers 2 and 9 exhibited rich knowledge of students' understanding and related their extensive knowledge of instructional strategies to it. Teachers 6, 7, 8, and 10 possess rich and well-connected PCK. Finally, we saw that teachers 3 and 5, despite their limited content knowledge, were able to relate their general knowledge of their students to their teaching strategies. The knowledge of their students' understanding and difficulties was the strongest component of their PCK. Other components of their PCK were weaker and so were the relations among these components too.

Regarding the instructional strategies, the teaching approach described here is in line with prevailing informatics teaching practices in the Netherlands. The extent of knowledge of topic-specific strategies [15] varies across teachers and seems to be related to their subject matter knowledge and teaching experience. The findings about young teachers 1 and 4 are in line with the results by Lee et al. [14] who found that *"a strong science background does not guarantee a proficient level of PCK."* The more experienced teachers sometimes behave like novices too, – e.g. teacher 5 – while in others, their extensive PCK seems to sustain them in the non-familiar area of modeling, in line with results of Borko et al. [18] who found in a similar situation that rich PCK for general science topic seems to sustain teachers *"in whatever content they are teaching"*.

We observed two characteristics allowing us to distinguish among teachers that were observed by Rahimi et al. [16] as well. However, our findings are not completely in line with these findings because in our case, for example, teachers 1 and 10, both leaning toward predominantly product-based assessment, require students to keep logbooks to document the modeling problems, difficulties and dead ends they encountered. On the other hand, teacher 6 would take customer's feedback into account and teacher 7 would have his students present their work in the class, while they both lean toward predominantly process-based assessment. Unlike Rahimi et al., we were not able to typify teachers' PCK through relating their knowledge of students' understanding and instructional strategies on one hand, to their knowledge of goals and objectives and knowledge of assessment on the other.

Remarkably, despite the great variation of assessment criteria mentioned, there is hardly any evidence of quality indicators used to judge to what extent these criteria are met and to what extent the students are able to apply the elements of modeling to a satisfactory degree.

Limitations of the study. In this study, we charted PCK of a small group of informatics teachers. However, because of the variations in their educational background, teacher qualification and teaching experience we expect that our findings are fairly typical for the population of Dutch informatics teachers. This is to be confirmed in further research.

Implications for educational development. We believe that teachers would benefit not only from a course on modeling, but also from the availability of teaching materials. We are convinced that the quality of assessment - an issue attracting a lot of attention in modern informatics education [1] – would improve if teachers get assistance with designing assessment instruments that would take into account both product and process.

In the subsequent phases of this research project we will focus on development of teaching materials and assessment instruments. In parallel, in-service teacher training based on these findings will be offered to interested teachers. Finally, all the participants in this study will be followed to chart the development of their PCK on modeling in the future.

Acknowledgments. This work is supported by the The Netherlands Organisation for Scientific Research grant nr. 023.002.138.

References

1. Alturki, R.A.: Measuring and improving student performance in an introductory programming course. Inform. Educ. Int. J. **15**(2), 183–204 (2016)
2. Barendsen, E., Tolboom, J.: Advisory Report (Intended) Curriculum for Informatics for Upper Secondary Education. SLO, Enschede (2016)
3. Cohen, L., Manion, L., Morrison, K.R.B.: Research Methods in Education, 6th edn. Routledge, New York (2007)
4. Grgurina, N., Tolboom, J.: The first decade of informatics in dutch high schools. Inform. Educ. **7**(1), 55–74 (2008)
5. Grgurina, N.: Computational thinking in dutch secondary education (2013)
6. Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., Stoker, I.: Computational thinking skills in dutch secondary education: exploring pedagogical content knowledge. ACM (2014)
7. Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., Stoker, I.: Computational thinking skills in dutch secondary education: exploring teacher's perspective. ACM (2014)
8. Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., Suhre, C.: Defining and observing modeling and simulation in informatics. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 130–141. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_11
9. Grgurina, N., Barendsen, E., van Veen, K., Suhre, C., Zwaneveld, B.: Exploring students' computational thinking skills in modeling and simulation projects: a pilot study. ACM (2015)
10. Henze, I., van Driel, J.H., Verloop, N.: Development of experienced science teachers' pedagogical content knowledge of models of the Solar System and the Universe. Int. J. Sci. Educ. **30**(10), 1321–1342 (2008)
11. Henze, I., van Driel, J.H., Verloop, N.: Science teachers' knowledge about teaching models and modelling in the context of a new syllabus on public understanding of science. Res. Sci. Educ. **37**(2), 99–122 (2007)
12. Justi, R., Gilbert, J.: Teachers' views on the nature of models. Int. J. Sci. Educ. **25**(11), 1369–1386 (2003)
13. Justi, R.S., Gilbert, J.K.: Science teachers' knowledge about and attitudes towards the use of models and modelling in learning science. Int. J. Sci. Educ. **24**(12), 1273–1292 (2002)
14. Lee, E., Brown, M.N., Luft, J.A., Roehrig, G.H.: Assessing beginning secondary science teachers' PCK: pilot year results. School Sci. Math. **107**(2), 52–60 (2007)
15. Magnusson, S., Krajcik, J., Borko, H.: Nature, sources, and development of pedagogical content knowledge for science teaching. In: Gess-Newsome, J., Lederman, N.G. (eds.) Examining Pedagogical Content Knowledge, pp. 95–132. Kluwer (1999)
16. Rahimi, E., Barendsen, E., Henze, I.: Typifying informatics teachers' PCK of designing digital artefacts in dutch upper secondary education. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 65–77. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_6
17. Saeli, M.: Teaching programming for secondary school: a pedagogical content knowledge base approach. Ph.D., Eindhoven Institute of Technology (2012)
18. Sanders, L.R., Borko, H., David Lockard, J.: Secondary science teachers' knowledge base when teaching science courses in and out of their area of certification. J. Res. Sci. Teach. **30**(7), 723–736 (1993)

A Notation for Sets, Sequences and Series
Possible Benefits for Understanding and Use

Susanna H. du Plessis and Vreda Pieterse(✉)

University of Pretoria, Pretoria, South Africa
vreda.pieterse@up.ac.za

Abstract. The use of a well-designed and uniform notational system is imperative to improve communication and understanding. Currently, a fairly consistent notational system is used to notate the elements of sets, but the notations for sequences, series and other quantifications often vary from one author to another. The most widely used notation for sets has limitations and cannot easily be generalised to sequences and series. Furthermore, the well-known sigma notation for series is ambiguous and difficult to grasp. This article proposes a clear notation that can be used for all of the above. We aim to test the hypothesis that this notation could aid an understanding of concepts which build on the use of this notation. As a first step, we conducted a pilot study to inform the design of an experiment to evaluate the effect of the proposed notation on understanding. Secondary-school learners who had never been exposed to the sigma notation participated in the pilot study.

1 Introduction

Mathematical notation is a language in its own right – unique and constantly evolving. Mathematicians, speaking a range of indigenous languages, use it to express mathematical concepts in a way which everyone can understand. The use of mathematical language has evolved over many centuries as users of this language share a mutual understanding of the meaning of its symbols, words and sentences. Good notation enhances the precision of expression and at the same time simplifies communication.

Mathematical notations are introduced to provide concise and accurate ways of communicating complex yet well-understood concepts. It is well known that brevity is the leading characteristic of mathematical elegance, but this is not the only requirement. Most mathematicians agree on the value of clever notations. Dijkstra and Van Gasteren [1] emphasise that the use of appropriate notation can make a difference in mathematical work. Lipton [2] gives an example of the European mathematicians who used Leibniz's $\frac{dx}{dt}$ differential notation, which enabled them to progress faster than their British counterparts who used Newton's $\dot{x}$ to express the same concept. To support mathematical thinking, notation should not only be designed to enhance the brevity of the text but should also control the number of rules governing the manipulation of expressions [1].

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 77–88, 2017.
https://doi.org/10.1007/978-3-319-71483-7_7

The influence of notational differences on the ease of mastering the underlying concepts, has been investigated in various contexts [3–5]. Chirume [6] found in a study involving secondary schools in Zimbabwe, that a clear notation plays a role at the initial stage of learning a new concept. He found that students might fail to grasp mathematical concepts because they take the symbols themselves as the objects of mathematics rather than the ideas and processes which they represent. These studies confirm that clear and unambiguous notation may promote better cognition and that issues of symbol familiarity and symbol density should be considered when teaching mathematics [7].

Electronic recognition of mathematical expression is a large area of research, one of which is the CROHME project that strives to build a comprehensive database of handwritten mathematical expressions which can be used for the online recognition of mathematical expressions. Other examples of such research are Simistira et al. [8] that propose an algorithm using probabilistic SVMs and stochastic context-free grammars to arrive at a marginal improvement in the recognition of mathematical expressions in order to produce MathML output. More specific research in the use of mathematical expressions online is that of Cuartero-Olivera et al. [9]. They found that a crucial factor in the use of online tools for mathematical expression is the time it takes to enter the expression. They suggest a speech-to-text tool to circumvent this issue.

One of the notations that has been identified as particularly cumbersome, when encountered for the first time, is Euler's sigma notation [10]. This notation is usually introduced for the first time to South African learners at the beginning of Grade 12 when covering the topic of sequences and series. This paper proposes a consistent notational system for sets, sequences and series, which has the potential to increase familiarity and reduce the introduction of additional symbols when teaching these topics. We aim to investigate the value of using the proposed notations: firstly to simplify the electronic formulation of the constructs and secondly, to promote an understanding of the concepts. The participants in the pilot study were secondary-school Grade 11 learners who had not previously been introduced to series or to the use of the sigma notation.

2 Sets

A set is a collection of objects. One way of describing or specifying the members of a set is by extension, i.e. by listing each member of the set. When the members of a set follow a pattern, the set can be specified by intentional definition, i.e. by using a rule or semantic description as specified in the international standard ISO 80000-2:2009 [11]. The general format for the notation is:

$$\{x \mid P(x)\}$$

The symbol $\mid$ means *such that*. Therefore, the above means the set of all elements x *such that* $P(x)$ is a true statement. The property P can be expressed in either words or symbols. The variable on the left of the $\mid$ specifies a dummy whereas the expression on the right delineates the scope. One may use formulas to specify the dummy. For example, the set of odd natural numbers can be expressed as:

$$\{2t + 1 \mid t \in \mathbb{N}\}$$

Dijkstra [12] criticises this notation. He gives the following example which reveals an inherent flaw in this notation:

$$\{i^n \mid i < n\}$$

It can be interpreted as $\{1^n, 2^n, 3^n, \dots, n^n\}$ or as $\{i^{i+1}, i^{i+2}, i^{i+3}, \dots\}$. This ambiguity arises because the specification of the dummy is not separated from the description of the elements. He proposes a notational system to remedy this deficiency. This notation requires the clear separation of three aspects, namely (i) the dummy elements, (ii) the scope description, and (iii) the description of the elements of the set in terms of the dummy elements. Dijkstra's notation for the intentional definition of a set specifies these aspects separated by the: character and enclosed in angle brackets. This notation not only removes ambiguities, it is also a more versatile expression of the conditions for membership of a set. The following is the general format for Dijkstra's notation:

$$\langle\ x : P(x) : f(x)\ \rangle$$

Here x is the dummy, $P(x)$ is a predicate that specifies the scope and $f(x)$ is an expression that describes the elements of the set. More than one dummy, as well as more than one predicate to specify the scope, may be used. When doing so, they should be separated by commas. This allows a distinction between the following:

$$\langle\ i : i < n : i^n\ \rangle, \langle\ n : i < n : i^n\ \rangle \text{ and even } \langle\ i, n : i < n : i^n\ \rangle$$

Dijkstra states that he has no logical objection to declaring the type of the dummy when identifying the dummy. For example, the following are equivalent specifications of the set of even numbers less than 100 using Dijkstra's notation:

$$\langle\ i \in \mathbb{N} : i < 100 : 2 \times i\ \rangle$$
$$\langle\ i : i \in \mathbb{N}, i < 100 : 2 \times i\ \rangle$$

In this paper, we introduce a notation similar to Dijkstra's notation but closer to the ISO standard. The following is the general format for the proposed notation:

$$\{\ x \mid P(x) \mid f(x)\ \}$$

An obvious difference between this notation and Dijkstra's is the use of the punctuation prescribed in the ISO standard, i.e. $\{\ \mid \mid\ \}$ instead of $\langle\ :\ :\ \rangle$. To avoid further ambiguities the following restrictions are also proposed:

- Type specifications of dummy variables are not allowed.
- Multiple scope predicates are not allowed.

If needed, the type of a dummy variable may be specified by including it in the scope predicate. Limiting the specification of the scope to a single predicate is not a real restriction, because if both P_1 and P_2 should hold, it can just as well be specified with the single predicate $P_1 \wedge P_2$ where $\wedge$ is the symbol for the Boolean AND operation. The following is therefore the only legitimate specification of the set of even numbers less than 100, using the proposed notation:

$$\{ \; i \mid (i \in \mathbb{N}) \wedge (i < 100) \mid 2 \times i \; \}$$

3 Sequences

A sequence is an ordered list of objects. A sequence differs from a set because the order of the objects matters. In addition, exactly the same elements can appear multiple times at different positions in the sequence. A sequence with n entries is called an *n-tuple*. As far as we know, no standard has been specified to notate an intentional definition of sequences. In this paper we propose a notation for cases where there is a relationship between i and the value of the i^{th} term in the sequence. This is an adaptation of the proposed notation for the intentional definition of sets in Sect. 2. Similar to the notation for sets, the three aspects — namely the dummy variables, the range predicate and the expression describing the entries — are specified and separated by the | character. This is a variation of the notation proposed by Pieterse [13]. Here we use the punctuation prescribed in the ISO standard for sequences.

To indicate that it is a series and not a set, parentheses are used instead of curly braces, for example, the following specifies the sextuple $(3, 5, 7, 9, 11, 13)$. The formula for the i^{th} term in this sextuple is $2 \times i + 3$. It can therefore be described by using the following intentional definition:

$$(\; i \mid i \in \mathbb{N}_6 \mid 2 \times i + 3 \;)$$

4 Series

A series is the sum of the terms of a sequence. The well-known sigma notation for the summation of sequences was introduced by Leonard Euler in 1755 [14]. Euler's notation and modern refinements of this notation are well established in the mathematical community. When using Euler's notation, $3^2+4^2+5^2+6^2+7^2$ is written as:

$$\sum_{m=3}^{7} m^2$$

Wees [10] contends that students find Euler's sigma notation difficult to understand at first. He attributes this to the complexity of the expression, which is a function that takes as many as four parameters, all of which have to be understood at once. He proposes a programming-like notation as a substitute for Euler's sigma notation. His notation requires descriptive names for the parameters. Though his proposal contributes significantly to the clarity of the expression, it loses two essential attributes of viable notational systems, namely conciseness and independence from natural language.

Dijkstra [12] points out a flaw in Euler's notation which is beyond the problem observed by Wees, namely an inherent semantic ambiguity resulting from uncertainty related to the extent of the description of the elements in the series, for example,

$$\sum_{m=3}^{7} m^2 + 1 \qquad \text{can be interpreted as} \qquad \left(\sum_{m=3}^{7} m^2\right) + 1 \qquad \text{or as} \qquad \sum_{m=3}^{7} (m^2 + 1)$$

This uncertainty can be resolved by introducing parentheses. Dijkstra, however, proposes that the use of an adaptation of his set notation should replace Euler's notation. Dijkstra's proposed notation enforces careful thought which would eliminate possible ambiguities, whereas Euler's notation allows ambiguous expression, placing the onus on the writer to solve possible ambiguities. Dijkstra follows Euler's idea of using the Σ symbol to indicate summation. In Dijkstra's proposal, this symbol is specified along with the dummy variable. The above-mentioned ambiguous specification of a series can therefore only be specified as one of the following unambiguous expressions:

$$\langle\ \Sigma m : 3 \leq m \leq 7 : m^2 + 1\ \rangle \quad \text{or as} \quad \langle\ \Sigma m : 3 \leq m \leq 7 : m^2\ \rangle + 1$$

Besides promoting unambiguity of expression, Dijkstra's notation reduces the need for spatial and size differences to convey meaning in comparison with Euler's notation. Dijkstra's notation has the advantage of clearly distinguishing the description of the terms in the sequence from the surrounding text. The linearity of the Dijkstra notation will make electronic recognition simpler and faster. It does, however, have drawbacks, namely the introduction of an unfamiliar symbol, overloading the purpose of a section in the expression, and applying the operation in the wrong context. We discuss these problems and state how we avoid each of these drawbacks when proposing our own notation.

4.1 Introduction of an Unfamiliar Symbol

Both Euler and Dijkstra use the Σ symbol to indicate summation. This is an entirely new and unfamiliar character for learners who have not been introduced to this topic. We avoid the introduction a new symbol by simply using the $+$ symbol. The learners are already familiar with the symbol and its meaning.

4.2 Overloading the Purpose of a Section in the Expression

Dijkstra overloads the purpose of the first section of the expression when requiring the specification of an operation along with the dummy variable in this section. We suggest that the symbol that specifies the operation should be placed as a prefix operation instead of embedding it in the specification.

4.3 Applying the Operation in the Wrong Context

Dijkstra amended the expression for a set to serve as an expression of a series. Interpreting the series as the summation of a set poses some problems. Consider the following series:

$$\sum_{i=1}^{10}(i \% 2)$$

where % is the symbol for the modulus operation, namely to determine the remainder when one divides the value of the left expression by the value of the right expression. The value of the this series is 5. This is determined by evaluating the following expression:

$$1+0+1+0+1+0+1+0+1+0$$

When representing this series using Dijkstra's notation, one would write

$$\langle\ \Sigma\ i : 1 \leq i \leq 10 : i \ \%\ 2\ \rangle$$

When interpreting the expression – assuming that the summation is over the set which may only have unique values – the expression is equivalent to the summation of the elements of $\{0, 1\}$. The resulting value is 1 instead of the intended 5. This error in interpretation arises from the assumption that the specified terms are elements of a set, instead of the intended meaning, namely to be the terms in a sequence, in which the same element may appear multiple times. To remedy this problem, our proposed notation amends the expression for a sequence to serve as an expression of a series.

4.4 Proposed Notation

Our notation is similar to Dijkstra's notation, yet it has the following differences:

- It uses the + symbol instead of the Σ symbol to denote summation.
- The operation symbol is written as a prefix to the sequence instead of placing it along with the specification of the dummy variable.
- It uses parentheses where Dijkstra's notation uses angle brackets. This notation therefore applies explicitly to series (not sets) and uses the punctuation prescribed for sequences in the ISO standard.

The following is the general format of the proposed notation for series:

$$+(\ x \mid P(x) \mid f(x)\ )$$

The meaning of the parameters is the same as in the notation for sequences: x is the dummy, $P(x)$ is a predicate that specifies the scope and $f(x)$ is an expression that describes the terms in the series. For example, when using our notation, the expression

$$\sum_{m=3}^{7}(m^2+1) \quad \text{is written as} \quad +\ (\ m \mid 3 \leq m \leq 7 \mid m^2+1\ ).$$

The proposed notation is more versatile than other notations. It does not introduce a new symbol and can therefore be used without adaptation for quantifications involving operations other than +. Other notations require the introduction of additional symbols when used for quantifications involving other operations. For example, when using Euler's notation, the symbol Π is introduced to indicate multiplication over a series using, but our notation simply uses $\times$.

Owing to its linear nature and the absence of non-standard keyboard characters the proposed notation may pose fewer problems when used electronically.

5 Pilot Study

We designed a pilot study [15]. The purpose of the pilot study was to gain insight into the use of the notation in the field and to evaluate the feasibility of conducting a full-scale investigation to determine the effect of the use of our proposed notation on the participants' comprehension of the mathematical concepts underlying sequences and series.

The participants in the study were Grade 11 learners who had never been exposed to the topics in question. It is unlikely that these learners would have been exposed to the sigma notation elsewhere, since the target school follows a rigid syllabus and the first introduction of the topic is in Grade 12. We divided the learners randomly in two groups.

We used the same basic material from the Grade 12 mathematics textbook published by Siyavula[1] for both groups. The one group's material was modified to use the Dijkstra notation but used curly brackets instead of angle brackets. The same teacher presented the material to both groups to avoid differences in presentation. We did not introduce more notations because we are not aware of any other notations for these constructs and we did not have enough participants to conduct a meaningful experiment involving more notations.

The experiment was conducted over a period of three days with a work period of approximately three hours every day. The teacher was an experienced lecturer who had taught mathematics in secondary school for more than five years, was involved in teacher training and was familiar with both the notations. The same teaching aids were used in both classes. The lessons involved verbal explanations combined with questions to explain the concepts. Each lesson was concluded with use of worksheets containing exercises which the learners did on an individual basis. The teacher moved from one classroom to the next, presenting the same topic but using a different notation. While one group was receiving a lesson, the other group was doing exercises.

After all the material had been covered, both groups wrote the same test (Test 1). We then inverted the groups, giving the first group a view of the Dijkstra notation and the second group a view of the traditional (Sigma) notation, and again gave the same test to both groups (Test 2), requesting that they use the notation of their choice to write their answers. The final test was intended to ensure that the learners were able to use both notations.

[1] https://www.siyavula.com/maths/grade-12.

At the end of the three days, an opinion poll was held, involving the participants who had remained in the experiment until the end. We asked four questions in this opinion poll:

1. Which notation is easier to write?
2. Which notation is easier to read?
3. Which notation is easier to understand?
4. Which notation do you prefer?

5.1 Learners' Performance

Table 1 shows the descriptive statistics of the marks that the learners achieved in the tests, while the individual performance of each participant is shown in detail in Figs. 1 and 2.

Table 1. Descriptive statistics

	Test 1			Test 2		
	Average	Median	Std Dev	Average	Median	Std Dev
Sigma group	37.00%	37.70%	0.09	20.71%	19.19%	0.09
Dijkstra group	17.21%	13.93%	0.17	13.30%	7.07%	0.11

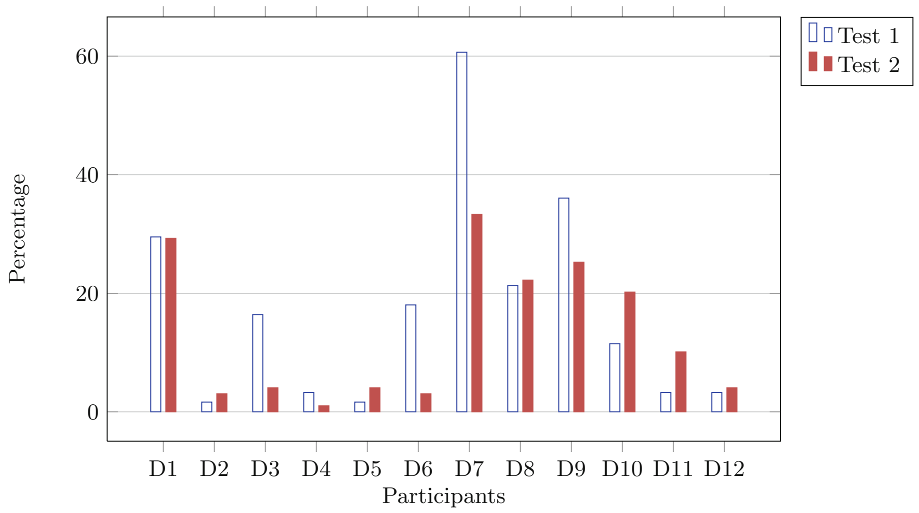


Fig. 1. Dijkstra group test results

The results of the final test were lower than the first, which is unexpected, since one would imagine that learners would perform better after having had

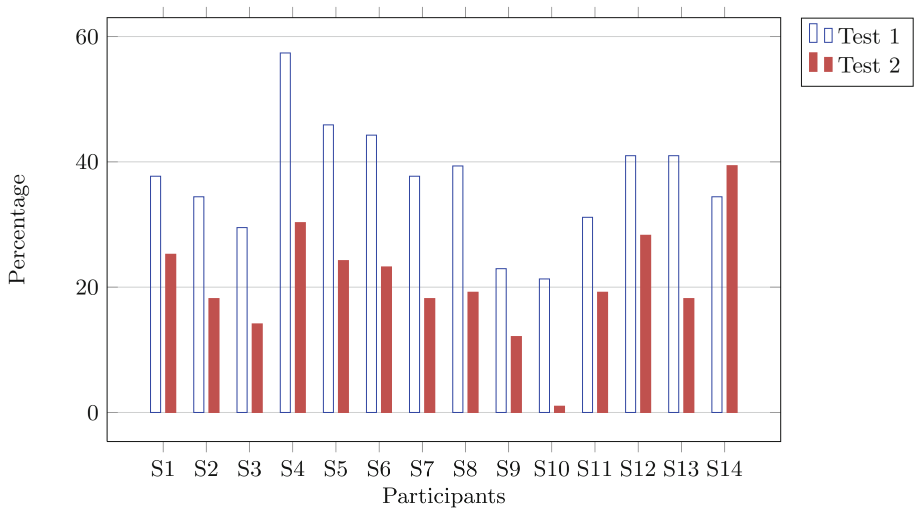


Fig. 2. Sigma group test results

more exposure to the underlying concepts. We can only theorise that the lack of performance was due to the learners having had little incentive to complete the tests, so the learners did not put much effort into the final test.

Statistical analysis is not feasible in this case, due to the small number of participants and the large deviation in the results. The one group was clearly stronger than the other. Evidently our selection method was flawed as it did not render comparable groups.

A factor which might have played a role is that the two groups were not equal in gender distribution. To make matters worse, the dropout rate over the three days skewed the gender distribution even more (Table 2). The group with a majority of girls performed much better on average than the group with predominantly boys. The difference in the performance of the groups is so obvious that it is unlikely to be coincidental. The reason for this difference is, however, unclear. We are inclined to attribute it to gender differences rather than the way in which the material was presented in this experiment. This opinion is a topic for another investigation, which is beyond the scope of the present research.

Table 2. Start and finish number of participants

	Start			Finish		
	Girls	Boys	Total	Girls	Boys	Total
Sigma group	11	5	16	10	4	14
Dijkstra group	5	12	17	4	8	12
Total	16	17	33	14	12	26

Table 3. Opinion poll results

	Sigma notation	Dijkstra notation	No preference	Sigma notation	Dijkstra notation	No preference
	Easier to write			Easier to read		
Sigma group	8	5	1	10	4	0
Dijkstra group	10	1	1	7	4	1
Total	18	6	2	17	8	1
	Easier to understand			Preferred		
Sigma group	9	4	1	10	4	0
Dijkstra group	8	3	1	8	4	0
Total	17	7	2	18	8	0

5.2 Opinion Poll Results

The results of the opinion poll were inconclusive but quite interesting, since the stronger academic group had a higher number of learners who preferred the new notation, although they had been primarily instructed in using the Sigma notation. This indicates that there might be merit in conducting a full-scale experiment to obtain more reliable results.

Table 3 shows the notational preferences of the 26 participants who completed the experiment. The majority of the learners favoured the sigma notation. Regardless of which notation they were taught at first, 18 preferred the Sigma notation and 8 the Dijkstra notation.
The following can be observed:

– 35% of the group instructed by using the Sigma notation felt that the Dijkstra notation was easier to write.
– 66% of the group instructed by using the Dijkstra notation preferred the Sigma notation.

All the tests and exercises were done in a hand-written format. It is clear that the participating learners preferred the sigma notation when they had to use pen and paper. This could be due to the easier visual separation of the elements in this notation but more research would have to be conducted before we could draw conclusions about this preference.

5.3 General Comments

The logistical issues involved in organising a study of this kind are daunting. Permission has to be obtained from governing bodies and teachers, and the parents also have to give their consent. The permissions alone could take several months. Since the school where we conducted our research, is semi-private and dependent on parental funding and goodwill, it is understandable that the schools was reluctant to introduce anything that could be seen as remotely controversial.

Once all the stakeholders had been persuaded and the requisite permissions obtained, a suitable time slot had to be found in the school's extremely packed agenda. For our experiment, this date ended up being after the final exams just before the summer holidays. In the South African school system, learners who have completed their end-of-year examinations do not wait for the official closing date of the schools to go on holiday. This meant that our pool of available learners was small and that the participating learners' motivation to sit through lessons and tests was very low. This could also explain the high dropout rate.

The participating learners had on average a scanty understanding of the material presented and none of the results could be used to draw conclusions about the influence that the use of the different notations had on the ease of comprehension of the underlying concepts.

6 Future Research

In the process of analysing the results of this experiment, several observations were made that warrant further research into the potential benefits of a new notation for the sum of sequences.

We have decided to conduct our full-scale research experiment online. This will enable us to reach a wider audience and to track learner progress more easily. Taking the research online will also allow us to improve some of the aspects of classroom teaching as experienced during this research.

7 Conclusion

Inventing a useful notational system that is economical and aesthetic is an art. A notational system should be equally convenient to write by hand and to typeset with the use of contemporary tools. If it is not promoted in the right place at the right time, it may remain unnoticed and unused. Abadir [16] concedes that it is likely that authors will not adhere to proposed notational standards. He uses the example of Bernoulli [17] who did not adopt the $=$ sign for equality 150 years after it had been proposed, even though many other mathematicians used it.

In this paper, we propose an alternative notation for series to replace the widely used yet cumbersome sigma notation. Our notation is an amendment of a notation designed by Dijkstra [12]. He proposed a set notation to avoid some flaws that are inherent in the standard set notation. He re-appropriated his set notation to specify series. We discuss problems with Dijkstra's notations and propose our own notations for sets, sequences and series, which we believe are more elegant, user friendly and robust than other known notations for these constructs. The notation might enhance understanding of the underlying mathematical concepts.

The results of our pilot study were inconclusive. There could be several reasons for this but we believe it was mainly because we had too little time to convey the topics sufficiently well for the learners to grasp the subject meaningfully.

We are still hopeful that additional research will show that the new notation is beneficial to learning and also more practical in electronic use and e-learning programs.

References

1. Dijkstra, E.W., van Gasteren, A.J.M.: On notation, January 1986. http://www.cs.utexas.edu/users/EWD/ewd09xx/EWD950a.PDF. Accessed 27 Nov 2013
2. Lipton, R.: Notation and thinking, November 2010. http://rjlipton.wordpress.com/2010/11/30/notation-and-thinking/. Accessed 12 May 2016
3. Hoch, M., Dreyfus, T.: Structure sense in high school algebra: the effect of brackets. In: Proceedings of the 28th Conference of the International Group for the Psychology of Mathematics Education, Bergen University College (2004)
4. Kisiel, V.M.: I saw the sign and it opened up my eyes I saw the sign! A study of the impacts of the use of different multiplication symbols in the mathematics classroom. Master's thesis, Department of Mathematical Sciences at the State University of New York at Fredonia, Fredonia, New York, July 2010
5. Torigoe, E., Gladding, G.: Symbols: weapons of math destruction. AIP Conf. Proc. **951**(1), 200–203 (2007)
6. Chirume, S.: How does the use of mathematical symbols influence understanding of mathematical concepts by secondary school students. Int. J. Soc. Sci. Educ. **3**(1) (2012). ISSN: 2223–4934 E and 2227-393X
7. Bardini, C., Pierce, R.: Assumed mathematics knowledge: the challenge of symbols. Int. J. Innov. Sci. Math. Educ. **23**(1), 1–9 (2015). Formerly CAL-laborate International
8. Simistira, F., Katsourosa, V., Carayannis, G.: Recognition of online handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars. Pattern Recogn. Lett. **53**, 85–92 (2014)
9. Cuartero-Olivera, J., Hunter, G., Prez-Navarro, A.: Reading and writing mathematical notation in e-learning environments. ELC Research Paper Series Issue 4 (2012)
10. Wees, D.: Mathematical notation is broken, May 2012. http://davidwees.com/content/mathematical-notation-broken. Accessed 24 Nov 2013
11. ISO/TC 12: ISO 80000–2:2009 Quantities and units – Part 2: Mathematical signs and symbols to be used in the natural sciences and technology, November 2009. http://www.iso.org/iso/catalogue_detail.htm?csnumber=31887. Accessed 22 Dec 2013
12. Dijkstra, E.W.: EWD1300: The notational conventions I adopted, and why. Formal Aspects Comput. **14**, 99–107 (2002)
13. Pieterse, V.: Topic maps for specifying algorithm taxonomies: a case study using transitive closure algorithms. Ph.D. thesis, University of Pretoria (2017)
14. Euler, L.: Foundations of differential calculus. Transl. by Blanton, J.D. Springer, New York (1755/2000)
15. du Plessis, S.H., Pieterse, V.: Die effek vann alternatiewe notasie op die begrip van wiskundige konsepte vir graad 12 leerders. Suid-Afrikaanse Tydskrif vir Natuurwetenskap en Tegnologie **36**(1), 1–2 (2017)
16. Abadir, K., Magnus, J.: Notation in econometrics: a proposal for a standard. Econometrics J. **5**(1), 76–90 (2002)
17. Bernoulli, J.: Ars conjectandi. Impensis Thurnisiorum, fratrum (1713)

Bebras Tasks

Bebras Task Analysis in Category Little Beavers in Slovakia

Lucia Budinská(✉), Karolína Mayerová, and Michaela Veselovská

Department of Informatics Education, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Bratislava, Slovakia
{lucia.budinska,karolina.mayerova,michaela.veselovska}@fmph.uniba.sk

Abstract. In Slovakia, there has been an international Bebras competition (iBobor) since 2007. The first category named Little Beavers (Primary or Bobríci) includes pupils aged between 7 and 10 (2nd - 4th grade of primary school). In this article, we analyse tasks and results from this category collected in years 2012–2017. By qualitative and quantitative research methods, we have created a new categorisation of tasks based on their text analyses in order to better understand some patterns (and correlations) between the tasks and the results. We have created four different categories: Programming oriented tasks, Algorithmic, Logic and Digital Literacy. Using qualitative task analysis, we have also defined several subcategories and, based on analysis of contestants' results, we have found different grade- and gender-to-performance correlation.

Keywords: Bebras competition · Primary education · Categorisation task analysis · Gender performance

1 Introduction

In the school year 2016/2017 we participated in the 10th year of the competition iBobor[1] in Slovakia. It is an international competition named Bebras which originated in Lihuania and it is focused on informatic tasks [1]. In Slovakia iBobor competition has been organized since the school year 2007/2008. We included the category named Little Beaver (at the official website of competition[2] named Primary) into competition iBobor in the school year 2011/2012 for the first time [2]. It was the first year when the fourth grade pupils had already had Elementary Informatics during primary school. This compulsory school subject has been taught since the school year 2009/2010 from the second to the fourth grade for one school period (45 min) per week. The new educational reform introduced in 2014 renamed aforementioned subject to Informatics [3] and placed it in the third and the fourth grade at primary school. In the first year, there were 7,727 competitors in category Little Beavers. Since then, the number of participants has gradually increased, and this year it was 15,486. The category Little Beavers

[1] see http://ibobor.sk/ - in Slovak only.
[2] see http://www.bebras.org/.

is intended for the third and the fourth grade pupils at primary schools. However, teachers may also register younger pupils into this category. Due to the age and cognitive development of the primary school pupils, the organizers have decided to reduce the number of tasks to 12 for this category along with the reduction of time to solve the tasks down to 30 min, while other categories include 15 tasks each with 40 min to find the solution. Tasks in this category typically contain less text information and more pictures. The tasks mentioned are in the form of stories or they focus on real-life situations that should be well known to pupils - so it tries to eliminate abstraction [2].

2 Research Problem

Over the past few years, we have gained a great deal of data that can help us to create a picture of the state of pupils' knowledge. In this article, we focus on the Little Beavers category and the results of pupils in this category over the last 5 years. Tasks in this competition usually come from an international database, yet about 50% of the used tasks in this category have been created by Slovak authors (Fig. 1). The reason may be that not all countries have a category for the primary education in the competition. Within the international database [4], the tasks are assigned to one or more of the following areas:

- Information comprehension,
- Algorithmic thinking,
- Using computer systems,
- Structures, patterns and arrangements,
- Puzzles,
- Social, ethical, cultural, international, and legal issues.

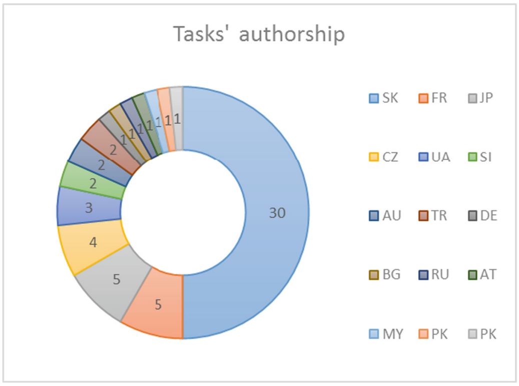


Fig. 1. Authorship of tasks in last five year of iBobor competition in category Little Beavers

However, the tasks used in Slovakia in the category Little Beavers are designed according to the State Educational Program (hereinafter referred to as SEP) [3], which is divided into five thematic areas:

– representations and tools (work with graphics, work with text, work with stories, multimedia work, information, structures);
– communication and collaboration (working with a web site, searching the web, working with communication tools);
– algorithmic problem solving (problem analysis, interactive solution compilation, solution by command sequence, interpretation of solution writing, finding and correcting errors);
– software and hardware (work with files and folders, work in the operating system, computer and add-ons, work on a computer network and on the Internet),
– information society (safety and risks, digital technologies in the company, legality of use).

The tasks categories used in the international database overlap only to a certain extent with the tasks originating from thematic areas of the SEP. Therefore, we are not able to uniquely include the tasks created for competition in the context of informatics in Slovakia. In 2009 Kalaš and Tomcsányiová proposed new categorisation for Bebras task [5] consisting of four categories: *digital literacy*, *programing*, *problem solving* and *data handling*, but categories are broad and overlapping as one task can fit into one or two categories.

In [6] the new two-dimensional categorisation system for tasks in Bebras contest has been introduced. It combines computational thinking skills with informatics concepts. Each task can be placed in only one informatics area, but in up to three computational thinking areas.

One of the objectives of this article was to investigate the success of pupils depending on different phenomena (e.g., conceptual complexity of the task, type of the task, age and gender of the contestant, etc.) and we therefore created our own categorisation to examine these dependencies. The base of it is similar to [5], but we tried to make unambiguous categories and define subcategories whenever possible.

There are not so many researches aimed at Little Beavers category, as many countries don't have this category, but overal we can say, that in most countries boys and girls has similar performance in lower categories [5,7,9]. Boys tend to be more successful in tasks aimed at spatial thinking, creating strategy and in harder tasks, and girls perform better in tasks with colorful pictures and social themes [8]. Contestants are more likely to guess the correct answer than to use no respond answer, and boys do it more often (and even more successful) [7]. In lower categories the proposed difficulty of tasks usually differ from real difficulty, and pupils in this categories tend to underestimate the complexity of the tasks [9]. Pupils in primary schools have problems with reading long texts, they cannot focus on task for a long time, and need to have unambiguous texts and pictures [2,9].

3 Methodology

The aim of our work was to better understand the results of contestants, and also find out correlation between assignments of individual tasks and their results. In our research, we have used qualitative and quantitative research methods. When analyzing the task assignment, we used grounded theory with the systematic design [10]. We coded 60 tasks from the Little Beavers category over the last 5 years, and, based on them, we subsequently created new categorisation. We tried to preserve the data triangulation and the objectivity of the results by individual authors' coding and subsequent joint processing based on inter-coder agreement [11]. The created categories are described in Sect. 4. As in the competition, all tasks are evaluated by points based on difficulty, using three groups: easy, medium and hard, with 4 tasks being in each group. The proposed difficulty does not always correspond to the pupils' results, so for our research, we have redistributed the analysed tasks into these three categories, depending on the real difficulty. (We sorted tasks based on their results, first four with the best average score were labeled as easy, next four as medium and four with the lowest score were hard.)

Next, we used qualitative methods to analyse data from the Slovak competition database, where the following information are kept about each contestant: gender, grade, school's ID, the chosen answer and the time they needed to solve the contest. From this database, we first exported information about pupils competing in the category Little Beavers, which we further processed using inferential statistics for different groups - based on gender, or grade. We have also used correlation designs [10], examining relations and dependencies between the tasks results and the category in which they were placed.

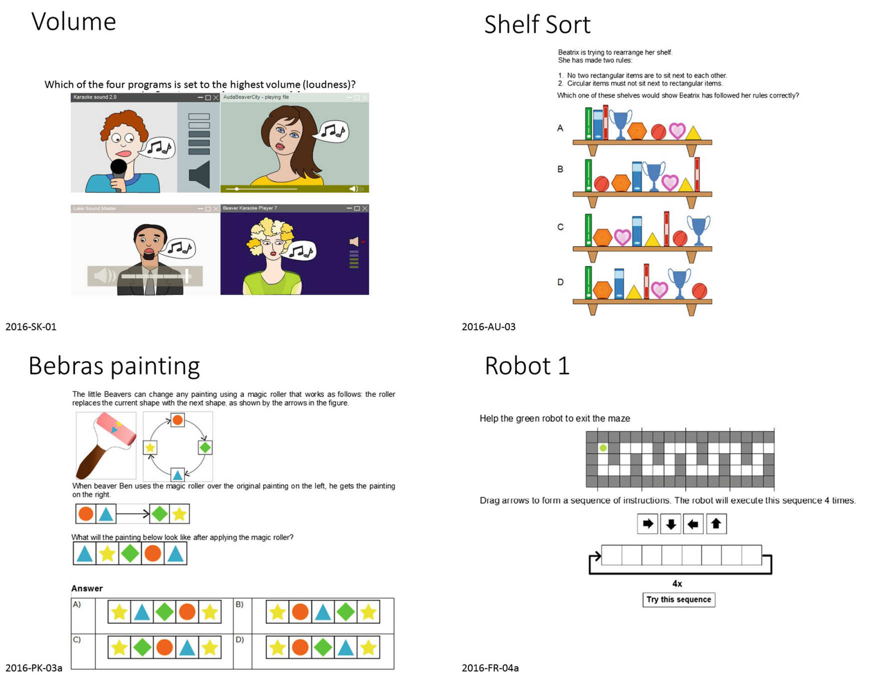
4 New Categorisation

As it was mentioned above, we have created four new categories where all types of tasks used in Slovak competition in category Little Beavers can be included. When creating these categories, we focused on analysing the tasks' text, or more specifically their form and content, and the type of tasks (which area of knowledge is tested). We have further analysed which information is available in text and which is needed to be analysed by pupils, and also how they come up with the answer - if they choose it or they need to create it. Example for each category is shown in the Fig. 2.

4.1 Digital Literacy Tasks

Tasks oriented towards digital literacy are those that focus on verifying the skills in using some (relatively general) software or hardware. We have divided them into two subcategories:

i. **basic** - if the task is focused only on one type of skill or knowledge,
ii. **combined-** if the task contains some more difficult concept or combines basic digital skills with algorithms, pattern recognition or rule identification.



Which of the four programs is set to the highest volume (loudness)?

Beatrix is trying to rearrange her shelf.
She has made two rules:

1. No two rectangular items are to sit next to each other.
2. Circular items must not sit next to rectangular items.

Which one of these shelves would show Beatrix has followed her rules correctly?

The little Beavers can change any painting using a magic roller that works as follows: the roller replaces the current shape with the next shape, as shown by the arrows in the figure.

When beaver Ben uses the magic roller over the original painting on the left, he gets the painting on the right.

What will the painting below look like after applying the magic roller?

Help the green robot to exit the maze

Drag arrows to form a sequence of instructions. The robot will execute this sequence 4 times.

Fig. 2. Examples of tasks for each category from iBobor in school year 2016/17 in Slovakia. *Volume* - digital literacy task. *Shelf Sort* - logic task (statement). *Bebras painting* - algorithmic task (rules are given, answer is created). *Robot 1* - programming oriented task (program creation).

This type of tasks are not usually used in Bebras in other countries and they are not very supported by the international community. However, we have found out that pupils improve between the third and the fourth grade, so we have come to the conclusion that this area of informatics takes a big part in Slovak education. In other countries like for example England, digital literacy is a part of different subjects, not only informatics or computing.

4.2 Logic Tasks

By logic tasks we mean tasks where the steps or algorithm to be followed are not clearly defined. The pupils use exclusion, matching or deduction to solve the tasks. We divided them into three subcategories:

i. **statement** - Statement logic tasks are tasks in which pupils work with logic statement usually given in text, sometimes in a graphical form. Using these statements they deduce the answer.
ii. **graphs** - In the graph tasks, pupils work with the data structure represented by a graph. Typically, it is a tree, a graph (e.g. a square network) or even a linked list or a diagram. More difficult tasks also include some graph algorithm - for example minimal path finding.

iii. **others** - This is a minimal set of tasks that do not belong completely to either of the two previous categories. It can be a mixture of the two, or it can be a task demonstrating a completely different information representation. This category also includes logic puzzles.

4.3 Algorithmic Tasks

These are the tasks in which a pupil needs to follow a procedure or an algorithm or a set of instructions or rules to work with some objects or information to get an answer. We have identified two additional subcategories for these tasks based on the tasks' rules and the way of response.

i. **rules**
   a. *are given* - Pupils have to work with exactly defined conditions, rules, an algorithm or steps to follow and it is obviously stated for them how to solve the task.
   b. *need to be identified* - Pupils are not explicitly told the algorithm or conditions for solving the problem, so they need to identify them by themselves in the text.
ii. **answer**
   a. *is created* - In this categorisation we do not focus on the way the answer is given in the competition (multi-choice or interactive), but on the way how pupils come up with the answer. They can create the answer from a scratch or simulate the whole algorithm, so that the answer is created.
   b. *is identified* - Pupils are already given a pre-prepared answer and they either need to finish it, or they need to identify the right answer which is not needed to be created from a scratch.

4.4 Programming-Oriented Tasks

Tasks which clearly include some execution or a creation of a program or a sequence of steps in a form of text, icons, pictures or combination of the previous are included in this category. Although these tasks might be quite similar to the algorithmic tasks, they specifically need to fall into one of the following subcategories:

i. **creation of a program** - In this category pupils had to create or complete a sequence of steps or a program.
ii. **interpreting the program** - In this category pupils need to interpret or simulate a program or a sequence of steps written in text to find the correct answer.

5 Results

First of all, we analysed data of the last five years of Little Beavers category for grade and gender groups. The competition is more attractive to boys than to girls and the average score of girls and boys is similar, with girls being slightly better in four out of five years (see Fig. 3). Girls achieved better results in tasks where it was not necessary to create or discover an algorithm or use some strategy, but to perform some sequence of steps or commands or to evaluate some states and statements.

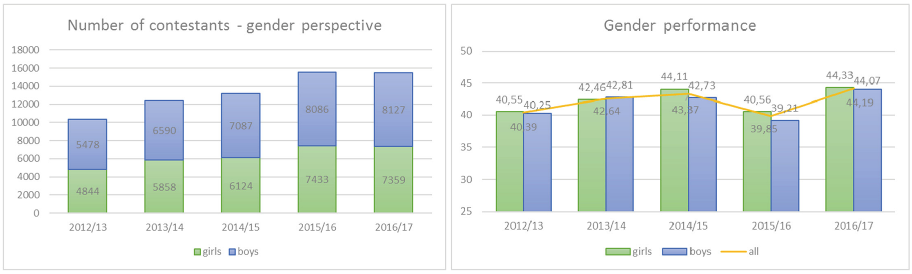


Fig. 3. The last five years of iBobor competition in category Little Beavers - a. number of contestants, b. overall performance for boys and girls (maximum score is 96 points, minimum score is 0 points

There were more fourth graders than third graders in competition and on average 2% of all pupils in category were second graders. Interestingly, the second graders, who entered the contest despite the fact that they were in the category with older pupils, achieved on average very similar results as third graders (see Fig. 4) and they even outperformed third graders and sometimes even fourth graders in some tasks. The reason may be that teachers only enrolled the most clever second graders, but there were often entire classes enrolled in the competition in the third and fourth grades.

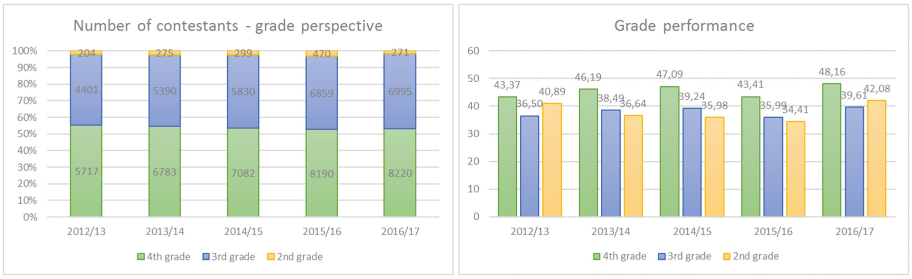


Fig. 4. The last five years of iBobor competition in category Little Beavers - a. number of contestants in each grade, b. overall average performance for each grade (maximum score is 96 points, minimum score is 0 points

Next, based on our new categorisation and data analysis, we were looking for some correlations and dependencies. We took a look at the group of pupils who contested in the school year 2015/16 as third-graders and again in 2016/17 as fourth-graders and compared them to the corresponding grade score average. We found that our group improved mostly in digital literacy based tasks. There could be many reasons why, but it is very likely that this confirms our belief that informatics teachers mostly include digital literacy in their lessons and therefore pupils are most advanced in developing user skills. The same pupils were slightly better than other contestants in the logic and algorithmic task categories and in the programming-oriented tasks they reached approximately the same level.

When comparing the results of all five years, we wanted to discover some correlation that would help us to estimate the difficulty of the task before its application in the competition. However, such correlation and dependencies could be ambiguous, as Vaníček [12] declares, although at least some correlation had been confirmed. These claims are divided into four groups according to the new categorisation, see below. For each group we have created a scatter plot to see if some gender correlation could be found (see Fig. 5).

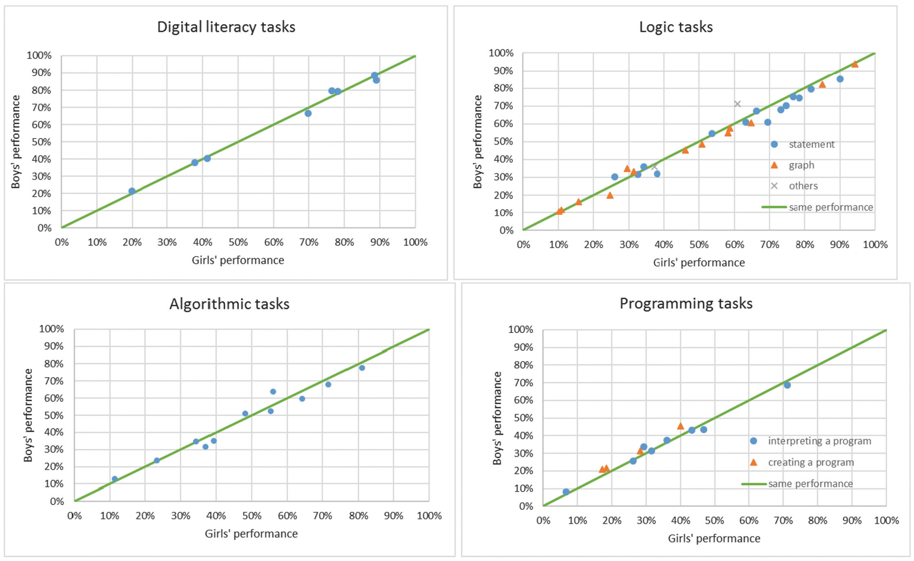


Fig. 5. Correlation of gender performance for each group of tasks. Green line represents the same score for boys and girls. Point below the line means that girls performed better in representing task, point above the line means that boys were better. (Color figure online)

5.1 Digital Literacy Tasks

– The differences between boys and girls in this group were minimal.
– If an algorithmic or logical problem is combined with testing user skills, tasks are more difficult for pupils - they are not easy.

5.2 Logic Tasks

- If the task falls into the logic statement group and task text is in form text-image with clearly defined rules, task is easy for pupils.
- This also applies to image-image tasks that have clear rules but they may not contain another algorithm-complex problem.
- If the task has a small finite number of solutions, it turns out to be easy.
- If the task had more than one correct solution or contains more complicated rules, it is medium difficult. They are mostly in image-image form.
- If pupils need to create answer while the task contains more complicated rules consisting of a number of conditions that need to be identified, the task turns out to be hard.

5.3 Algorithmic Tasks

- If rules are clearly defined in the task, it will be hard or medium difficult for pupils. This may be caused by the fact that authors, seeing that the task contains a more complex algorithmic problem, try not to make it more complicated by identifying rules.
- On the other side, if the pupils had to identify the rules in the task, it usually had a high success rate, i.e. about 70%. In these tasks, pupils usually did not have to create a strategy to solve them.

5.4 Programming-Oriented Tasks

- If task contains a programming concept – such as a variable, a cycle, an obstacle movement or conditions – and program commands are represented by images or pupils need to interpret text commands, task are difficult.

6 Discussion and Conclusion

In this paper we have described new categorisation of the tasks for Little Beavers group in iBobor (Bebras) contest. We have used this categorisation to analyse the last five years of the competition in Slovakia. We have found some correlations and dependencies, mentioned in Sect. 5, although we cannot definitely tell if some causalities really exist. So in the next competition iBobor, in the school year 2017/18, we plan to use these dependencies to categorise tasks in the difficulty groups, and, based on the pupils' results, we would be able to better validate our statements. Also, we would like to use our categorisation on tasks used in other countries in similar age category to compare results of slovak pupils with their peers in other countries.

Pupils' good performance in logic tasks can be influenced by their math lessons or even other subjects, so it would be helpful to do qualitative research aimed at primary school teachers (which in Slovakia use to teach all subjects in their class) and talk with them about tasks in Little Beaver category and possible interdisciplinary relations. So we would find correlation not only with informatics, but get a whole picture.

We were looking at the difficulty of the tasks using recategorisation as follows - first four tasks with the best results were easy, next four medium and the last four (with the lowest score) were hard. In further analysis we would like to recategorise tasks' difficulty based on the percentage results, so the categories could contain more or less than 4 tasks, and then use this recategorisation to find dependencies between our proposed tasks categories and the task's difficulty.

In gender-based analysis, we have found out that girls and boys have overall the same score, but some differencies between the tasks exist. Girls usually perform better in easier tasks, while boys are better in the hardest tasks. Yet to find out in which categories girls or boys are better, we need to create more specific categorisation and analyse the data.

In [12] interactive tasks showed up as easy, but in our data some of them were hard, so in the following research we will try to extend our categorisation to answer type used in the contest with subcategories for interactive tasks. All the proposed improvements could help us better understand what causes the difficulties in the contest and also allow us to create more suitable tasks for Little Beavers category.

Acknowledgement. The research was supported by the project Granty Univerzity Komenského (Comenius University grants) UK/372/2017.

References

1. Dagienė, V.: Information technology contests - introduction to computer science in an attractive way. Inform. Educ. **5**(1), 37–46 (2006)
2. Tomcsányiová, M., Tomcsányi, P.: Little beaver – a new bebras contest category for children aged 8–9. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 201–212. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_18
3. Štátny pedagogický ústav: Informatika. Inovovaný Štátny vzdelávací program (2014). http://www.statpedu.sk/sites/default/files/dokumenty/inovovany-statny-vzdelavaci-program/informatika_nsv_2014.pdf. Accessed 29 May 2017
4. Dagienė, V., Futschek, G.: Bebras international contest on informatics and computer literacy: criteria for good tasks. In: Mittermeir, R.T., Sysło, M.M. (eds.) ISSEP 2008. LNCS, vol. 5090, pp. 19–30. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69924-8_2
5. Kalaš, I., Tomcsányiová, M.: Students' attitude to programming in modern informatics. In: Proceedings 9th WCCE: World Conference on Computers in Education, Paper-No 82 (2009)
6. Dagienė, V., Sentence, S., Stupuriene, G.: Developing a two-dimensional categorization system for educational tasks in informatics. Informatica **28**(1), 23–44 (2017)

7. Dagiene, V., Mannila, L., Poranen, T., Rolandsson, L., Stupuriene, G.: Reasoning on children's cognitive skills in an informatics contest: findings and discoveries from Finland, Lithuania, and Sweden. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 66–77. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_7
8. Hubwieser, P., Hubwieser, E., Graswald, D.: How to attract the girls: gender-specific performance and motivation in the bebras challenge. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 40–52. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_4
9. Dagiene, V., Stupuriene, G.: Bebras - a sustainable community building model for the concept based learning of informatics and computational thinking. Inform. Educ. **15**(1), 25–44 (2016)
10. Creswell, J.W.: Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research, 4th edn. Pearson Education, Boston (2012). 650 p. ISBN 978-0-13-136739-5
11. Silverman, D.: Doing Qualitative Research: A Practical Handbook, 4th edn. SAGE, Newcastle upon Tyne (2013). 633 p. ISBN 978-1-4462-6014-2
12. Vaníček, J.: What makes situational informatics tasks difficult? In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 90–101. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_8

Promoting Computational Thinking Skills: Would You Use this Bebras Task?

Annalisa Calcagni, Violetta Lonati, Dario Malchiodi, Mattia Monga(✉), and Anna Morpurgo

Università degli Studi di Milano, Milan, Italy
mattia.monga@unimi.it
http://aladdin.di.unimi.it

Abstract. Bebras, an international challenge organized on an annual basis in several countries (50 in 2016), has the goal of promoting informatics and computational thinking through attractive tasks. We analyzed Bebras tasks by considering the Computational Thinking (CT) skills each task promotes, starting from the operational definition of CT developed by the International Society for Technology in Education (ISTE) and the ACM-founded Computer Science Teachers Association (CSTA). We argue that such an approach is indeed well-suited to present Bebras tasks, especially with the goal to use them in curricular teaching: framing them as CT enablers helps in making explicit their educational potential, that can be appreciated also by teachers without a formal education in informatics and adapted to a wide range of ages and schools. We explored the viability of our approach by interviewing teachers of different kinds of schools. We propose to use these CT skills also as a tool to classify Bebras tasks, which results in a more uniform distribution of tasks, w.r.t. the one obtained by leveraging content topics.

Keywords: Informatics and education · Computational thinking · Learning contests · Bebras

1 Introduction

The last decade has seen an increasing need for spreading the fundamental concepts of informatics to a vast audience of students, stemming from the belief that some basic concepts of the discipline should be taught even in the first stages of the educational systems. An important contribution to this goal is provided by informatics contests organized worldwide: they are indeed able to stimulate interest among pupils and teachers with different mixes of fun games and safe levels of competitiveness [3]. An initiative that proved to be particularly successful is the Bebras challenge[1] [6,10,11], organized since 2004 on an annual basis in several countries (50 in 2016), with about one and a half million participants in the last edition.

[1] http://bebras.org/.

The Bebras community gathers annually to discuss a pool of new tasks to be proposed to pupils; from this pool the organizers in each country choose the tasks to set up the local contests. The tasks should be fun and attractive, be adequate for the contestants' age and the solution should take on average three minutes per task. Moreover, since the contest is aimed at a non-vocational audience, tasks should be independent of specific curricular activities and avoid the use of jargon. In fact, Bebras tasks focus on that part of informatics that should be familiar to everyone, not just computing professionals. This computing core is sometimes called *computational thinking* (CT) and its promotion is one of the key goals in Bebras, whose full name is indeed "International Challenge on Informatics and Computational Thinking" [8].

In the years since its inception, the Bebras community has developed hundreds of tasks. Most of them are proposed as interactive and/or open-ended questions. However, even when answers have to be chosen from a list, there is no unique way of getting to the solution. The tasks can be used to organize new contests, but they can also be the starting point for in-depth educational activities (a recent proposal is [8]). Potentially, their diversity represents a trove in which every teacher could find suggestions and insights for introducing a computational topic or reflecting on it. Indeed, the Bebras community equips the tasks with comments about their key points ("It's informatics"). While some of the Bebras countries (Switzerland, Lithuania, Singapore) provide additional material to teachers, normally related to the latest edition (booklets or websites with tasks, solutions, and an expanded version of their informatics context), in most cases teachers just discuss the tasks with their students in one session after the contest, and there is little evidence that teachers re-use older tasks and integrate them in their curricular teaching activities (see, for instance [12]).

On the contrary, the Bebras corpus could become a considerable resource to teach informatics and computational thinking, provided that tasks are made easy to retrieve and their content is clearly signposted [8]. This statement is consistent with a survey we conducted recently in our country (see Sect. 2).

After having considered other proposals for the classification of the Bebras corpus (see Sect. 4), we decided to analyze it by means of the operational definition of computational thinking [1] developed by ISTE (International Society for Technology in Education) and CSTA (Computer Science Teachers Association). We first identify the CT skills that are mentioned in the definition and that are relevant when solving a Bebras task; for each of these skills we then give a description that shows which kinds of tasks can promote such skill; finally we analyse several tasks in order to detect the skills they promote. We argue that such approach is indeed well-suited to present Bebras tasks since it helps in making explicit their teaching potential. First, differently from classifications based on a taxonomy of informatics content, this approach is more suitable for identifying the cognitive skills involved in a task. Second, it can be appreciated also by teachers without a formal education in informatics (still the majority, in primary and non-vocational secondary schools): in fact the CT skill set uses terms and concepts which can be grasped even without a deep knowledge of informatics technicalities. Finally, CT skills can be adapted to a wide range of ages and schools, still maintaining their peculiarity, distinct from generic logical/analytical thinking.

We explored the viability of such approach by interviewing individually some teachers of different kinds of schools. Teachers were provided with a short description [4] of the CT skills mentioned above and then were requested to associate some Bebras tasks with those CT skills. More precisely, for each task and each CT skill, we asked teachers the following question *"Would you use this task to promote such skill?"*, and then discussed their answers with them, to understand their motivations. The outcome of such interviews seems to confirm the good potential of the approach (see Sect. 3).

We also propose to use this CT skills as a tool to classify Bebras tasks. We manually classified the pool of tasks prepared for the 2016 contest (120 tasks) and we verified that the resulting distribution among classes is more uniform than that obtained by leveraging content topics.

The paper is organized as follows: in Sect. 2 we describe how the CT skills mentioned in the ISTE/CSTA definition are related to Bebras tasks, and in Sect. 3 we reports the findings of our interviews with teachers. In Sect. 4 we apply this approach to obtain a classification of tasks according to the CT skills they promote, and in Sect. 5 we draw some conclusions.

2 Bebras and CT Skills

After Bebras's last edition (2016), we conducted a survey among the teachers who had registered in the site of the contest in our country: an online questionnaire was filled in by 46% of those participating in the last edition (342 over 742 teachers); for most of them (71%) this was the first participation in Bebras (w.r.t. the previous edition, the number of participant teachers has more than doubled); in general we measured a strong appreciation for the initiative.

Among other questions, we were interested in understanding how teachers make use of tasks (or intend to) in their classes. When asked whether they intend to use Bebras tasks in their classes, 30.9% of the respondents answer affirmatively (in particular 20.3% say that they've already used them); 65% show interest in such a possibility but are uncertain (in particular 20.3% of the respondents choose "I would like to, but I cannot figure out how"); negative answers were marginal (3.6%).

A promising 43.7% of the respondents believe that Bebras tasks can be used for curricular activities (the subjects more frequently mentioned are mathematics, informatics, technology, depending on the grade and kind of school). Moreover, we asked which options would make Bebras tasks easier to re-use for teaching: besides some logistic issues concerning the access and form of online and printed materials, the most appreciated proposals are: to have examples of teaching units that use Bebras tasks (69.2%); to have tasks categorized/arranged according to the computational thinking skills they require and/or promote (51.5%); to have tasks categorized/arranged according to the informatic theme they refer to (46.4%).

Hence teachers are clearly eager to have descriptions of the tasks that could help them in finding the ones more suitable to their goals: both CT and informatics seem useful for them, but while all tasks traditionally have an "It's informatics" section designed to explain the informatic idea behind it, no effort is currently made by Bebras authors to identify and highlight the cognitive skills they require. Thus, we decided to focus on CT skills, which seem to us the core educational value of Bebras tasks, since they keep their computational peculiarity, while being accessible even without a deep knowledge of the more technical details of the discipline of informatics.

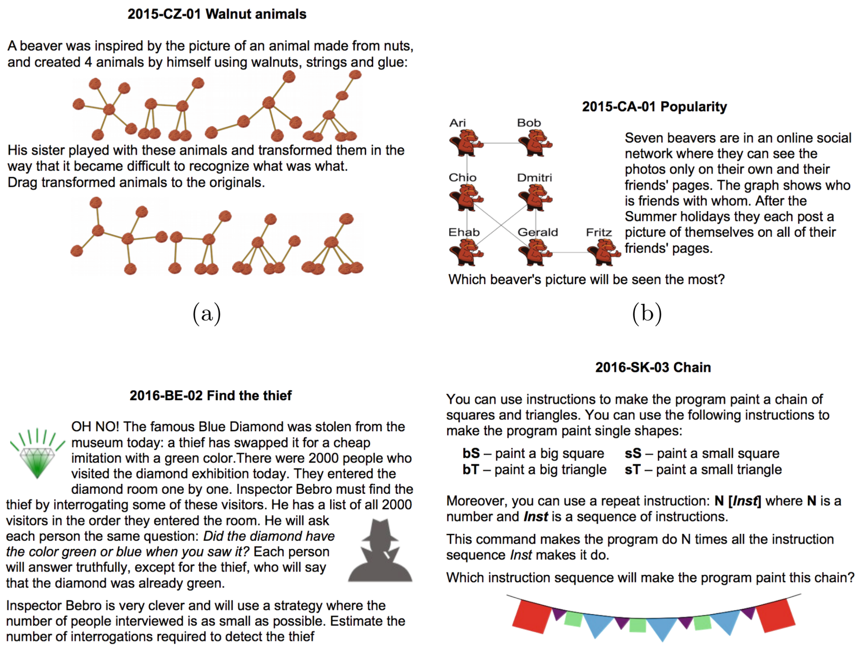
According to the operational definition proposed by ISTE and CSTA [1], CT is a problem-solving process that includes (but is not limited to) the following characteristics.

(a) Formulating problems in a way that enables us to use a computer and other tools to help solve them.
(b) Logically organizing and analyzing data.
(c) Representing data through abstractions such as models and simulations.
(d) Automating solutions through algorithmic thinking (a series of ordered steps).
(e) Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources.
(f) Generalizing and transferring this problem solving process to a wide variety of problems.

The first and last skills in the definition (a and f) are almost never addressed by a single Bebras task, mainly due to its brevity, even though the contest aims at helping the development of such skills in the long-term, by working on and solving several tasks. Instead, most Bebras tasks deal with organization, analysis, representation of data (skills b and c), algorithmic thinking (skill d) or design, analysis and implementation of algorithmic methods (skill e). Thus, we believe such a definition can be fruitful for analyzing and describing tasks.

ISTE/CSTA also propose a vocabulary of CT skills with a progression chart suggesting possible activities for different ages and subjects [2]. The intended goal of the vocabulary is "to unpack the operational definition by listing CT concepts implicit in the operational definition". The vocabulary lists and explains nine terms, giving example activities suitable for the age groups: Data Collection, Data Analysis, Data Representation, Problem Decomposition, Abstraction, Algorithms & Procedures, Automation, Simulation, Parallelization. This level of description, however, seems to enter in the explicit domain of the practice of informatics: as such, it could provide further enrichment for the "It's informatics" section which accompanies each task, but it is less useful to highlight their teaching potential. In general, different (and only partially overlapping) definitions of CT exist (a good recent survey can be found in [5], which discusses also frequent misconceptions of CT by primary teachers), but in this proposal we decided to focus on the operational ISTE/CSTA's definition as the one with the right granularity to emphasize the specificity of computational cognitive skills versus a generic analytical approach, while being accessible to teacher without any formal education in informatics.

In the following we illustrate the CS skills mentioned in the ISTE/CSTA operational definition by referring to Bebras tasks that can promote such skills.



2015-CZ-01 Walnut animals

A beaver was inspired by the picture of an animal made from nuts, and created 4 animals by himself using walnuts, strings and glue:

His sister played with these animals and transformed them in the way that it became difficult to recognize what was what.
Drag transformed animals to the originals.

(a)

2015-CA-01 Popularity

Seven beavers are in an online social network where they can see the photos only on their own and their friends' pages. The graph shows who is friends with whom. After the Summer holidays they each post a picture of themselves on all of their friends' pages.

Which beaver's picture will be seen the most?

(b)

2016-BE-02 Find the thief

OH NO! The famous Blue Diamond was stolen from the museum today: a thief has swapped it for a cheap imitation with a green color.There were 2000 people who visited the diamond exhibition today. They entered the diamond room one by one. Inspector Bebro must find the thief by interrogating some of these visitors. He has a list of all 2000 visitors in the order they entered the room. He will ask each person the same question: *Did the diamond have the color green or blue when you saw it?* Each person will answer truthfully, except for the thief, who will say that the diamond was already green.

Inspector Bebro is very clever and will use a strategy where the number of people interviewed is as small as possible. Estimate the number of interrogations required to detect the thief

(c)

2016-SK-03 Chain

You can use instructions to make the program paint a chain of squares and triangles. You can use the following instructions to make the program paint single shapes:

bS – paint a big square	**sS** – paint a small square
bT – paint a big triangle	**sT** – paint a small triangle

Moreover, you can use a repeat instruction: **N [*Inst*]** where **N** is a number and ***Inst*** is a sequence of instructions.

This command makes the program do N times all the instruction sequence *Inst* makes it do.

Which instruction sequence will make the program paint this chain?

(d)

Fig. 1. Four tasks suitable to teach how to logically analyze data (a), how to represent information (b), how to identify strategies and analyse algorithmic solutions (c), and how to implement algorithmic solutions (d).

2.1 Logically Organizing Data

Typical tasks that promote this skill deal with ways to organize data according to given criteria, with (generally hidden) references to databases or set theory: they may ask to execute a query over a table of records representing a set of objects, to split a set of items into categories according to their characteristics, or to pick the misplaced object in a figure.

Other tasks for this skill focus on organizing data so that they enjoy relevant properties: that is the case for instance of cryptography, where we want a message not to be understandable even if eavesdropped, data compression, where we organize data in order make them easier to store or transmit, or correction codes, where we add bits to detect or recover possible errors in the representation of stored or transmitted data.

This skill is promoted also by tasks where data structures are used to organize data for processing. For instance, a task may concern the use of binary search trees to find quickly any element in a set, without considering the whole set; similar examples can be done for heaps, queues, stacks, and so on. However, if a data structure is used to represent an intrinsic property of the data (e.g., graphs for binary relations, or trees for hierarchical relations), the task will be most suitably related to the skill *Representing information*.

2.2 Logically Analyzing Data

Many tasks for this skill are perceived as "logical problems", since they require logical inference, deductive reasoning, and drawing conclusions about the data presented in the task.

Other tasks ask to check whether the data of the problem satisfy certain properties; often such properties are not straightforward but some reasoning, accurate observations (e.g., recognizing patterns), or a systematic approach are needed to come to the right conclusion. This kind of tasks are well represented by "Walnut animals", proposed by Czech delegates in 2015 and depicted in Fig. 1(a): to solve the task pupils need to abstract from the specific features of animals and consider only their structure, then pairs of isomorphic animals can be matched by analyzing their properties like the number of nuts, or the number of connections from each nut.

2.3 Representing Information

Typical tasks for this skill deal with the digital representation of numbers, images, colors, and sets of objects in general, or the visual representation of data with diagrams like histograms or charts.

Other tasks introduce data structures to represent relevant properties of the data, e.g., graphs for binary relations, or trees for hierarchical relations. For instance, consider "Popularity", proposed by Canadian delegates in 2015 and depicted in Fig. 1(b) where a graph is used to represent and visually show the friendship relation of people in a social network.

2.4 Algorithmic Thinking

Algorithmic thinking deals with transforming an intuitive idea into a form suited to be processed automatically. Hence it allows, for instance: to design a systematic method to tackle a problem, starting from an intuitive approach; to transform an intuitive idea of how to accomplish a task into a step-by-step procedure that achieves the goal; to give a synthetic description of a situation or a process, by detecting and exploiting its patterns and regularities; to start from a set of examples or an informal description and formalize a rule that can be applied in general; and so on.

Thus, typical tasks that promote this skill require to manipulate data following a formal procedure (i.e., a sequence of ordered steps or moves) or a set of rules or primitives. Tasks may require to execute some procedure or to compute or recognize its output; to apply some transition rules to a system in a given configuration; to predict the final state of a process described by a diagram (e.g., the transition diagram of a finite state automaton); to decompose a problem into components; to combine primitive operations in order to compute a result or accomplish a task; to systematically enumerate or examine all the possible cases that can occur in a given context; and so on.

Notice that when the focus of the task is on implementing (analyzing, or devising, respectively) an algorithmic solution, then the task should be most suitably related to skill *Implementing algorithmic solutions* (*Analyzing algorithmic solutions*, or *Identifying strategies*, respectively).

2.5 Identifying Strategies

This skill concerns problem solving and in particular finding a suitable algorithmic strategy. Typically, formulating a solution algorithmically (so that it could be automated) is not sufficient, and tasks that promote this skill usually require to devise a non-trivial idea to address the problems they present.

The tasklet "Find the thief", proposed by Belgium delegates in 2016 and depicted in Fig. 1(c), is a good representative for this task since, in order to estimate the number of interrogations required to detect the thief, students need to understand that examining all visitors sequentially is too time consuming and they need to address the problem with an original approach (in this case, binary search). Thus, the task would be a good choice to promote the ability of devising strategies to solve problems. Notice that, dealing the task also with complexity issues, it could be used to promote the next skill, too.

2.6 Analyzing Algorithmic Solutions

This skill is promoted by tasks concerning global characteristics of the considered algorithm, like correctness or complexity. Thus, tasks for this skill may require to examine an algorithm (or, more generally, a computation method) in order to understand its semantic, predict its overall behavior, determine its invariant properties, estimate how many resources it will consume. Moreover, other typical tasks are those inspired by optimization problems, in that they require to compare and evaluate different approaches in order to find the best one.

An example is given again by "Find the thief" (Fig. 1(c)): if a teacher wants to introduce computational complexity, s/he can start from this task whose solution relies on binary search and requires a complexity analysis in order to estimate the right number of interrogations needed.

2.7 Implementing Algorithmic Solutions

Tasks that promote this skill may be referred to as programming or coding tasks since the focus is on the implementation of algorithms according to a formal syntax defined in the task. If the algorithm to be implemented is not straightforward, then the task will also belong to skill *Algorithmic thinking* or *Identifying strategies.*

Tasklet "Chain", proposed by Slovak delegates in 2016 and depicted in Fig. 1(d) is apt to teach this skill: a simple programming language is provided in the text of the task and a small program has to be implemented to answer the question. Actually, the multiple choice form of the question allows to find the right answer by simply executing four programs; however, from a didactic point of view, the tasks is perfect to introduce programming to young pupils.

3 Key Informant Interviews

To explore the suitability of using CT skills to present Bebras tasks and make them easier to retrieve as a teaching resource, we used the *key informant technique*, *i.e.,* we interviewed eight teachers of different kinds of schools, selecting them for their first-hand knowledge, expertise and reflective practice in teaching. Half of them were quite expert of Bebras tasks, in that they had participated in the Bebras challenge in the past or had contributed in the preparation of tasks themselves; the others instead knew Bebras only a little and are not expert of computing education, but are very much concerned about curricular issues, professional development of teachers, or teaching methodologies; some of them had participated in our teacher training courses in computer science education.

Teachers were first provided with a short description [4] of the CT skills presented in Sect. 2 and were then requested to associate some Bebras tasks with those CT skills. More precisely, for each task and each CT skill, we asked them the following question *"Would you use this task to promote such skill?"*, emphasizing that each task could be associated with more skills. To express their answers, we suggested that they fill out a double entry table (task/skill), where the answers might range between 0 and 3: 0 = 'absolutely not'; 1 = 'more no than yes'; 2 = 'more yes than no'; 3 = 'absolutely yes'.

We then (qualitatively) interviewed them and asked them to discuss their answers with us, in order to understand their motivations. The interviews were loosely structured, mainly relying on three wide issues.

- Are the (descriptions of the) skills clear? Are there terms or expressions that you do not understand or about which you are not sure or that are ambiguous?
- Why do you relate a certain task to a certain skill? (In particular we delved into those cases where the association between a task and a skill was unexpected for us).
- Do you think that if tasks were presented with this approach (that is considering the computational thinking skills they can promote), the educational value of the tasks would emerge more clearly? That is, if presented this way, would it be easier for a teacher to use them in the curricular educational activity, as a curricular resource?

During the interviews, we first noticed that despite the effort of removing (or reducing to the minimum) computer science technicalities, our description of skills still revealed some computer science implied meanings that teachers are not familiar with, and hence needed to be clarified, especially for teachers of primary schools. For instance, when we use terms like *problem* and *solution* we implicitly think of *computational* problems and solutions, while in primary schools a problem is what we would call an *instance of a problem*, and vice versa a computational problem would be seen as a class of similar problems in the primary school meaning. Similarly, for primary school teachers with no formal computer science education, a *solution* is simply an answer to an issue, while when we write "analysing a solution" we are usually thinking of a computational solution for a (computational) problem, *i.e.,* an automatic method to find the correct answer to any instance of the problem. Other expressions that ran into a similar misunderstanding are *representing data* and *organizing data*. Indeed, the digital representation of data as usually meant in computer science is not common knowledge for primary school teachers. When a task deals with some formal/symbolic representation of data, teachers realize that, to understand and tackle the task, one needs to rearrange or reformulate the data somehow and hence they associate the task with the "organizing data" skill. Also the terms *implement* and *coding* were not broadly familiar and needed some explanation. Despite the need to clarify these terms, in general we got confirmation of our hypotheses. We agreed on most associations between tasks and CT skills the teacher highlighted, which seems to confirm that the definitions of skills are understandable and their use to describe tasks is feasible. Teachers seem to appreciate the use of CT skills to analyze Bebras tasks since "make the underlying skills of a task explicit helps in choosing more consciously what to work on and how". The CT skill lens seems to foster the identification of the educational potential of tasks; indeed, often teacher highlighted associations between tasks and skills that we did not expect, but they were usually able to support their association with a convincing reasoning, or with clear examples (for instance envisaging original ways to use the tasks in the classroom, in order to promote a skill that was not directly addressed by the question of the task itself). From the interview we also got some new ideas and insights. We noticed that tasks that required some kind of analytic thought were often associated by teachers with "algorithmic thinking". For instance, solving a task by using the technique of *decomposition* or *step-by-step* reasoning was often associated with algorithmic thinking even when no algorithm or formal procedure or rule was involved. This could be acceptable at the primary school level but, for older kids, algorithmic and analytical thinking should be distinguished more neatly by teachers, in order to appreciate the true computational thinking value of a task, and to promote it in general education.

4 Categorization of Bebras Tasks Using CT Skills

Bebras tasks categorization is also important for the contest itself in that it helps authors span on different topics/skills when they produce tasks and it helps national organizers in putting up a varied contest covering as many informatics topics and computational thinking skills as possible. In fact tasks categorization is an issue in the Bebras community since the beginning. The classification proposed in 2008 [7] turned out to be too coarse to be applied to a variety of tasks; for example Table 1 shows how 120 tasks were classified in the 2016 edition (in fact only 92 of them were indeed classified, since in the other cases the authors of the tasks did not mark them, and we did not complete the classification).

Other categorizations based on the informatics content were proposed first in [12] where four components of informatics education are considered (digital literacy, programming, problem solving, and data handling) and more recently in [13] where a hierarchical classification is suggested based on Schwill's master ideas [14] (algorithmization, structuring, formalization); a classification starting from the Bloom Taxonomy of cognitive skills is presented in [10].

Finally, in [9] a two-dimensional classification is proposed: the first dimension is based on informatics knowledge and proposes five informatics "domains", the second one is based on five CT skills. The five informatics domains, described by means of several technical keywords (*e.g.,* bubble sort, binary tree, *etc.*), were also proposed and tentatively used to classify the 2017 tasks during their creation and discussion, with the vast majority of tasks falling in the first two categories. We also categorized 120 tasks from the 2016 edition within these five domains; the results are given in Table 2. The second dimension, including abstraction, algorithmic thinking, decomposition, evaluation, and generalisation as CT skills, is indeed in the same direction we are proposing here. As such, however, it is very high level, with some categories (especially abstraction) that risk to be too ubiquitous to be useful. We chose not to apply such a classification on our own, in order to avoid misinterpreting the original authors' intention. We instead applied our own scheme to classify the same 120 tasks from the 2016 edition, with the results shown in Table 3, also showing cases in which two (or more) categories were used together, in bold the number of times a category was used alone.

Table 1. Classification of 120 tasks from 2016 edition according to [7]

Topics and concepts	No. tasks
Algorithmic thinking	67
Information comprehension	27
Structures, patterns and arrangements	16
Puzzles	16
Social, ethical, cultural, international, and legal issues	1
Using computer systems	3

Table 2. Classification of 120 tasks from 2016 edition according to [9]

Topics and concepts	No. tasks
Algorithms and programming	87
Data, data structures and representations	37
Computer processes and hardware	15
Communication and networking	3
Interactions, systems and society	2

Table 3. Classification of 120 tasks from 2016 edition: number of tasks and category pairs; bold figures count the occurrences of a category alone

CT skill	No. tasks	OD	AD	RD	AT	ID	AS	IS
Organizing data (OD)	17	**4**	8	2	3	0	0	0
Analyzing data (AD)	31		**11**	8	3	1	0	0
Representing data (RD)	25			**8**	0	0	7	0
Algorithmic thinking (AT)	40				**19**	0	5	9
Identifying strategies (ID)	11					**5**	5	0
Analyzing strategies (AS)	20						**3**	0
Implementing a solution (IS)	15							**6**

5 Conclusions

Bebras tasks are a considerable teaching resource, but unfortunately they are mostly underused beyond the contest.

We analyse and describe Bebras tasks by using the operational definition of computational thinking and identifying seven fundamental CT skills, concerning the organization, analysis, representation of data, algorithmic thinking, and the design, analysis and implementation of algorithmc methods.

Such an approach applies to a wide range of ages and schools and can be described with terms and concepts that do not rely on a strong or wide knowledge of computer science, hence also teachers without a formal education in informatics can understand it and relate it with their curricular teaching activity. Moreover, differently from other classifications based on a taxonomy of informatics topics, this approach also helps in detecting the cognitive skills involved by a task, thus it would make their educational potential more explicit.

We gathered feedbacks about this approach by interviewing teachers and applied it also to classify Bebras tasks. We will also collect feedbacks by building a website where teachers will be able to retrieve tasks according to their potential in promoting CT skills. We are currently working to link such approach to the recommendations of the Italian Ministry of Education for non-vocational schools.

Finally, and overall, we believe the awareness of the importance of a CT-based classification can improve the way tasks will be written in the future. In particular we suggest to expand the Bebras task templates with a new "It's computational thinking" section, containing for each class an articulated answer to the question "Is this task suited to teach this CT skill?".

Acknowledgments. We would like to thank the whole Bebras community for the great effort spent in producing exciting tasklets and the key informant teachers.

References

1. International Society for Technology in Education & Computer Science Teachers Association: Operational definition of computational thinking for K-12 education (2011). http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf
2. International Society for Technology in Education & Computer Science Teachers Association: Operational definition of computational thinking for K-12 education (2017). http://www.csteachers.org/resource/resmgr/472.11CTTeacherResources_2ed.pdf
3. Lonati, V., Monga, M., Morpurgo, A., Torelli, M.: What's the fun in informatics? working to capture children and teachers into the pleasure of computing. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 213–224. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_19
4. Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A.: Bebras as a teaching resource: classifying the tasks corpus using computational thinking skills. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2017, Bologna, Italy, 3–5 July 2017, p. 366. ACM (2017)
5. Corradini, I., Lodi, M., Nardelli, E.: Conceptions and misconceptions about computational thinking among Italian primary school teachers. In: Proceedings of the 2017 ACM International Computing Education Research (ICER 2017), August 2017, to appear
6. Dagienė, V.: Supporting computer science education through competitions. In: Proceedings of the 9th WCCE 2009, Education and Technology for a Better World, Bento Goncalves (2009)
7. Dagienė, V., Futschek, G.: Bebras international contest on informatics and computer literacy: criteria for good tasks. In: Mittermeir, R.T., Sysło, M.M. (eds.) ISSEP 2008. LNCS, vol. 5090, pp. 19–30. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69924-8_2
8. Dagienė, V., Sentance, S.: It's computational thinking! Bebras tasks in the curriculum. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 28–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_3
9. Dagienė, V., Sentance, S., Stupurienė, G.: Developing a two-dimensional categorization system for educational tasks in informatics. Informatica **28**(1), 23–44 (2017)
10. Dagienė, V., Stupuriene, G.: Informatics education based on solving attractive tasks through a contest. In: Key Competencies in Informatics and ICT, KEYCIT 2014, pp. 97–115 (2015)
11. Haberman, B., Cohen, A., Dagienė, V.: The Beaver contest: attracting youngsters to study computing. In: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, pp. 378–378. ACM (2011)
12. Kalos, I., Tomcsanyiova, M.: Students' attitude to programming in modern informatics. Informática na Educação: teoria & prática **12**, 127–135 (2009)
13. Pohl, W., Hein, H.W.: Aspects of quality in the presentation of informatics challenge tasks. In: ISSEP 2015 local proceedings, pp. 21–22 (2015)
14. Schwill, A.: Fundamental ideas of computer science. Bull.-Eur. Assoc. Theor. Comput. Sci. **53**, 274–295 (1994)

Country Reports

Introducing Programming and Digital Competence in Swedish K-9 Education

Fredrik Heintz[1,2,3,4(✉)], Linda Mannila[1,2,3,4], Lars-Åke Nordén[1,2,3,4], Peter Parnes[1,2,3,4], and Björn Regnell[1,2,3,4]

[1] Linköping University, Linköping, Sweden
fredrik.heintz@liu.se
[2] Uppsala University, Uppsala, Sweden
[3] Luleå University of Technology, Luleå, Sweden
[4] Lund University, Lund, Sweden

Abstract. The role of computer science and IT in Swedish schools has varied throughout the years. In fall 2014, the Swedish government gave the National Agency for Education (Skolverket) the task of preparing a proposal for K–9 education on how to better address the competences required in a digitalized society. In June 2016, Skolverket handed over a proposal introducing digital competence and programming as interdisciplinary traits, also providing explicit formulations in subjects such as mathematics (programming, algorithms and problem-solving), technology (controlling physical artifacts) and social sciences (fostering aware and critical citizens in a digital society). In March 2017, the government approved the new curriculum, which needs to be implemented by fall 2018 at the latest. We present the new K–9 curriculum and put it in a historical context. We also describe and analyze the process of developing the revised curriculum, and discuss some initiatives for how to implement the changes.

1 Introduction

In recent years, we have witnessed an active discussion surrounding the role of programming and computer science (CS) for everyone (see e.g., [9,16]). As a result, an increasing number of countries have introduced or are in the process of introducing CS in their school curriculum. For instance, in Europe, the majority of countries (17 out of 21) taking part in a survey conducted by the European Schoolnet in 2015 reported doing so [1]. The way in which this is accomplished varies. Some countries focus on K–12 as a whole, whereas others primarily address either K–9 or grades 10–12. Some countries have introduced CS as a subject of its own (e.g. Computing in England [3]) while others have decided to integrate it with other subjects, by for instance making programming an interdisciplinary element throughout the curriculum (e.g. Finland [6]). A review of models for introducing aspects of CS in K-12 education is presented in, for instance, [7].

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 117–128, 2017.
https://doi.org/10.1007/978-3-319-71483-7_10

The role of CS and information technology (IT) in school curricula has – in general – varied over the years, placing focus on different areas, ranging from using technology as a tool to learning how the computer works and how to use it to create programs. This has also been the case in Sweden. In this paper, we present a historical overview of technology and CS in Swedish K–12 curricula, leading up to a revised curriculum being accepted in March 2017, introducing digital competence and programming as interdisciplinary traits, also providing explicit formulations in subjects such as mathematics (programming, algorithms, and problem-solving), technology (controlling physical artifacts) and social studies (fostering aware and critical citizens in a digital society). We describe and analyze the process of developing the new curriculum, and discuss some initiatives for how to implement the changes.

2 History of IT and CS in Swedish Schools

Already in the early 1970s, the Swedish government gave the agency responsible for school curricula the task of investigating the possibilities of starting to teach computer technology in Swedish schools. In the curriculum from 1980 (Lgr 80), CS (datalära) was introduced in mathematics in grades 7–9, with a clear focus on letting students learn *about* computers. Computers were also mentioned in natural and social studies, with emphasis being on understanding the consequences the rapid development has on both individuals and the society as a whole.

> All students need to be taught about the use of computers in our society and about the fast development in the field. In particular, students should realize that the computer is a technological tool, being controlled by humans. (Lgr 80, p. 107, free translation from Swedish)

National programs were initiated to prepare students for living in a world where computers were thought to play a significant role. The UIDA programme (Utbildning inör datorsamhället, Education for the computer society) aimed at giving all students in grades 7–9 “such knowledge that they can, want and dare to influence the use of computers in the society” [10, p. 104]. In the mid 1980s a specialized school computer called COMPIS (COMPuter in Shool) was introduced. It rather quickly lost ground to the PC due to its limited functionality.

In the 1990s, computers became easier to use, the number of existing software grew and the user interface became more easily approachable. In addition, teachers rarely had sufficient competence for teaching CS, in particular programming. As a consequence, focus was switched from CS towards digital literacy, i.e. how to use applications, tools and computers. Instead of learning *about* computers, the curriculum from 1994 (Lpo 94) focused on helping students learn *with* computers. At the same time, there was a number of initiatives introducing computers in school, for instance the DIG (Datorn i grundskolan, the computer in school) and DOS (Datorn och skolan, the computer and school) projects. Access to computers at homes also rose during the same time as the Home-PC-reform was introduced. This reform made it possible for employees to acquire a computer from their employer against a gross deduction. In the early 2000s, initiatives such as ITiS (IT i Skolan, IT at school) and PIM (Praktisk IT- och Mediekompetens, practical IT and media competence) aimed at helping teachers find good ways of using computers at school. The most recent curriculum reform took place in 2011 (Lgr 11). For a more detailed historical overview of computers and programming in Swedish education see [11].

3 Reintroducing CS in Swedish Basic Education

The Swedish school debate has, in recent years, circled around poor PISA results, difficulties in providing all children and youth with equal opportunities, and about modernizing the curriculum to meet changing job market requirements. As "programmer" is one of the most common jobs in Stockholm, and the need for software professionals is estimated to increase heavily both in Sweden and internationally, some have argued that the educational system should teach programming to prepare young people for these jobs. Others, the authors of this paper included, believe that school should offer all students general preparation for any kind of work, and have therefore argued for digital competence as part of all-round-learning, including computational thinking as a set of general problem-solving skill useful for all in the spirit of Jeannette Wing [17].

In 2012, the Swedish government established the Digitalization Committee (Digitaliseringskommisionen) with the task of providing guidelines for the future of work related to digitalization in Sweden. One of the committee's reports [4] highlights the need for the school system to put larger focus on digital competence. The report explicitly points out the need for including programming in the curriculum as part of existing subjects.

As a result of the discussion around schools, programming and CS as part of all-round learning, persons representing school, universities and industry engaged in voluntary initiatives to help bridge the lack of CS in Swedish basic education. Teacherhack [14] is a nonprofit organization aiming at inspiring "teachers to hack the current curriculum (Lgr 11) to include the essential skills that students need in a digital world". The Teacherhack website provides reviews of all subjects in Lgr 11 with practical advice on how the current texts can be interpreted in order to allow for a more active inclusion of content and practices related to CS, programming and the Internet, as well as to security and integrity issues. Extracurricular activities such as CoderDojos, code camps, after-school clubs and makerspace activities are organized to give children and youth access to informal learning opportunities. Teachers throughout the country are experimenting and sharing experiences from introducing programming in different subjects, ranging from languages to handicraft and music. Heintz et. al. [8] present an overview of ongoing activities related to CS and computational thinking in Sweden, highlighting several projects that show how one can introduce CS already within the current curriculum.

In September 2015, the Swedish government gave the National Agency for Education (Skolverket) the task of presenting a national IT strategy for the

Swedish school system. As one part of this work Skolverket was to update the curricula for primary (K–9) and upper secondary education (grades 10–12). The government explicitly stated that the curriculum should (1) strengthen students' digital competence and (2) introduce programming at the K–9 level.

Vinnova, Sweden's innovation agency, has also funded a number of research projects related to the digitalization of school. One of these, the Trippel Helix Project (www.trippelhelix.se), aims at getting school, industry and academia, with regional backing, to together formulate a common concrete and achievable action plan. The plan should be coordinated with the IT-strategy of the Swedish National Agency for Education, pushing intellectual and operational changes in the school of the future, based on the possibilities and knowledge demands created by the digitalization.

4 Process

The directive from the government to Skolverket called for involving relevant agencies, organizations and groups in the process. Skolverket formed several reference groups, including one in CS education research. The authors of this paper have participated in the process from two different perspectives; as a reference group in CS education research and as part of the Trippel Helix project.

In the first stage, the researchers in the CS education reference group were asked to review the current curriculum for grades 1–9 and suggest changes and additions supporting the introduction of programming and digital competence. The reference group met with Skolverket in December 2015 and presented their input. All researchers suggested that digital competence and programming should be integrated in as many existing subjects as possibly (preferably all of them), and not as a subject of its own. The role of CS was also emphasized, rather than focus on programming which is seen as a particular skill within the broader subject of CS.

Within the Trippel Helix project, representatives from industry, academia and the school system work together collectively on supporting the digitalization of education. During January and February 2016, three workshops were arranged in different parts of Sweden (Stockholm, Gothenburg and Lund) to collect input to the National IT Strategy and the new curriculum, both K–9 and 10–12. The outcome of these workshops was handed as input to Skolverket's ongoing work.

In early March 2016, a first draft including suggested curriculum changes was sent out to different interest groups, with a request for feedback. Many of the additions suggested by the CS education group were included in the draft, but quite a few had also been left out. A few weeks later the first public draft was published on the web. The authors provided feedback on the public draft from a CS education perspective. In late April, the final public draft for a new curriculum was published for anyone to give feedback on, with no explicit request for feedback from the reference groups. In June 2016, Skolverket handed over a proposal for revising the current curriculum to the government.

5 Revised Curriculum for Grades 1–9

In March 2017, the Swedish government accepted Skolverket's proposal [15] introducing a new general section on the importance of digital competence. The new curriculum is mandatory from fall 2018, but schools and teachers are free to start following it earlier if they want to.

Skolverket acknowledges that the meaning of digital competence changes over time due to changes in society, technology and available services [12]. Skolverket's definition is based on the DigComp framework developed by the European commission [5] and the work by Digitaliseringskommissionen [4]. In the Swedish curriculum, digital competence includes four aspects: (1) understanding how the digitalization affects individuals and society, (2) understanding and knowing how to use digital tools and media, (3) critical and responsible usage of digital tools and resources, and (4) being able to solve problems and implement ideas in practice. In a supplemental material [12], Skolverket stresses that the responsibility of helping students develop their digital competence is a responsibility for all subjects.

Programming is considered part of this definition. The supplemental material clarifies that focus is not on coding skills, but on programming as a pedagogical tool and problem-solving process including many phases. Programming should also be seen in a wider context, including "creation, controlling and regulating, simulations and democratic dimensions" [12, p. 10, freely translated]. Skolverket emphasizes the importance of seeing programming in this wider perspective both as a basis for teaching and as part of all four aspects of digital competence.

This provides the general setting for the revised curriculum and the four aspects of digital competence can be found throughout the curriculum, both in general parts and in the course plans for specific subjects. In the following, we outline the most important changes made to the content of three subjects – mathematics, technology and social studies – to address various aspects of digital competence. The outline also shows the progression within different subjects, based on the three curriculum levels (grades 1–3, grades 4–6, and grades 7–9). The Swedish curriculum categorizes content for each subject under a number of headlines in order to make the document more readable.

Programming is introduced mainly as part of the *mathematics* curriculum under the topics "Algebra" and "Problem-solving". The progression is as follows:

- Grades 1–3: How unambiguous step-wise instructions can be constructed, described and followed as a basis for programming. The use of symbols for step-wise instructions.
- Grades 4–6: How algorithms can be created and used for programming. Programming in visual programming environments.
- Grades 7–9: How algorithms can be created and used for programming. Programming in different programming environments. How algorithms can be created, tested and iteratively improved when using programming for mathematical problem-solving.

The subject *technology* was introduced in the Swedish curriculum in 1994, with the goal of helping students develop knowledge and skills needed for orienting and acting in a technology intensive world. The revision puts increased attention on digital technology and the need for developing an understanding for how computers and networks work. The changes are made under three main topics. For the topic "Technical solutions", the revisions focus on understanding aspects:

– Grades 1–3: What computers are used for and some fundamental devices for input, output and storage of information, for instance, keyboard, monitor and hard drive. Some common artifacts that are controlled by computers.
– Grades 4–6: Some of the components and functions of a computer, for instance processor and working memory. How computers are controlled by programs and can be connected through networks.
– Grades 7–9: IT solutions for exchanging information, such as computers, Internet, and mobile phones. Technical solutions that use electronics and how these are programmed.

Under the topic "Development of technical solutions", focus is placed on programming as a means for controlling and creating technology.

– Grades 1–3: Controlling artifacts through programming.
– Grades 4–6: Controlling own constructions or other artifacts through programming.
– Grades 7–9: Controlling and regulating own constructions, for instance using programming. How digital tools can support the development of technical solutions, for instance making drawings and doing simulations.

The related content under the topic "Technology, man, society and the environment" focuses on safety and integrity aspects:

– Grade 1–3: Safety when using technology, for instance when handling electricity and using Internet services.
– Grade 4–6: Safety when using technology, for instance when transmitting information digitally. How technology is part of and changes the basic conditions for different professions within all areas of society.
– Grade 7–9: Internet and other global technical systems, their benefits, risks, and limitations. Safety when using technology, for instance storing and protecting data.

In *social studies*, the changes focus on digital competence, understanding the influence that the digital transformation has on us as both individuals and a society, and helping students become critical and responsible citizens in today's digitalized society. For the early grades (1–3), the previous texts on norms and rules as well as evaluating sources have been expanded to also include digital contexts. For grades 7–9, the main revisions concern two topics, with the progression for the topic "Information and Communication" being the following:

– Grades 4–6: How to act responsibly regarding the use of digital and other media from a social, ethical, and legal perspective.

– Grades 7–9: The valuation of news and how it can be influenced by people's views of the world. How individuals and groups are portrayed, for example based on gender and ethnicity as well as how information in digital media is controlled through hidden programming. The opportunities and risks associated with Internet and digital communication through electronic media as well as how to act responsibly regarding the use of digital and other media from a social, ethical, and legal perspective.

Under "Societal resources and distribution" the progression is as follows:

– Grades 4–6: The importance of digitalization for the individual, for example in the form of improved possibilities for communication and e-commerce.
– Grades 7–9: The importance of digitalization for the development of society in different areas, for example influence on the job market and infrastructure as well as changed attitudes and values.

In addition to the quite notable revisions made in these subjects, other subjects are also affected by the revision. Concrete examples are for instance the introduction of modeling and simulations in natural sciences, extending the range of materials and techniques to be used in crafts and emphasizing responsible communication in languages. In summary, digital competence is visible throughout the new curriculum, both in the general parts and in specific subjects.

6 Implementation

Revising the curriculum is only the first step, the next challenge is to implement these changes in practice. There are more than 200 000 teachers in Sweden and the need for in-service professional development is huge. The important question is: How can teachers throughout the country be supported in teaching programming and digital competence in their subjects?

Although the situation when introducing Computing in the curriculum in England was somewhat different (own subject, ICT teachers already available at schools), many of the challenges are the same. While ICT teachers were proficient in teaching how to use technology, most of them had no prior experience in programming or other topics focusing on understanding the technology. Hence, one can expect to learn from their experiences in Sweden as well.

In England, companies such as Microsoft and BBC have supported the introduction of Computing. The support organization Computing at School (CAS) has helped build a network of teachers, who teach their peers, and a community supporting teachers through local hubs, material distribution, and professional development. It may seem as if teachers in England have received sufficient support. The transition to Computing has, nevertheless, not been unproblematic. After the first year of having Computing, two main challenges were found [13]: teachers' limited programming skills and a lack of self-esteem for teaching the new subject. Even though many of the teachers had taught ICT previously, most of them had no prior experience in programming and CS. The formal professional development offered has not been sufficient and many teachers have felt the need to learn more on their own. Before the first school year of teaching Computing, 60% of all teachers felt they did not know enough to teach the new subject [2]. After the first term, almost half of the students felt teachers needed more training and over 40% of the students said that they had helped their teachers. Over 80% of teachers called for more training, in particular in programming.

In Sweden, Skolverket has developed online course modules to support teachers in learning to teach the new content. These modules include, for instance, school leadership in a digitalized world, programming as an interdisciplinary approach, and programming in mathematics and technology. These modules can be taken at any time, but the recommendation is for colleagues to get together and take the course at the same time, forming collegial learning groups where teachers can learn from each other and share experiences and ideas. In addition, Skolverket arranges conferences throughout the country aimed at showing teachers what the changes mean in practice. Moreover, the supplemental material describes what the new content entails and how it fits in the school context.

The universities are also in the process of slowly renewing the teacher training to meet the new requirements. Currently, the teacher training programs do not include any mandatory courses in digital competence or programming. Skolverket does not control university education, and can hence not place any requirements on them. The universities do however need to educate teachers in the competences and skills required by the current curriculum. Many universities are, for instance, planning courses on different aspects of digital competence, including programming, aimed at both pre- and in-service teachers. One challenge is that the process of introducing new courses often have lead-times of 18–24 m. At least one university (Uppsala) has chosen to compensate for these lead-times and pre-service teachers' lack of training in digital competence and programming by offering complementary education in the form of coding camps offered to students that are graduating within the next 24 m. Other organizations, both public and private, also take an active role in making the transition as smooth as possible. For instance, a school in Stockholm (Årstaskolan) has started a portal of courses, which are freely available to anyone and help teachers learn the basics of programming in different languages in a concrete manner. Swedish national TV produces TV series, showing how to introduce programming to children. Funding is offered for development and research projects, aimed at finding good ways to teach the new content, hence contributing to a set of best practices. Dedicated social media groups gather teachers from all over the country to share ideas and experience as well as answer each other's questions. Regular digichats are organized to provide an informal channel to discuss digitalization, digital competence, and school.

The proposed IT strategy, which is not approved by the government yet, also discusses important issues related to the new curriculum. For instance, the need for updating the IT infrastructure. This includes access both to stable and open Wi-Fi networks and computing devices (each teacher should have her own device within two years and each student should have their own device within three years). The IT strategy also discusses the need for computing devices to be open for teachers and students to test and install their own software. This is very important as in several cases in Sweden, the power of selecting learning material has moved away from teachers and schools to their IT-departments. That is, the format of the tool (from pen and paper to a digital device) dictates who should decide on which learning material to use in the classroom. There are examples where this process has led to computers and networks being totally locked down with committees being the deciding body on which tools to allow.

The IT strategy also requires principals and other school leaders and administrative personnel to have an understanding of the issues involved. Principals play a particularly important role in the implementation process as they must promote and support their teachers in their teaching development efforts.

7 Discussion

In this section, we discuss the revisions from several perspectives including subject integration, bootstrapping, ambition, and equal opportunities.

Own subject or integrated approach. As noted above there seems to be two ways in which countries are introducing CS or programming in their curriculum. England provides a prime example of where the content has been packed into one subject (Computing). In other countries, Sweden included, this has not been the route taken; instead programming and CS content has been integrated in existing subjects. Introducing a subject of its own makes practical questions related to who will teach the content unnecessary – dedicated Computing teachers teach the subject Computing in England. The rationale for choosing an integrated approach is manifold: (1) lack of space for introducing a new subject in the curriculum, (2) letting students see and experience the use of programming in different subjects (e.g. for raising interest among previously underrepresented groups), and (3) introducing computational thinking, providing students a framework for how they can work together with the computer to solve increasingly complex problems. There are also practical considerations such as the negative experience of introducing the subject Technology in 1994 which needed about 20 years to find its place, as well as the need for creating a teacher accreditation for a new subject, which would take years to settle. In addition, as Sweden is only revising the current curriculum, not creating a new one, introducing a totally new subject would most likely not have been feasible.

Bootstrapping problems. The introduction of a new curriculum that spans grade 1 to grade 9 naturally has to take into account a phased ramp up where the progression is continuously adapted to the penetration of the curriculum in terms of number of years of prior knowledge to build on. There is hence a notable challenge in how to bootstrap the introduction of the revised curriculum. Students in all grades (1–9) will need to start with the basics to be able to take on more advanced topics later on. This requires teachers to take the curriculum requirements for lower grade levels into account when designing the progression for their current student group. A particular challenge arises in the intersection between school levels, for instance, when students move from grade 6 to grade 7. At this point many students move to another school, resulting in teachers having a group of students, whose background in the new content can vary greatly depending on which school they have attended and what teachers they have had. The progression in the revised curriculum is expressed as if the curriculum has already reached a steady state. This leaves local authorities throughout the country with the responsibility of implementing the changes and solving the bootstrapping challenge.

Too much or too little. As always when discussing changes and renewal processes, critical voices are heard. Whereas some seem to think that schools should not focus too much on digitalization aspects but pay more attention to basic – and traditional – skills such as reading, writing, and calculating, others think that the revisions are too vague and not bold enough. Developing steering documents for education is difficult, as they should stay relevant for many years. The situation is particularly difficult for a rapidly moving area such as information technology. It is therefore crucial to focus on key principles and ideas instead of detailed instructions and buzzwords. Describing content and learning objectives in general terms is the only way to guarantee that a curriculum will not be outdated after a short period of time. This also holds for the revisions, which are written quite broadly, leaving ample room for interpretation. However, these descriptions are often too vague for teachers and school leaders who are to implement the curriculum in practice. There is hence a need for additional material providing concrete examples of how the curricula requirements can be implemented in the classroom. Such additional material can be updated and expanded on in a fast and agile manner, compared to the curriculum. Experience and knowledge acquired from the large number of active projects and collaboration initiatives between schools, municipalities, industry, and universities will be of great importance when implementing the changes at national level. Material developed and lessons learned abroad can also help avoid reinventing the wheel.

Equal opportunities and broadened participation. In the beginning, the lack of detail in the curriculum can, however, be positive, as this leaves teachers with a more easily approachable task. Instead of having detailed requirements, they can do what they feel is enough and cover more content at more advanced levels as they have learned more and become more experienced themselves. On the other hand, this also introduces the obvious risk of teachers not raising their ambition level as time goes by. Another risk is that all teachers will not see the new content as their responsibility, but rather as part of some other teacher's duties. A challenge for the future is hence to get everyone to do enough – this is a crucial aspect if we are to arrive at a school system providing equal opportunities to all students regardless of where in the country they live and who they happen to have as their teacher. The role of principals and school leaders thus need to be stressed – it is their job to make sure that all teachers are able to provide children and youth with the skills and content listed in the curriculum.

One important, but implicit, goal in the revised curriculum is to broaden the participation in CS related study programs and jobs. This has also been one of the important aspects that the authors of this paper raised in our discussions with Skolverket. This is also one of the reasons why we argued for integrating programming into as many subjects as possible, in particular also in aesthetic subjects such as arts, handicraft, and music. The revisions, however, mainly introduce programming in mathematics and technology, two subjects that can be seen as the traditional and expected choice. We therefore see a risk that the revised curriculum will not broaden participation as much as one would have liked. At the same time, programming is introduced from grade 1 so everyone will get exposed to programming from an early age, which may have a positive influence on participation.

The role of Computer Science. The revised curriculum does not mention computer science or any other synonym to the word. Rather, the focus is on digital competence and programming. We believe this is very unfortunate, as CS is the foundation for programming and includes much more than only programming. This is further exacerbated by introducing programming mainly for mathematical problem-solving and controlling technical artifacts. The international trend, especially in the US, is to broaden the perspective and move the focus from programming to the broader subject of computer science. This is also related to the desire to broaden the participation as discussed above. To engage and interest a broader group, the content needs to be broadened to speak to this larger group. Further, CS is an academic subject, which programming is not. To us introducing programming without CS is like introducing physical experiments without introducing physics. We see upgrading CS to a status of accepted general knowledge, similar to for instance mathematics and physics, as an important goal.

8 Conclusions

Educational systems all over the world are being updated to reflect the possibilities and challenges in the digitalized society. One part of these changes is related to the inclusion of programming and computer science content in basic education. The ways in which this is accomplished vary, and the Swedish revisions are well in line with other countries, where the content is integrated in other subjects.

Making a large-scale change in the education naturally takes time. The situation for students in different municipalities, schools, and even classrooms will vary for many years before all teachers have had time to embrace digital competence and programming concepts as part of their everyday teaching. As the early experience from England indicates, sufficient financial support is not enough; what seems to be most important is access to continuous professional development of high quality, suitable material, and peer support. In addition, school leadership is of crucial importance, as teachers need time, access to professional development, and support to make the best of the new situation.

Professional development is a continuous process, where teachers learn from various sources and from each other. In the field of CS, the principles of life-long learning become increasingly important, as technology moves forward at a fast pace. Although the basic principles and ideas remain the same, new tools are introduced at a regular basis, and teachers need to be able to evaluate and select the right tool for their particular teaching situation.

The authors believe that the ongoing work on introducing programming and digital competence in the curriculum for Swedish basic education is a good start, although much still needs to be done. By presenting the new Swedish curriculum and the practical considerations surrounding its implementation, we hope to add to the discussion on how CS is being introduced in primary education throughout the world. We think that an international discussion on teaching CS in basic education is both important and valuable, both for sharing experiences and learning from each other.

References

1. Balanskat, A., Engelhardt, K.: Computing our future. computer programming and coding. priorities, school curricula and initiatives across Europe (2015)
2. Bateman, K.: Computing teachers need more training, say students, January 2015
3. Department for Education. National curriculum in England: Computing programmes of study (2013). https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study
4. Digitaliseringskommissionen. En digital agenda i människans tjänst : en ljusnande framtid kan bli vår : delbetänkande. Technical report SOU 2014:13 (2014)
5. European Commission. DigComp 2.0: The digital competence framework for citizens (2016). Accessed 22 April 2017
6. Finnish National Board of Education. Perusopetuksen opetussuunnitelman perusteet 2014 (2014)
7. Heintz, F., Mannila, L., Färnqvist, T.: A review of models for introducing computational thinking, computer science and computing in k-12 education. In: 2016 IEEE Frontiers in Education Conference (FIE), pp. 1–9 (2016)
8. Heintz, F., Mannila, L., Nygårds, K., Parnes, P., Regnell, B.: Computing at school in Sweden - experiences from introducing computer science within existing subjects. In: Proceedings ISSEP (2015)
9. Informatics Europe and ACM Europe. Informatics in education: Europe cannot afford to miss the boat, 2015. Report of the joint Informatics Europe and ACM Europe Working Group on Informatics Education
10. Johanssen, M., Nissen, J.: It i framtidens samhälle och i dagens skola. Utbildning och demokrati **10**, 103–132 (2001)
11. Rolandsson, L., Skogh, I.-B.: Programming in school: look back to move forward. Trans. Comput. Educ. **14**(2), 12:1–12:25 (2014)
12. Skolverket. Få syn på digitaliseringen på grundskolnivå, June 2017
13. Swidenbank, R.: Coding in British schools: A review of the first term, January 2015
14. TeacherHack - En digitaliserad Lgr11. http://www.teacherhack.com/
15. Utbildningsdepartementet. Skolfs 2017:11 (2017)
16. White House. Computer science for all (2016). https://www.whitehouse.gov/blog/2016/01/30/computer-science-all
17. Wing, J.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)

Preparedness of Japan's Elementary School Teachers for the Introduction of Computer Programming Education

Yutaro Ohashi(✉)

Nippon Institute of Technology, Saitama, Japan
ohashi@nit.ac.jp

Abstract. By 2020, computer programming will be introduced to every elementary school in Japan. This study explored the extent to which elementary school teachers in Japan are prepared for this. A two-stage research plan was adopted. The initial stage took a qualitative and inductive approach, in which the author visited elementary schools to observe computer classes, interview teachers, and administer a questionnaire. This stage was designed to identify problems and note actual conditions in the field. This knowledge was used to develop the questionnaire for the second survey. A quantitative and deductive approach was used, involving a nationwide survey. Completed questionnaires were returned by 309 elementary school teachers, covering 44 out of 47 of Japan's prefectures. The results show ICT to be currently underused in teaching and revealed that many teachers lack confidence in, or are anxious about, the plan. Four barriers were identified: the low level of ICT use in teaching, teacher skepticism about the plan, the use of conventional teaching methods, and teacher workloads. Overall, teachers were neither technically nor emotionally prepared for the planned introduction of programming education. Ways of addressing each barrier were proposed: increasing access to technological resources, improving teacher training, introducing pupil-centered teaching and learning methods, and reducing teacher workloads.

Keywords: Programming education · Elementary school · Teachers' readiness · ICT in education · Japan

1 Introduction

1.1 Introducing Programming Education into Elementary Schools in Japan

The introduction and integration of Information and Communication Technology (ICT) into school curricula has presented major challenges internationally [16]. At the same time, however, programming has been growing in significance in K-12 education. Its role in the development of childhood skills has been acknowledged, and many countries have incorporated it into their school programs, as programming, computing, coding, or computational thinking [6]. Many model curricula are available [2, 4, 6, 10, 28].

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 129–140, 2017.
https://doi.org/10.1007/978-3-319-71483-7_11

Japan has specified that programming must be introduced into every elementary school by 2020 [15]. In the Japanese context, “Programming Kyouiku” can be translated as “nurturing ‘programming thinking’, which will be required in every job when the children become adults, by letting them experience information processing on computers” [18]. While programming would be a critical part of such a curriculum, the concept goes beyond coding. The specific course content, however, including the programming language to be studied, the hours devoted to the subject, and the grade in which the subject is to be introduced, is left to the individual school and teacher. Programming will not be introduced as an independent subject, but should be taught in combination with existing subjects [15].

Not surprisingly, there has been a debate over the feasibility of this approach. Advocates argue that it is important to keep pace with other countries, where computer programming is already compulsory at the elementary level [17], or that an understanding of coding is essential in a world in which digital equipment forms a large part of daily life [8]. Opponents urge caution, and point to a shortage of teachers who are qualified to teach coding [1]. A more specific concern is that the elementary level is too early [25]. Despite the increase in interest, few studies have attempted to explore the views of elementary school teachers on the planned reforms, or to examine the measures that are being taken. Many case studies have examined advanced schools, and are of little relevance to ordinary in-service teachers, especially given Japan’s relatively low use of computers in learning activities [22, 23, 29]. This low usage has been attributed to shortages of time, content availability, facilities, and resources available to teachers [9].

Many factors are known to play a role in determining ICT practices in the classroom [5, 11, 12, 19, 24]. Key factors include access to resources, the quality of the software and hardware available, the ease of use, the incentives for change, support and collegiality, local and national policies, the commitment to professional learning, and the amount of formal computer training that the teacher has undergone [20]. The beliefs and attitudes of the teacher are also known to play a decisive role in programming education, yet few studies have explored these.

1.2 Research Objectives and Research Questions

This study explored the attitudes towards and awareness of ICT use and programming education by in-service elementary school teachers in Japan. The goal was to identify potential and already existing obstacles and limitations. The following were the research questions:

(1) How is ICT used by teachers in school affairs or teaching?
(2) What do teachers think about programming education?

2 Research Content

A two-stage approach was used. The initial stage was a qualitative and inductive, and was used to gain an understanding of the problems and real conditions in the field, and to establish the scope of the second stage. The author visited elementary schools to observe computer classes, interview teachers, and administer a questionnaire. The second applied a deductive approach, based on a quantitative, nationwide survey and reflecting the hypotheses arising from the first stage fieldwork.

2.1 First Stage: Pilot Study in Four Schools

The author visited four neighborhood elementary schools in Saitama Prefecture to observe computer classes, interview teachers, and administer a questionnaire. The four schools had applied to join our information volunteer program, in which students from our university undertake activities such as supporting computer classes or preparing teaching materials under the concept of service-learning [21]. Visits were arranged to coincide with these activities, and lasted for about half a school day, allowing observations of two or three hours. The observer took notes, asked questions of the students, and interviews the teachers, if possible. Teachers were then asked to complete a paper-based questionnaire, to explore their use of ICT in school and their views on programming education. In some schools, the questionnaire was administered by students from the university.

The questionnaire contained no more than 10 question, excluding demographics (age and years of employment as a teacher. Sex was not explored), and was restricted to one side of an A4 sheet. Questions concerned ICT use in education and attitudes towards programming education.

2.2 Second Survey: Nationwide Questionnaire

This pilot study was used to construct the main survey questionnaire. It was administered nationwide, with the help of a private online research service. Respondents were recruited from monitors registered with the firm. The target response was not less than 300. To eliminate regional differences, sampling attempted to maximize prefectural coverage.

2.3 Analysis Methods

In both surveys, multiple-choice questions were analyzed statistically and average scores and correlations between responses were calculated. Responses to open-ended questions were post-coded using a case-code matrix. This approach to analysis of qualitative data is based on grounded theory [7], and identifies general patterns for a specified case by fully considering both individuality and materiality [26]. Word counting was not used in this study, as it does not always capture the significance of a word and may decontextualize its description. The analytical procedure was as follows.

(1) Codes or keywords were assigned to each description using the following sub-procedures:
    (1-1) Open-ended coding to freely assign a code to a description
    (1-2) Focused coding to assign a more abstract code to a description.
(2) Codes were structured to attain an overall balance.
(3) Steps (2) and (3) were repeated to ensure consistency across all codes.

3 Results

3.1 First Survey: Fieldwork at School

The questionnaire contained 10 questions, and 19 responses were collected from six schools. Questions Q1 to Q8 were multiple choice, and the responses are shown in Table 1. Statistically significance at the one percent level were found for Q1 (ICT use in school affairs), Q4 (confidence in using ICT), Q5 (familiarity with the proposals for programming education), Q7 (experience of computer programming), and Q8 (confidence in teaching computer programming). No significant correlation was found between actual ICT use (Q1, Q2, Q3, and Q4) and confidence in teaching computer programming (Q8). Responses to Q10 were coded into five categories (excluding "No other comments" and other insignificant ones). Many of the comments reflected uncertainty over the coming programming education (Table 2).

Table 1. Responses in the first survey (from Q1 to Q8).

N.	Question	Strongly disagree	Disagree a little	Neutral	Agree a little	Strongly agree
Q1	Do you use ICT in school affairs on a daily basis?*	0	0	1	5	13
Q2	Do you use ICT in teaching on a daily basis?	2	1	4	4	8
Q3	Have you received training to use ICT?	0	5	4	5	5
Q4	Are you confident in using ICT?*	1	5	4	9	0
Q5	Do you know computer programming will be compulsory in elementary schools?*	2	1	1	3	12
Q6	Have you received an explanation of the planned programming education?	3	4	2	3	7
Q7	Have you received training in computer programming?*	10	3	1	3	2
Q8	Are you confident in teaching computer programming?*	11	5	2	1	0

* $p < 0.01$

Responses to Q9 included "support staff and/or help desk (16)", "training and/or instruction course for teachers (16)", "new facilities such as computers and/or tablet PCs (12)", "time (10)", "textbook (9)", and "budget (8)."

Table 2. Responses in the first survey (Q10: Please write your opinions about ICT use or programming education.)

Responses	N.
Need for facilities and/or equipment "If we have easy-to-use tablet PCs and other equipment we can teach pupils in all classes in common (30s teacher)"	3
Anxiety "I don't know about programming at all. I'm anxious about it (40s teacher)"	2
Need for training and/or instruction course "I need to serve apprenticeship for it (40s teacher)"	1
Need for specialized staff "It would be helpful if specialized staff could come over and teach us (50s teacher)"	1
Expectation "It is important for children. *snip* We'd like to explore the way we address it." (50s teacher)	1

Responses to Q10 were coded into five categories, including "need for facilities and/or equipment (3)", "anxiety (2)", "need for training and/or instruction course (1)", "need for specialized staff (1)", and "expectation (1)."

The results of the first survey are summarized as follows.

- Most respondents (18) reported using ICT in school affairs, but fewer in teaching (12).
- Some respondents (15) reported knowing about the planned programming education, but only half (10) had been informed about it.
- Measures concerning programming education had rarely been taken.
- Teachers felt uncertain about or dissatisfied with the planned programming education. They felt under-informed and lacking the facilities and/or equipment needed, or the chance to acquire the necessary knowledge or skills.
- A majority of respondents (16) expressed no confidence in teaching programming.
- Overall, the teachers surveyed were neither technically nor emotionally prepared for programming education.

3.2 Second Survey: Nationwide Questionnaire

The first stage identified certain weaknesses in the questionnaire.

- It did not fully capture the views and attitudes of all teachers, including those who are opposed to or anxious about programming education, and those who put it in practice in advanced cases.
- Some terms were ambiguous. For example, it was unclear whether the option "time" in Q9 referred to teacher time or a time slot in the curriculum.

To address this, the options in the questionnaire were revised. A company specializing in online research checked the consistency of wording and phrases. A nationwide online survey was then conducted on August 3–5, 2017. Completed

Table 3. Revised questions used in the nationwide questionnaire.

N.	Question	Answer
Q1	How many years have you been working as a teacher?	Select from <10 to >40 years, by 10 year intervals
Q2	How often do you use ICT in school affairs?	Six-point evaluation
Q3	How do you use ICT in school affairs?	Multiple choice
Q4	How often do you use ICT in teaching?	Six-point evaluation
Q5	If yes to Q4, how do you use ICT in teaching?	Multiple choice
Q6	Are you confident in using ICT in school affairs and/or teaching?	Five-point evaluation
Q7	Do you know computer programming will become compulsory in elementary school (in Japan)?	Yes/No
Q8	Do you think programming education is necessary in elementary school?	Five-point evaluation
Q9	Have you taught pupils programming at school?	Yes/No
Q10	If yes to Q9, what programming language or development environment did you use?	Multiple choice
Q11	Are you confident in teaching computer programming?	Five-point evaluation
Q12	What is (are) needed to teach programming at school?	Multiple choice
Q13	What do you expect from programming education?	Multiple choice
Q14	Please write your any opinion about ICT use or programming education.	Open ended

questionnaires were returned by 309 respondents from 44 prefectures (out of 47). All were elementary school teachers (Table 3).

- Demographics of the respondents (Q1)
  Forty respondents (12.9%) were in their 20s, 66 (21.4%) in their 30s, 73 (23.6%) in their 40s, 103 (33.3%) in their 50s, and 27 (8.7%) over 60. The average age was 44.9 years. The majority duration of employment was "less than 10 years" (104, 33.7%), followed by "10 to 19 years" (62, 20.1%), "20 to 29 years" (68, 22.0%), "30 to 39 years" (68, 22.0%), and "more than 40 years (7, 2.3%)."
- ICT use in school affairs (Q2 and Q3)
  Respondents used ICT in school affairs "almost every day" (198, 64.1%), "four to five days a week" (38, 12.3%), "two to three days a week" (20, 6.5%), and "a day a week" (10, 3.2%). "Rarely used" was reported by 32 respondents (10.4%) and "never used" by 11 (3.6%). No statistically significant correlation was found between age and years of teaching experience.

  The reasons given for using ICT in school affairs were, in descending order, "preparing for teaching" (252, 84.6%), "document preparation" (e.g. report) (243, 81.5%), "data management" (e.g. pupil scores) (255, 75.5%), "searching for information" (219, 73.5%), "communication via mail and/or intranet" (144, 48.3%), "crafting or making a presentation" (143, 48.0%)", "miscellaneous work for club activity" (27, 9.1%)", and "other" (e.g. preparing a class paper, photo administration, etc.) (8, 2.7%).

- ICT use in teaching (Q4 and Q5)
  ICT use in teaching was less common than in school affairs. Thirty seven respondents used ICT for teaching "almost every day" (12.0%), 31 used it "four to five days a week" (10.0%), 59 used it "two to three days a week" (19.1%), and 56 used it "a day a week" (18.1%)". Meanwhile, 103 respondents reported using ICT "rarely" (33.3%) and 23 answered "never" (7.4%), accounting for more than 40% of respondents. The distribution was statistically significant at the one percent level. No correlation was found with age or years of teaching experience (Q1).

  The most frequently reported use of ICT use in teaching was "teacher shows teaching material" (233, 81.5%), followed by "pupils search for information" (165, 57.7%), "pupils use educational software" (80, 28.0%), "pupils craft a presentation of self-introduction or a survey conducted" (69, 24.1%), "pupils practice typing" (60, 21%), "pupils practice programming" (9, 3.1%), and "others" (5, 1.7%)."
- Confidence in using ICT in school affairs or teaching (Q6)
  When reporting their level of confidence, 40% responded "confident" (22, 7.1%) or "somewhat confident" (104, 33.7%), 92 "neutral" (29.8%), 71 "not very confident" (23.0%), and 20 answered "not at all confident" (6.5%)." The distribution was statistically significant at the one percent level. A weak correlation was found with the frequency of ICT use in school affairs (Q2) ($r = 0.327$) and frequency of use in teaching (Q4) ($r = 0.366$). No correlation was found with age or years of teaching experience (Q1).
- Awareness of programming education (from Q7 to Q13)
  In total 247 respondents (79.9%) were aware that computer programming will become compulsory.

Opinions about programming education varied. Eight respondents rated it "necessary" (2.6%), 113 "somewhat necessary" (36.6%), 84 "neutral" (27.2%), 84 "not very necessary (27.2%), and 20 "not necessary" (6.5%). The distribution was statistically significant at the one percent level. No correlation was found with age or years of teaching experience (Q1).

Forty five respondents (14.6%) reported already having taught programming at school. Popular programming languages were Scratch (18, 40.0%), followed by Programin (13, 28.9%), Viscuit (3, 6.7%), LEGO Mindstorm (2, 4.4%), Google Blockly (2, 4.4%), Logo (2, 4.4%), and "other".

More respondents were unconfident in teaching programming than were unconfident in using ICT, with a majority (65%) lacking confidence in teaching. Four respondents answered "confident" (1.3%)", 28 "somewhat confident" (9.1%), 76 "neutral" (24.6%), 107 "not very confident" (34.6%), and 94 "not at all confident" (30.4%). A correlation was found with confidence in using ICT (Q6) ($r = 0.608$), but not with age or years of teaching experience (Q1).

When asked what was needed to realize programming education, 221 responded "support staff" (71.5%), 202 "textbooks, teaching material, supplemental teaching

Table 4. Revised responses from the nationwide questionnaire.

Categories and examples	N.
Need for fulfilling learning environment "In any event, teachers are short on time. We have more and more tasks (49-year-old male teacher)" "ICT facilities vary depending on the local government. I'm anxious about programming because we have one computer for two pupils. I cannot do what I want to do because computers are too slow and internet filtering is too strict (32-year-old female teacher)"	87
Disbelief "The local government where I work is in extreme poverty and barely in financial reconstruction. Even cracks or leaky roofs cannot be repaired. *snip* I'm in doubt if the ICT-driven classroom can contribute to develop pupils' academic ability. Before programming education, we should be there for pupils (41-year-old male teacher)" "I have no idea in which subject we do programming. If it is left solely in the hands of each teacher or school we don't need it, as we already have other planned subjects such as Ethics and English. The ministry should scrutinize what pupils learn (52-year-old male teacher)"	81
Expectation "I came to think the programming education is required because, for example, foundation of English is nurtured in childhood. I learned electronic engineering with kits in my childhood and it affected me (53-year-old male teacher)" "I think it is important to nurture logical thinking. We need to consider ways of literacy to address various social problems at the national level. (56-year-old male teacher)"	4

material" (65.4%) or "instruction course and/or training for teachers" (65.4%), 162 "time for research and/or practice" (52.4%), 129 "computers" (41.7%), 128 "tablet PCs" (41.4%), 115 "Wireless LAN" (37.2%), 105 "a portal site for programming education" (34.0%), 59 "helpdesk for customer call and/or e-mail" (19.1%), and "others" (9, 2.9%). Sixteen respondents answered "nothing in particular" (5.2%)."

Most respondents (253, 81.8%) had positive expectations of programming education: "knowledge and/or skills needed in the future" (132, 42.7%), "learning with joy" (119, 38.5%), "logical thinking" (102, 33.0%), "creativity" (83, 26.9%), "catalyst to think about future jobs" (80, 25.9%), "experiencing latest technologies" (66, 21.4%), "mathematical thinking" (19.4%), 45 "teachers use computers (60, 14.6%)", "cooperating with other subjects" (40, 12.9%)", and "pupils come to like school and/or studying through programming" (34, 11.0%)." Fifty seven respondents reported expecting nothing (18.4%).

- Opinions about ICT use or programming education (Q14)

After excluding uninterpretable responses (e.g. "No other comments."), 172 comments were collected. Comments were analyzed using the method noted above and divided into three categories: "need for fulfilling learning environment", "disbelief", and "expectation" (Table 4).

4 Discussion

4.1 Revisiting Research Questions

We first discuss the two research questions noted in Sect. 1.2.

(1) How do teachers use ICT in school affairs or teaching?
Respondents have used ICT in school affairs and were reasonably confident in so using it. However, ICT is underused in teaching. Even those who do use ICT in teaching use it in a teacher-centered way (e.g. presentation of teaching material). This confirms previous studies. A report by OECD noted that only 10% of teachers in Japan reported frequent or regular use of ICT for student projects or class work in all or nearly all lessons [22]. An OECD PISA country note stated that computers have less presence in Japanese schools than the average across OECD countries [23]. Cuban (2001) claimed that computers were oversold and underused in US classrooms [3], reflecting the situation in classrooms in Japan.

(2) What do teachers think about programming education?
More than 80% of respondents had positive expectations of programming education and 14% had already taught programming. However, more than half expressed disbelief in or anxiety about the plan. Sixty five percent lacked confidence in teaching programming, twice as many as lacked confidence in using ICT in school affairs or in teaching (29.5%). Supporters of programming education (39.2%) were only slightly more numerous than opponents (33.7%), suggesting that this is a controversial issue even among teachers. In short, the teachers studied were neither technically nor emotionally prepared for programming education.

4.2 Four Barriers and Provisions

Four barriers found from the two surveys, and possible remedial measures, are summarized below.

(1) Low level of ICT use in teaching
The current underuse of ICT in Japanese classrooms is a potential major barrier to its wider implementation. As Kusano *et al*. (2013) suggested, greater access to a technological resource stimulates greater usage [12]. Japanese classrooms should therefore prioritize increasing access. The correlation between confidence in using ICT and confidence in teaching programming suggests that promoting ICT use among teachers may extend its implementation. A realistic approach should reflect our results; computer science unplugged could be a good option for those who cannot use ICT well.

(2) Teachers' lack of confidence in the plan
One notable finding of this study was the number of Japanese teachers who lack confidence in or are anxious about the plan. Previous studies had not identified this. The major reasons are that the teachers are poorly informed, or think the plan fails to reflect actual conditions. The decision to use ICT in the classroom is affected by the teacher's commitment to professional learning and background in formal computer training [20]. Proper teacher training is therefore essential. When

designing a training plan, known barriers such as isolation, lack of adequate computer science background, and limited resources for professional development [30] should be taken into account. However, training more than four hundred thousand in-service elementary school teachers [14] within the available time will be a considerable challenge.

(3) Conventional teaching methods
In the course of the fieldwork, conventional teaching methods were observed in the classroom. Some were teacher-centered even when computers were used, for example all pupils doing the same exercise, or merely copying the teacher. Paper-based teaching and learning styles were also encountered, even in computer classes, with an emphasis on writing by hand. This may reflect Japanese writing culture, which encourages many (and especially teachers) to believe that good writing must be stressed in childhood.

While such conventional methods have a role in the classroom, they may also construct an invisible barrier to ICT integration, which requires self-motivated learning or learning by doing. The current discussion is mainly focused on coding. An equal focus should be placed on the teaching and learning methods that are best suited to programming education.

(4) Overworked teachers
The fourth barrier is the lack of teacher free time, which was found in the observational fieldwork rather than the survey. Teachers were working busily and were heard to complain about overwork on several occasions. One female teacher observed that she was unable to handle her current workload, and that the addition of programming education would be impossible. This is supported by the fact that Japanese teachers work the longest hours in the OECD member countries, due to the extra load of administrative work and extracurricular activities (e.g. coaching clubs) [27]. While measures are being taken to tackle this [13], overtime work is a custom that remains rooted in Japan's working culture. Ways must be found to lighten teacher workloads. Our fieldwork observations and survey results suggest that this will be a significant barrier to the successful introduction of programming education.

5 Conclusions

This paper explored the extent to which elementary school teachers in Japan are prepared for the planned introduction of programming education. Field observations and a nationwide survey were conducted. Completed questionnaires were returned by 309 elementary school teachers, covering 44 out of 47 prefectures. The results show that ICT is currently underused in teaching and that many teachers lack confidence in, or are anxious about, the plan. Four barriers were identified: the low level of ICT use in teaching, teacher skepticism about the plan, the use of conventional teaching methods, and teacher workloads. Overall, teachers were neither technically nor emotionally prepared for the planned introduction of programming education. Ways of addressing each barrier were proposed: increasing access to technological resources, improving teacher training, introducing pupil-centered teaching and learning methods, and reducing teacher workloads.

Acknowledgments. Author thanks all of the concerned parties, i.e., teachers and staff at the participating schools, participating students, supporters of this study, and those who proofread our work.

References

1. AERA: Few teachers can teach computer programming even though it will be compulsory in elementary school from 2020 (in Japanese). https://dot.asahi.com/aera/2016102500219.html. Accessed 10 June 2017
2. Computing At School (CAS): Computing in the national curriculum: a guide for primary teachers (2012). http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf. Accessed 10 June 2017
3. Cuban, L.: Oversold and Underused: Computers in the Classroom. Harvard University Press, Cambridge (2001)
4. Du, J., Wimmer, H., Rada, R.: “Hour of Code”: can it change students’ attitudes toward programming? J. Inf. Technol. Educ. Innov. Pract. **15**, 52–73 (2016)
5. Ertmer, P.A., Ottenbreit-Leftwich, A.T., Sadik, O., Sendurur, E., Sendurur, P.: Teacher beliefs and technology integration practice: a critical relationship. Comput. Educ. **59**, 423–435 (2012)
6. European Schoolnet: Computing our future: Computer programming and coding Priorities, school curricula and initiatives across Europe (2015). http://www.eun.org/publications/detail?publicationID=661. Accessed 10 June 2017
7. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Transaction Publishers, Piscataway (1967)
8. Harada, Y.: Peta-gogy for future: programming for children with viscuit. Inform. Process. **53**, 60–64 (2011). (in Japanese)
9. Japan Association for Promotion of Educational Technology (JAPET): Actual condition survey on ICT use in school (in Japanese). http://www2.japet.or.jp/ict-chosa/ict_chosa_data.pdf. Accessed 10 June 2017
10. k12.org. https://k12cs.org/. Accessed 10 June 2017
11. Kim, C., Kim, M.K., Lee, C., Spector, J.M., DeMeester, K.: Teacher beliefs and technology integration. Teaching Teacher Educ. **29**, 76–85 (2013)
12. Kusano, K., Frederiksen, S., Jones, L., Kobayashi, M., Mukoyama, Y., Yamagishi, T., Sadaki, K., Ishizuka, H.: The effects of ICT environment on teachers’ attitudes and technology integration in Japan and the U.S. J. Inf. Technol. Educ. Innov. Pract. **12**, 29–41 (2013)
13. Ministry of Education, Culture, Sports, Science and Technology – Japan (MEXT): Making school working environment proper (2016) (in Japanese). http://www.mext.go.jp/a_menu/shotou/uneishien/detail/__icsFiles/afieldfile/2016/06/13/1372315_03_1.pdf. Accessed 12 Sept 2017
14. Ministry of Education, Culture, Sports, Science and Technology – Japan (MEXT): MEXT Statistical Directory (2016) (in Japanese). http://www.mext.go.jp/b_menu/toukei/002/002b/1368900.htm. Accessed 12 Sept 2017
15. Ministry of Education, Culture, Sports, Science and Technology – Japan (MEXT): A concept of programming education at elementary school level (2016) (in Japanese). http://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm. Accessed 12 Sept 2017

16. Ministry of Education, Culture, Sports, Science and Technology – Japan (MEXT). Informatization of education. http://www.mext.go.jp/b_menu/hakusho/html/hpac200101/hpac200101_2_067.html. Accessed 25 August 2017
17. Ministry of Education, Culture, Sports, Science and Technology – Japan (MEXT): Report on programming education in other countries (in Japanese). http://jouhouka.mext.go.jp/school/pdf/programming_syogaikoku_houkokusyo.pdf. Accessed 10 June 2017
18. Ministry of Education, Culture, Sports, Science and Technology – Japan (MEXT): The concept of programming education in elementary school (in Japanese). http://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm. Accessed 10 June 2017
19. Msila, V.: Teacher readiness and information and communications technology (ICT) use in classrooms: a South African case study. Creative Educ. **6**, 1973–1981 (2015)
20. Mumtaz, S.: Factors affecting teachers' use of information and communications technology: a review of the literature. J. Inf. Technol. Teacher Educ. **9**(3), 319–341 (2000)
21. Nejmeh, B.A.: Service-Learning in the Computer and Information Sciences: Practical Applications in Engineering Education. Wiley, Hoboken (2012)
22. OECD: Result from TALIS (2013). https://www.oecd.org/japan/TALIS-2013-country-note-Japan.pdf. Accessed 10 June 2017
23. OECD: PISA Students, Computers and Learning: Making the Connection, Country note Japan. https://www.oecd.org/pisa/keyfindings/PISA-2012-students-computers-japan.pdf. Accessed 10 June 2017
24. Prestridge, S.: The beliefs behind the teacher that influences their ICT practices. Comput. Educ. **58**(1), 449–458 (2012)
25. Sankei Shimbun. http://www.sankei.com/column/news/160801/clm1608010006-n1.html. Accessed 10 June 2017
26. Sato, I.: Qualitative Data Analysis Methods, Shinyosha (2011). (in Japanese)
27. The Japan Times: Japanese teachers work longest hours among OECD members. http://www.japantimes.co.jp/news/2014/06/26/national/japanese-teachers-work-longest-hours-among-oecd-members/#.WTuDBJLyiCg. Accessed 10 June 2017
28. Tucker, A. (ed.): A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee (2003). https://www.acm.org/education/curric_vols/k12final1022.pdf. Accessed 10 June 2017
29. UNESCO: Information and Communication Technology (ICT) in Education in Asia: a comparative analysis of ICT integration and e-readiness in schools across Asia. http://www.uis.unesco.org/Communication/Documents/ICT-asia-en.pdf. Accessed 10 June 2017
30. Yadav, A., Gretter, S., Hambrusch, S., Sands, P.: Expanding computer science education in schools: understanding teacher experiences and challenges. Comput. Sci. Educ. **26**(4), 235–354 (2016)

Baltic and Nordic K-12 Teacher Perspectives on Computational Thinking and Computing

Arnold Pears[1(✉)], Valentina Dagiene[2], and Egle Jasute[2]

[1] Royal Institute of Technology (KTH), Stockholm, Sweden
pears@kth.se

[2] Vilnius University Institute of Mathematics and Informatics, Akademijos Street 4, LT-08663 Vilnius, Lithuania
{valentina.dagiene,egle.jasute}@mii.vu.lt

Abstract. This paper reports on the results of a study of teacher preparedness and practices in relation to teaching computing and computational thinking at schools in Sweden, Finland and Lithuania. The study was conducted as part of a NordForsk funded project to explore how Computing Education Research in the Universities can help the development of teacher training and K-12 curriculum and teaching practices. The study found that many teachers are already engaged in teaching relevant material in the schools, and that many have good support in their local school environment. However, there are also significant challenges which emerge from the new curricula that have been introduced in Sweden and Finland. To meet these challenges new teacher training programmes will be needed, and we recommend that computational thinking and computing concepts be introduced into the core subject content of teacher education programmes in order to better prepare teachers to meet the educational demands of our increasingly digitalised society.

Keywords: Computational thinking · Teacher training · School curriculum

1 Introduction

This paper draws on data collected over a two year period as a part of a two year transverse action, the NordPlus Network on Innovative Computing Education - NordNICE, funded by NordPLUS. The goal of the action was to increase the interaction and knowledge flow between the compulsory schooling sector and university research centers to enhance mutual understanding of how best to equip future generations with the computational thinking and computing knowledge which underpin the success of our increasingly technological society.

The network activities were designed to facilitate extensive collaboration between educators, teachers, and researchers in computing teacher education, both in pre-service and in-service training. Coordinated by Vilnius University, the network was a collaboration between several Nordic and Baltic universities - Uppsala University, Åbo Akademi University, University of Tartu, University

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 141–152, 2017.
https://doi.org/10.1007/978-3-319-71483-7_12

of Eastern Finland, Aalto University, KTH University, and Latvian University. These universities shared expertise and collaboratively developed materials for teaching computing together with partners in the schools of the participating countries. The initiative introduced new methods of teaching programming, using visualization tools in teaching, and using modern and innovative teaching methods in teaching computer science to school teachers. The work was conducted as a mix of live and online meetings to establish and develop the Network. A programme of activities was carried out to develop teacher training guidelines and a framework for computing curricula for schools. The initiative developed mechanisms for sharing best practice, inclusion of new partners, and dissemination of results. The ultimate goal is to improve the quality of computing teacher education in all Nordic and Baltic countries, and in this way support the development of top-level experts for the ICT industry and public organizations.

Data from the NordNICE action included responses to a survey designed to benchmark teacher perceptions of the relevance of computational thinking and their associations with computational thinking and computing in general. While the term Computational Thinking (CT) has been ascribed to Papert [1], many of the fundamental ideas have existed for a long time [2]. The recent wave of change in school education was sparked by the use of the term by Jeannette Wing in her 2006 article [3] in which she suggested that computational thinking was *"... a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability."* [3] She followed this up with a more concrete definition in 2011 where she formulated CT as *"the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be carried out by an information-processing agent"* [4]. Crucial to this latter definition is the idea that there are layers of abstraction of both data and processes involved in computational thinking.

However, as has been demonstrated by the range of initiatives in the Baltic region and internationally this is not a trivial process. Developing curricula that integrate CT components into a broad range of school subjects is a long term process requiring considerable systematic work. New curricula have been devised and deployed [5–9], but, how well do teachers feel prepared to teach these new curricula? One of the reported weaknesses of prior initiatives which have introduced CT and computing content into the compulsory school curriculum is the lack of attention to supporting teachers [10]. This general challenge can be decomposed into several sub-components which need to be addressed. The most critical to short term success is provisioning of in-service training for teachers who are expected to implement new content in the contexts of their school teaching. In the longer term there is also a need to revise the curricula and courses in teacher education programs at universities to include new material and to discuss how to deal with the specific pedagogical challenges that the new curricula content presents for pupils.

A major area of need is primary school teacher training, since these teachers do not traditionally have any background in computing. To help inform the development of new in-service teacher training initiatives the NordNICE partners surveyed school teachers in 2015 and 2017 to gather perceptions of how development of their competence in these areas was supported by their local environment.

2 Status of Computing in School Curriculum

2.1 Lithuania

In Lithuania Computing (and CS) is referred to as Informatics ("informatika"). As early as 1978–1979, proposals were developed for pupils' education in programming. In 1986–1997 Lithuania established compulsory informatics content in all secondary schools. The course was lectured for two years with 70 h overall in grades 11 and 12 (leaving certificate). The curriculum covered three main areas: (1) the concept of information, principles of its transmission, storage and processing; (2) logical principles of computers; (3) algorithms, particularly data types and basic control structures of programming.

As a part of the education reform in 1997, the Informatics core curriculum went through a major revision and it was expanded from two years to four years teaching time (in total 136 h) with more focus on application and developing algorithmic thinking. The teachers were formally qualified, usually with a bachelor or masters degree in informatics combined with mathematics. Teacher training institutions offered various courses in programming and applications.

Since 2005, the main part of the compulsory informatics course has been dedicated to computer literacy and applications. As a result, the teaching of informatics has become quite poor. Pupils are introduced to the basics of informatics in grades 5 and 6 based on Logo or Scratch [11]. For grades 9 and 10, an optional module on algorithm design and coding is recommended for advanced students. Consequently lower secondary school introduces basic informatics and serves as a starting point for the informatics education of all students in high school.

The most significant influence to deepen informatics education are shaped by the development of the informatics maturity exam in 1995. Those, who pass the informatics exam, have enhanced opportunities to enter CS-related studies in higher education. The test provides a reliable indication as to whether a student has the aptitude to study informatics. The informatics exam consists of two parts: one part (over 50%) is allocated to programming, while the rest concerns the issues of computer literacy and application. The exam focuses on: knowledge and understanding 30%, skills 30% and problem solving 40%. Tasks are oriented towards knowledge of data structures and the development of simple algorithms.

In 2016 the Lithuanian Ministry of Education and Science developed new guidelines (framework) of new informatics curricula for grades 1–10 with special focus on informatics concepts and skills [5]. Students should gain familiarity with basic concepts in informatics in grades 5 and 6 when they learn coding with Logo and Scratch, etc. In grades 9 and 10 when they progress towards developing algorithms. Education quality depends on teacher's knowledge and engagement. Outreach activities have a significant influence on informatics curricula e.g. the Bebras Challenge on Informatics and Computational Thinking [6]. Students in upper secondary school (grades 11 and 12) have the option of choosing advanced modules on programming, data bases and a range of other CS topics.

2.2 Finland

In Finland, Computer Science (CS) was part of the upper secondary curriculum (grades 10 to 12) until the 1994 national curriculum reform, later ICT (Information and Communication Technologies) were integrated in all subjects. In 2016, Finland introduced a new national curriculum for general education (grades 1–9). The new curriculum includes both ICT-related skills and programming. Programming is explicitly included in mathematics education, starting with primary students giving instructions to each other and moving towards graphical programming environments and using programming languages in secondary school. The introduction of programming in the core curriculum has resulted in different initiatives facilitating and supporting this reform. For instance, a teacher guide called Koodi2016 (Code 2016) was published in early June 2014 with both state and industry support [12].

The National Board of Education, which is in charge of the curricula reform, provides some funding for professional development aimed at in-service teachers as well as curriculum related to the use of programming environments and tools in schools. However it is the duty of education providers like municipalities to ensure a sufficient in-service training for teachers to increase the professional skills and professional self-esteem.

The new curriculum has clearly intensified the use of computing in the classes and classrooms. Many teachers are however looking forward to have a more radical integration process to intensify the use computers in everyday situations to make computers an essential part of learning.

The teacher students at the universities are introduced to computational thinking quite deeply but a lot of practice in schools is needed.

2.3 Sweden

The first curriculum for K-12 CS education in Swedish schools was established in the 1970s, under the term informatics, later changed to information technology. Informatics was a supplementary subject in vocational education, and later became obligatory for upper secondary students in the natural science programmes [9]. For a number of years computing related material was offered at primary and lower secondary level, but was later removed as primary and secondary teachers were not properly qualified to teach the content, and more specifically the programming element proved hard for many teachers to teach.

Over the last decade computer science has transformed into a subject mainly focusing on digital literacy at the primary and secondary levels, with primary focus on applications and tools (e.g. the Office suite) and its applications in other subjects. In Sweden's compulsory school CS has not been taught as a separate subject. Instead the content is offered as parts of other school subjects like Technology and Mathematics. In the upper secondary schools (years 10–12), the CS education varies depending on the programme attended.

Teacher training in Sweden does not include courses in CS, making it is unusual to find teachers with computing expertise in many schools. There are courses at upper secondary school in programming, multimedia, web design etc. that are offered to any student who wishes to take them. However, programming courses are mainly attended by students who attend one of the three programmes in technology, natural sciences or electronics. Hence, historically only a minority of students have taken programming courses.

The ICT content in the Swedish educational system is summarized as follows: One ICT learning outcome is specified for Swedish preschools, which prescribes that pupils shall become capable in the use of "multimedia and information technology in creative processes as well as in applications."

In the compulsory school (grades 1–9) ICT training focuses on equipping students with the "how of information", how to use digital technology, how critical thinking can be applied to the information available on the Internet, etc.

In 2012, the Swedish government established the Committee for Digitization (Swedish: Digitaliseringskommisionen) to provide direction and guidelines for future work on the digitalisation of Swedish society. The committee recently published a report which pinpoints the need for school to embrace the concept of digital competency, in accordance with the DigiComp framework developed in the European Union.

In 2017 the government introduced a revised curriculum which focused more on digital competences and the development of relevant computing and computational thinking skills at all levels of the K-12 school system.

3 Teacher Attitudes to Computing

Programming and system development skills have been an issue in terms of teachers' competencies from the initial phases of all the initiatives we described above. Riis [13] notes that "The teachers whose ideas are expressed in the school reports are enthusiastic, competent, and generally very optimistic about the future, and they have a constructive attitude toward computer use in school. However, they hardly represent the majority of Swedish teachers today. Their experience perhaps can not in all cases and without difficulty be transferred to others. This is an area in which in-service training could give these teachers a chance to share their experiences with those who have not been in the front ranks." To assess the current situation in the teaching community this study of teacher attitudes to teaching computing in schools collected data using two surveys (Survey I and Survey II) in the NordNICE project.

Survey I was developed during a NordNICE workshop in Druskininkai in Lithuania in December of 2015. The full survey can be found in through the

WWW link in Appendix A. This survey uses items from several earlier research initiatives [14] and comprises three major sections, background demographics, common classroom activities, and a concept association table.

The aim of the first survey was to obtain a broad picture of the attitudes and associations of teachers in relation to central concepts in computational thinking and computing identified in prior research. The survey was broadly distributed through teacher networks and social media to teachers in all the countries participating in the action.

Survey II uses a purposeful sampling approach with the intention to select responses which represent a heterogeneous (maximum variation) sample. This was done by intentionally selecting 3–5 schools in each country using "engagement with computing" as a primary variable for selection. The goal was to collect data from teachers who are already actively teaching computing content as well as teachers in schools where teaching of computing content was not yet implemented. Schools in Lithuania, Latvia, Estonia, Sweden and Finland participated in this second data collection as shown in Fig. 1.

Country	N	Teaching	Levels
Estonia	9	2 @ < 5 years, 3 @ 5-10 years	Grades 2-12
Latvia	6	1 @ 11-15 years, 5 @ 20+ years	Grades 4-12
Sweden	9	3 @ < 5, 2 @5-10 years, 2 @11-15 years, 2 @ 16-20 years	Grades K-12
Finland	7	1 @ < 5, 2 @16-20 years, 4 @ 20+ years	Grades 1-12
Lithuania	14	6 @ 5-10 years, 1 @ 16-20, 4 @ 20+ years	Grades 1-12

Fig. 1. Survey II respondents

3.1 Data Analysis

The data collected in survey I on teacher's associations between a range of terms and conceptual categories, Computational Thinking, Information Technology and so forth were summarized and plotted as a heat map to reveal the patterns of association in the survey population.

Both surveys also collected qualitative data in the form of responses to a number of free text questions. These data have been coded using a thematic coding scheme using the TAMS software. The results presented in this paper draw on a detailed analysis of the data collected in Survey II in response to two questions which asked teachers to reflect on teaching challenges they had experienced and teaching practices they adopted.

The thematic coding focused on capturing teacher attitudes and teaching content, and was conducted in accordance with standard inductive coding principles, as described by Mayring [15]. Three iterations of classification were conducted by the primary author to arrive at the clusters presented in Sect. 3.2.

3.2 Results and Discussion

Preparedness: The data collection includes a total of 504 responses of which 144 are complete and 360 are primarily demographic information about the respondents. Lithuanian teachers dominate the sample, contributing approximately two thirds of the responses, Swedish teachers comprise most of the remaining third and Finnish teachers approximately 6% of the total sample. The final question which asks respondents to associate terms and concepts with a number of main focus areas, Computational Thinking, Computer Science, Information Technology, Information and Communication Technology, Programming, Mathematics, Algorithmic Thinking was answered by 144 respondents.

The majority of the respondents were directly involved in teaching computing and IT concepts in their schools. Teachers from nearly all year levels are represented in the sample (see Fig. 2 for the distribution between year levels). The sample also includes many very experienced teachers, the mean number of years in teaching service in schools was 17 years among respondents, and approximately 60% of the respondents were over 41 years of age (30% in the age group 41–50 years). Most respondents felt that computing skills were important ranking them as “vital” (40%) “very important” (40%) and “somewhat important” (20%).

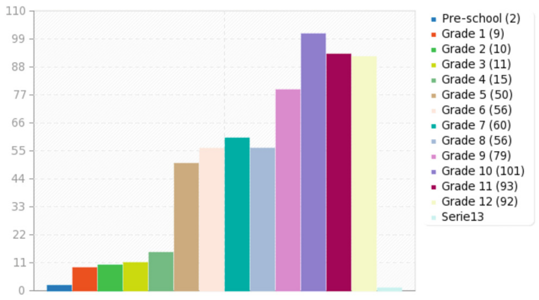


Fig. 2. Respondents teaching level

The curriculum in Sweden, Finland and Lithuania in 2016 emphasized the teaching of CS content, especially programming in the last three years of school education, typically years 10 to 12. The responses to Survey I reinforce this, most of the teachers with active experience in teaching computing related content were also heavily engaged in teaching at the upper secondary and middle school level. While the largest volume of teaching is clearly in the years 10 to 12 of the school system, there is also significant amounts of CS being taught in years 5 to 9.

Analysis: Many respondents felt that their school provided a somewhat, or very, supportive environment for their efforts (78%). While this is encouraging these teachers were already predominantly engaged in teaching technology, computing mathematics or natural science subjects in their schools, and are also from countries where national curricula include CS as a formal subject. In the light of the background of the respondents this result, while reassuring for Lithuanian and Finnish school policy makers, is perhaps not particularly surprising.

A heat map which maps intensity of association in the response group of terms and concepts to a range of higher level aggregate terms is presented in Fig. 3.

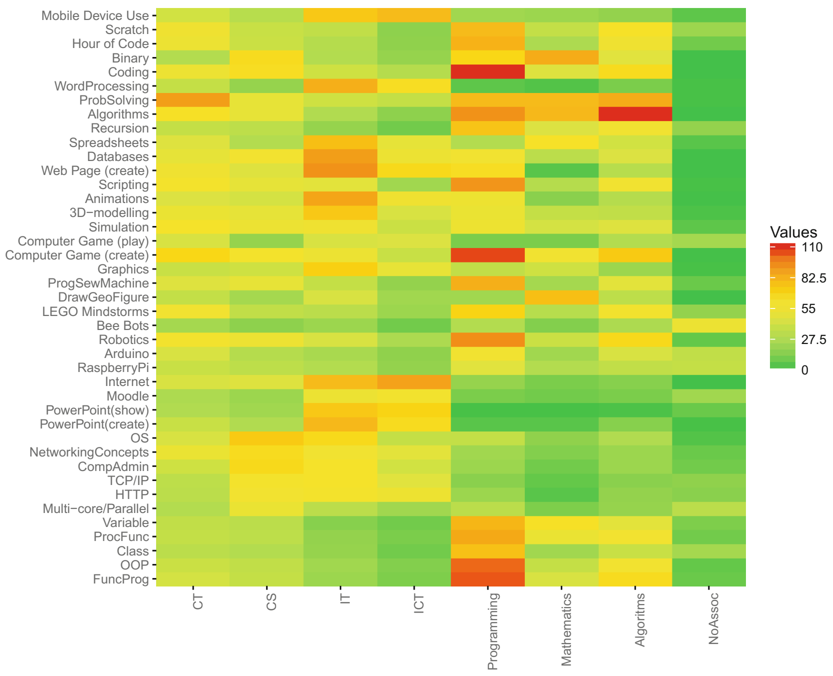


Fig. 3. Heat map of teachers' concept associations

Teachers clearly had a clearer set of associations with Programming than with many other terms and concepts listed in the table. Interestingly mobile devices Use was quite strongly associated with IT and ICT, as was Internet. Bee Bots were not clearly associated with any area, which may imply that the use of these devices to reinforce relevant concepts is unclear to teachers. This warrants further investigation, since robotics was clearly associated with programming, algorithms and CT or CS by many teachers in the sample. Problem solving and algorithms as were strongly associated with programming, mathematics and algorithms which reinforces the conclusion that teachers consider these concepts as a knowledge cluster.

The only strong association with Word Processing was IT, which might reflect the emphasis on tools such as Power Point and Word in the Swedish school IT curriculum. Recursion was associated with programming and algorithms, but less strongly with CS and mathematics, another area which may motivate additional research.

Identifying Challenges: The sampling of teachers at schools in Estonia, Latvia, Finland, Sweden and Lithuania resulted in 44 responses. The number of responses from each country is shown in Fig. 1. The answers to question X are indicative of a general consensus among the respondents that computing content is important, and should be included in the school curriculum.

Our data reveals a central concern among the teachers sampled related to student conceptual development and the relationship between the materials and examples used to teach computing and other subject knowledge such as mathematics and statistics. Historically speaking the traditional applications used to teach computing concepts are dominated by examples drawn from Mathematics, one of the disciplines from which Computing emerged during the 1960's and 1970's [16]. The emergence of computing in schools requires new resources be developed to meet the needs of contextualizing learning of computing for students of ages from 5 to 18. This is expressed concisely by one of our respondents, who comments as follows.

> "Younger students didn't find appealing making graphs in Excel because they don't know much about statistic in math yet. I must make graphs about topics that are relatable [sic] to their everyday."

In addition new classroom resources are needed to support teachers as they introduce computational thinking content into subjects like languages, arts and crafts, and technology.

An in depth thematic analysis of teachers motivations for teaching computing content in their classes results in two dimensions *motive* and *goal*. In the motivational dimension values identified are *intrinsic, profession, societal*. While in the goal dimension values are associated with the pupil and focus on aspects of *cognitive development, enrichment, empowerment, future agency, and workplace skills*. Many teachers express an intrinsic motivation to engage with teaching computing. The second greatest motivator is a sense of professionalism, this is expected of them in their teaching role, and is important for the future of society.

However, many of the teachers also felt that they were compelled to engage with the challenges of introducing computing competencies into their educational activities. Many were also clearly uncertain about their abilities. What prompted feelings of insufficiency and failure among our sample of teachers? Further analysis was conducted focusing on data collected in response the following question.

> "Describe at least one teaching activity or lesson where you felt that you failed in introducing aspects of computing in your teaching. What made the experience feel like a failure? What did you learn from it?"

Here the majority of responses indicate that lack of appropriate examples and teaching materials, as well as the need to teach outside one's personal comfort zone are the two areas of greatest difficulty. A typical statement in this regard describes the challenges experienced by many of our participants.

> "I am not at all as in control of the situation as I usually am, since I am not a particularly good programmer myself. The students manage to solve most problems themselves, but sometimes it feels very frustrating not to be able to help them when they get stuck."

This highlights the need for in service training in programming for teachers who engage in more advanced programming based classroom exercises.

4 Addressing Challenges with Action

The NordNICE project established a number of direct actions to support the lack of support perceived by teachers. The very low levels of computing content teaching in early school years reflect the current status of the curriculum, but also highlight a potential lack of resources, since there is little incentive to produce teaching materials for the current low volumes of pupils. The NordNICE project has directly addressed this need.

The project repository collects teaching materials from the project school partners helping to create a Nordic and Baltic repository of educational resources which can support teachers at all school levels. These materials include classroom exercises, resource books for teachers, and examples of good practice in a range of disciplinary application areas from Art and Craft courses to Technology, Mathematics and Physics.

To address the issue of accessible resources for younger learners the project has developed translations of Bebras task cards for different age groups into several European languages, including Finnish, Swedish, English and German.

While the NordNICE project conducted a series of in-service training activities for teachers, and organized network events where groups of teachers from the participating countries made site visits to each other's schools and met to exchange ideas, and observe practices, much more focus on such initiatives is needed. The teachers participating in our initiative were very positive to the experience, and emphasized the importance of building up support networks and training for themselves and their colleagues through similar national and international initiatives. Larger national programmes should be created to support teacher's professional development over the next five years as the new curriculum is introduced.

5 Conclusion

The development of holistic curricula for computational thinking and computing concepts and techniques presents many opportunities and challenges. The increasing dependence of our society and citizens on digital services, products and systems makes the revision of curricula to include new skills which empower future generations to manage a digital world vital to our future wellbeing.

This paper provides a Nordic and Baltic school teacher perspective on these challenges and in particular how well governments, the teaching profession and teacher training institutes are meeting these challenges. Our results indicate that there is a competent cohort of in-service teachers who are very capable of managing the transition to the new curricula. However, many of these teachers have more than twenty years in the profession and are nearing retirement age. This will leave a considerable competence gap to be filled in many countries (see Fig. 1).

At the same time, many teachers experience a lack of support, in particular access to appropriate teaching materials which contextualize computing into their subject area. The NordNICE project is a first step towards an integrated approach that draws together expertise from Computing Education Research in Higher Education, in-service teachers as mentors and coaches, and teacher training programmes to generate new approaches and provide access to innovative materials.

While some progress has been made, much still needs to be done. Our work suggests that it will be necessary to restructure teacher training programmes in all subjects to provide appropriate computational thinking and computing content knowledge to future teachers. Without such an investment the new curricula are at risk and pupils left without an appropriate high quality education. Failure to attend carefully to these issues, especially failure to invest in teacher training, places the future industrial competitiveness of the Nordic and Baltic regions at risk.

Acknowledgements. The authors gratefully acknowledge the support of the Nordic Research Council through the NordPLUS programme of transverse actions. In particular the funding of the two year network initiative, NordNICE, under project code NPHZ-2015/10153.

A Appendix: Questionaires

The questionaires used to collect the data used in this paper are available on the following link. Questionaires

References

1. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books Inc., New York (1980)
2. Tedre, M., Denning, P.J.: The long quest for computational thinking. In: Koli Calling, pp. 120–129 (2016)
3. Wing, J.M.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)
4. Wing, J.M.: Computational thinking. In: VL/HCC, p. 3 (2011)
5. Benaya, T., Zur, E., Dagiene, V., Stupuriene, G.: Computer science high school curriculum in israel and lithuania-comparison and teachers' views. Baltic J. Mod. Comput. **5**(2), 164 (2017)
6. Dagiene, V., Stupuriene, G.: Bebras-a sustainable community building model for the concept based learning of informatics and computational thinking. Inform. Educ. **15**(1), 25 (2016)

7. Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., Boyle, R., Mendelson, A., Stephenson, C., Ghezzi, C., et al.: Informatics education: Europe cannot afford to miss the boat. ACM (2013). http://europe.acm.org/iereport/ie.html
8. Hubwieser, P., Armoni, M., Giannakos, M.N., Mittermeir, R.T.: Perspectives and visions of computer science education in primary and secondary (k-12) schools. ACM Trans. Comput. Educ. (TOCE) **14**(2), 7 (2014)
9. Rolandsson, L., Skogh, I.B.: Programming in school: Look back to move forward. Trans. Comput. Educ. **14**(2), 12:1–12:25 (2014)
10. Clear, T., Daniels, M., Cajander, Å., Parsjö, E., Lagerqvist, N., McDermott, R.: A framework for writing personal learning agreements. In: IEEE/ASEE Frontiers in Education Conference (2016)
11. Dagienė, V.: Teaching information technology and elements of informatics in lower secondary schools: curricula, didactic provision and implementation. In: Mittermeir, R.T., Sysło, M.M. (eds.) ISSEP 2008. LNCS, vol. 5090, pp. 293–304. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69924-8_27
12. Liukas, L., Mykkanen., J.: Koodi 2016 - ensiapuaohjelmoinnin opetukseen peruskoulussa, June 2016. http://www.koodi2016.fi
13. Riis, U., ed.: IT i skolan mellan vision och praktik: En forskningsöversikt. Number BEST. NR. 00:560 in ISBN 91-89313-89-5. Liber Distribution, Publikationstjänst, 162 89 Stockholm, Skolverket (2000)
14. Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., Settle, A.: Computational thinking in k-9 education. In: Proceedings of the Working Group Reports of the 2014 on Innovation & #38; Technology in Computer Science Education Conference. ITiCSE-WGR 2014, pp. 1–29. ACM, New York (2014)
15. Mayring, P.: Qualitative content analysis. Forum Qualitative Sozialforschung / Forum: Qualitative Social Research 1(2) (2000)
16. Tedre, M.: The Science of Computing: Shaping a Discipline. Taylor & Francis, Boca Raton (2014)

Computational Thinking/Programming, Computational Abilities

Identifying Students' Misconceptions on Basic Algorithmic Concepts Through Flowchart Analysis

Ebrahim Rahimi[1(✉)], Erik Barendsen[2], and Ineke Henze[3]

[1] Radboud University, Nijmegen, The Netherlands
e.rahimi@cs.ru.nl

[2] Radboud University and Open University, Nijmegen, The Netherlands
e.barendsen@cs.ru.nl

[3] Delft University of Technology, Delft, The Netherlands
f.a.henze-rietveld@tudelft.nl

Abstract. In this paper, a flowchart-based approach to identifying secondary school students' misconceptions (in a broad sense) on basic algorithm concepts is introduced. This approach uses student-generated flowcharts as the units of analysis and examines them against plan composition and construct-based programming problems to identify students' misconceptions. In this study, 102 flowcharts, generated by 50 students in two informatics classes in the Netherlands, were examined and various sorts of misconceptions were identified. The results suggest that, given their abstract and language-independent nature, flowcharts can be considered as an effective tool for revealing students' difficulties in understanding algorithmic concepts. Our approach contrasts the more traditional use of program code to investigate students' misconceptions. We found several misconceptions mentioned in the literature, together with two misconceptions which appear not to have been described before. Our research contributes to the usage of flowcharts as a formative assessment tool, directing informatics teachers' instruction toward resolving these misconceptions.

1 Introduction

There is an ongoing global debate among educational policy makers, researchers, technology developers and practitioners around establishing computer science (CS) and in particular computer programming as an integral part of school curricula at different levels [1,6]. The advocates argue that due to the highly technology-driven and information-dependent nature of working and living in the 21st century, learning fundamental computer science concepts and programming has become a must for everyone [6]. They argue that CS represents an essential and unprecedented sort of literacy people need to survive in the knowledge era where thinking critically and computationally to solve complex, ill-defined, and open-ended problems is becoming more and more important [21]. This sort of

© Springer International Publishing AG 2017
V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 155–168, 2017.
https://doi.org/10.1007/978-3-319-71483-7_13

literacy is meant to help people to not simply become technology consumers, but also to serve as technology creators and knowledge developers [6,10].

Programming generally includes two main tasks: algorithmic problem solving and coding. The problem solving part includes analyzing the problem, formulating a solution, and devising an algorithm for implementing the solution. The coding part consists of implementing the devised algorithm using a programming language, debugging and modification [15]. Algorithmic thinking thus plays an essential role in problem solving and programming and accordingly has a key position in computational thinking notions [21]. Despite the important role of problem solving and algorithmic thinking in programming, in general, these aspects seem to have received less attention than coding within CS in secondary education [18].

Flowcharts are advocated as useful pictorial representations of algorithms, the program logic or its flow of control. They have been an integral element of programming since the introduction of computers in 1940s [15]. Flowcharts have been used as simple and effective tools to comprehend, devise, modify, visualize, debug, express and communicate algorithms and programs [12]. As remarked by [15], a flowchart represents "a high level definition of the solution to be implemented on a machine" (p. 1). Given their abstract and language-independent nature, flowcharts have been suggested as effective tools for the novice programmers to enhance their problem solving and algorithmic thinking competencies, and to promote a "thinking first" approach to programming [18]. As held by [2], *flowcharting* helps "distinguish between the procedure a computer program is written to express and the syntactical details of the language in which the program is written" (p. 53).

The purpose of this study is to investigate the usability of flowcharts as a tool for identifying and highlighting students' misconceptions on basic algorithmic concepts. Following [20], we will use the term *misconception* as an overall term covering students' lack of understanding, problems, mistakes, bugs, and difficulties with basic algorithm concepts. Flowcharts can be seen as a way to get into the students' minds and provide a better picture of their understanding of basic computer science concepts. By doing so, flowcharts ultimately serve to improve the quality of computer science education through a better understanding of what goes wrong [20]. In [18] an analytical approach to identifying programming misconceptions using flowcharts is introduced. This approach uses two categories of plan composition and construct-based problems introduced by [20] to detect students' misconceptions. It was used to detect programming misconceptions exhibited by 11 high school students in the Netherlands.

Our study adopted the approach introduced in [18] and modified it to detect students' misconceptions on each of basic algorithmic concepts including sequence, condition, iteration, and sub-algorithm. To this end, 102 flowcharts generated by 50 students in two informatics classes in the context of upper secondary education in the Netherlands were examined.

This study is a part of a bigger research project with the main objective of encouraging and facilitating the learning of computer science concepts through conducting *authentic design assignments* (see [9]). This study plays a multifold role in this research project: (*i*) highlighting the importance of student-generated flowcharts as intermediate design products, (*ii*) examining the usability of flowcharts as an effective formative assessment tool, (*iii*) helping the teachers and researchers in identifying students' misconceptions on basic algorithm concepts, and (*iv*) directing teachers instruction toward resolving these misconceptions. This way, applying flowcharts as a formative assessment tool might help to further develop the teachers' PCK (Pedagogical Content Knowledge) [16] on algorithms.

2 Programming Misconceptions

There are worldwide observations that students, regardless of institution or country, show generally poor performance in the introductory programming courses and learning programming is a difficult task for novices [3,17]. A reason for this difficulty stems from the fact that programming puts a high cognitive and knowledge demand on novices including knowledge on a specific programming language and knowledge and understanding of basic programming concepts and constructs such as variables, loops, conditions, abstraction, and procedures. Any misconception on these constructs might result in programming difficulties [17,18]. Therefore, realizing and resolving these misconceptions seems useful in diminishing the novices difficulties in programming.

Students' misconceptions on basic programming constructs and features have been vastly researched (see [4,8,11,14,17,19,20]). Four common examples of the identified programming misconceptions include considering classes as containers for objects, reverse assignment, interpreting assignment statements as mathematical equations, and boundary problem (i.e., choosing inappropriate boundary points) [17,20]. Seppälä et al. in [13] categorized students' errors on algorithm exercises into systematic and careless errors ('slips'). According to [13], slips result from randomly trying out algorithm exercises by students, whereas a systematic error is "a symptom of a misconception that could be corrected if recognized" (p. 244). According to [20], students' misconceptions on programming fall in two categories: *plan composition* and *construct-based* problems. Plan composition problems refer to the encountered misconceptions in composing a solution by putting the pieces of plans together. These misconceptions are concerned with the solution formulation, algorithm development, and planning the semantic and logic of the program [3]. On the other hand, construct-based problems concern with the misconceptions on language constructs and the syntax of the program. Tables 1 and 2 summarize plan composition and construct-based problems introduced by [18,20].

However, the majority of these misconceptions have been researched and recognized in the context of program's code in specific programming languages. There are very few studies examining the programming misconceptions at a more abstract level such as algorithms and flowcharts (see [18]). Arguably, the sorts of misconceptions exhibited by students in a programming language depend, to

Table 1. Plan composition problems, adopted from [20]

Misconception	Description
Summarization problem	The complex combinations of plans are summarized in terms of some primary functions and secondary functions have been overlooked
Optimization problem	Novices aim to optimize their plans but do not adequately check if the optimization can really be carried out
Previous-experience (or pollution) problem	Novices constantly develop and tailor plans on the basis of previous experience. This error is introduced when inappropriate aspects of previous plans pollute a related plan that is being used in a new situation
Specialization problem	Inappropriate and incorrect customization of an abstract plan developed for other situations
Natural-language problem	Errors happen during the process of mapping from natural language to a programming language.
Boundary problem	Novices difficulties for deciding on appropriate boundary points in specializing a plan
Unexpected case problem	The program is not working correctly for all cases (e.g. uncommon, unlikely, or boundary cases)
Interpretation problem	Not considering the implicit specifications of plans or misinterpreting them
Cognitive load problem	Omitting and overlooking small but important parts of the plan or plan interactions

Table 2. Construct-based problems, adopted from [18, 20]

Misconception	Description
Human interpretation problem	Novices assumption that computers are able to interpret problems as people do
Assignment misconception	Inverted assignment: the positions of the giver and receiver variables at the right and left side of the assignment operator are misplaced
Condition misconception	Misconceptions about how the condition construct (or control structure) works
Boolean statements	Misconceptions on how boolean statements behave
Loops	Misconception on how a loop control variable works
Method-related misconceptions	Misconceptions on method calling
Control flow	Misconceptions such as incorrect use of print and return statements, omitting arrows in a flowchart, or omitting start or stop steps

some extent, on the specifications of that programming language and might be shaped or filtered by its structures and specifications. In contrast, flowcharts provide a more abstract and language-independent way to get into the minds of students and seem useful in gaining a firsthand and more comprehensive picture of what goes wrong in their minds.

3 Study Setting

The participants in this study were 50 students of age 15 or 16 from two schools in upper secondary education in the Netherlands. All of the participants were at the university preparatory education level (VWO in Dutch) [7]. In the aforementioned surrounding research project (see the introduction section), the participants were provided with the background and design documents. The background document covered various aspects of algorithms including the definition, basic concepts (i.e. sequence, condition, iteration, and sub-algorithms), trace table, and flowcharts. Each of these aspects was explained using several examples and questions with a different level of difficulty. The design document included seven authentic design assignments in the context of text analysis. In each assignment, students were asked to follow a step-wised approach to learning algorithms and programming. These steps include analyzing the design problem, formulating a conceptual solution, devising an algorithm, developing a flowchart to implement the algorithm, testing the flowchart using trace tables, and converting their flowcharts into a program using a programming language (either PHP or Python, as the students had learned one of these languages in the previous years). Students could use the background document whenever necessary for explanation and advice on conducting these steps. To embed formative assessment within this educational intervention, students were asked to answer three exit questions during their design endeavours on a weekly basis. These questions were derived from the background document and were meant to assess students' understanding of basic algorithm concepts underpinning the design assignments. In each question, the students were asked to draw a flowchart for solving the given problem. To foster collaboration and team working, the students were grouped into teams of 2–3 students and each team developed a flowchart for each of the questions. The flowcharts were written on paper or developed digitally using a specific tool called Draw.io and were handed over manually or via the school's learning management system. The teachers then could use these flowcharts to detect students' misconceptions and provide them with appropriate feedback. Table 3 presents these questions and their associated algorithm concepts.

The following research question directed the data collection and analysis processes: *What misconceptions can be seen in students' flowcharts?*

The flowcharts developed by the participating students in response to the questions in Table 3 were used as the data source to answer this research question. A qualitative deductive content analysis approach was followed to analyze the flowcharts. Content analysis is a method used for analyzing written, verbal or visual communication messages and documents [5]. Following the process of

Table 3. Questions asked to investigate students' understanding of basic algorithmic concepts

Question	Concepts
Q1: Write an algorithm that receives three numbers (a, b, c) and determines and reports the maximum and minimum numbers	Sequence, condition
Q2: (i) Write two sub-algorithms *findmax (a,b)* and *findmin (a,b)* which receive two numbers a, b and determine and return the maximum and minimum numbers, respectively (ii) Using these sub-algorithms write an algorithm that receives 10 numbers (i.e., a_0, a_1, ... , a_9) and determines and reports the maximum and minimum numbers	Sub-algorithms, problem composition and decomposition, condition
Q3: Write an algorithm that receives a text value (i.e. t) as input and reverses it (i.e. $t =$ "book", reverse of $t =$ "koob")	Loop, condition

deductive content analysis introduced by [5], three steps of preparation, organization, and reporting were taken. In the preparation step, all of the flowcharts (102 in total) as the unit of analysis were uploaded into Atlas.ti software. In the organizing phase, first the plan composition and construct-based problems presented in Tables 1 and 2 were chosen as the initial categories for coding the observed misconceptions in the flowcharts. Then following an iterative process of coding and revising, the misconceptions on basic algorithm concepts observed in the flowcharts were coded according to these categories. The coding process was flexible to allow emerging new codes which did not exist in these categories. Possible alternative interpretations of the observed misconceptions were discussed within the research team until a consensus was reached. Finally, the identified misconceptions on each of basic algorithm concepts were processed and reported as can be seen in the next section.

4 Results

Tables 4 and 5 present the identified *plan composition* and *construct-based* problems in the flowcharts, respectively.

• Plan composition problems

(i) *Condition misconceptions*: Three plan composition misconceptions for the *condition concept* were identified. *Not considering equal inputs in plans* is an *unexpected case problem* where the equal numbers are excluded from a solution, specifically for answering questions 1 and 2 in Table 3. This was one of the most frequently observed misconceptions in the flowcharts. Interestingly, some of the students reacted to passing equal numbers to their flowcharts as a fatal error (see Fig. 1(a)). *Incorrect adaptation* is the next condition misconception and represents a *previous experience problem*. Figure 1 (b) illustrates an example of this

Table 4. The plan composition problems observed in the flowcharts

Category	Misconceptions
Condition misconceptions	*Not considering equal inputs in plans (unexpected case problem)*: the situations where the equal numbers are excluded from the comparison operations
	Incorrect adaptation (previous experience problem): refers to incorrect applying of a previous condition-related plan in another situation
	Misinterpreting the condition concept (Interpretation problem)
Loop misconceptions	*Incorrect loop construction*
Sub-algorithm misconceptions	*Composition problem*: refers to students problems and mistakes with composing an algorithm by putting together sub-algorithms
Translation	*Mapping problem*: incorrect mapping from natural language or programming languages to flowcharts

misconception where an incorrect adaptation of the plan used for determining the maximum number (i.e. *max*) between *a*, *b* is used for calculating the minimum number (i.e. *min*). *Misinterpreting the logic of condition concept* is a sort of *interpretation problem* with regard to the condition concept, as depicted by Fig. 1 (c).

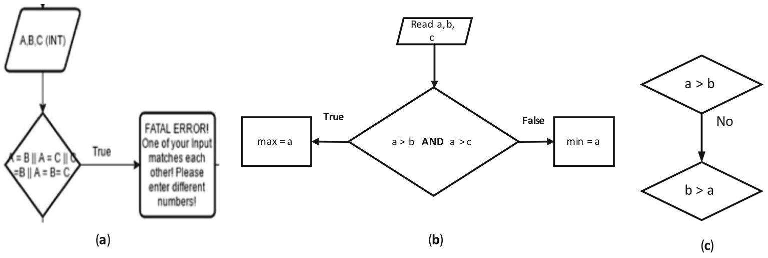

A,B,C (INT)
A = B || A = C || C =B || A = B= C
True
FATAL ERROR! One of your Input matches each other! Please enter different numbers!
(a)
Read a,b, c
a > b AND a > c
True
max = a
False
min = a
(b)
a > b
No
b > a
(c)

Fig. 1. Examples of identified plan composition problems related to the condition concept

(ii) *Iteration misconceptions*: *Incorrect loop construction* is the encountered misconception related to the implementation of the iteration concept. Figure 2 depicts an example of this misconception.

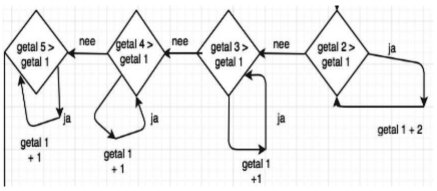
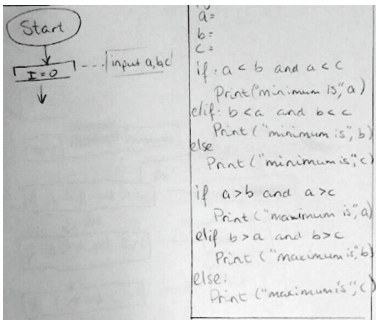


Fig. 2. An example of incorrect loop implementations

(iii) *Sub-algorithm misconceptions*: The *composition problem* refers to students problems, difficulties, and mistakes with composing an algorithm using its sub-algorithms. The results showed that while many of the students devised the asked sub-algorithms (i.e. *findmax* and *findmin*) correctly, many of them were not able to use the developed sub-algorithms to compose the main algorithm.

(iv) *Translation misconceptions*: The *mapping problem* or incorrect translation from formal language or programming languages to flowcharts was another plan composition problem observed in the flowcharts. Figure 3 illustrates an example of this misconception. Surprisingly, as shown by this example, although some of the students already know how to program, but they cannot map their program into a flowchart.


Start
I=0
input a,b,c

```
a=
b=
c=
if: a < b and a < c
    Print("minimum is", a)
elif: b<a and b<c
    Print ("minimum is", b)
else
    Print ("minimum is", c)
if a>b and a>c
    Print ("maximum is", a)
elif b>a and b>c
    Print ("maximum is", b)
else:
    Print ("maximum is", c)
```

Fig. 3. An example of difficulty in mapping a plan from programming languages to a flowcharts

• Construct-based Misconceptions

(i) *Condition misconceptions*: *Missing the false part* represents a misconception where only the *true* output of a condition statement is addressed. Figure 4 (a) presents an example of this misconception. The *3-output condition* is another misconception where 3 possible outputs are considered for a conditional statement, as shown in Fig. 4 (b).

(ii) *Sequence misconceptions*: The sequence misconceptions were among the frequent problems observed in the flowcharts. These misconceptions reflect students' misunderstanding about the flowchart's flow control and sequential execution of its steps. The *parallel execution misconception* reflects a perception

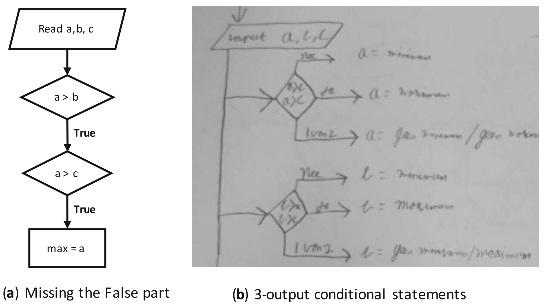

Read a,b, c
a > b
True
a > c
True
max = a
(a) Missing the False part
(b) 3-output conditional statements

Fig. 4. Example of construct-based misconceptions for the condition concept

held by some of the students that the flowchart's steps can run in parallel to each other, as shown by Fig. 5 (a). Another observed misconception with the sequence concept is called the *dense step* which refers to a flowchart's step that executes more than one operations. Figure 5 (b) shows two examples of this misconception. It seems that in these examples the students assumed that the flowcharts interpret these steps like what people do. The last sequence misconception concerns with students' *issues with Input/output and start/stop steps.* These issues refer to situations where students forgot to get input, report or return the final results, forgot to add start and stop steps in their flowcharts, or used return instead of report.

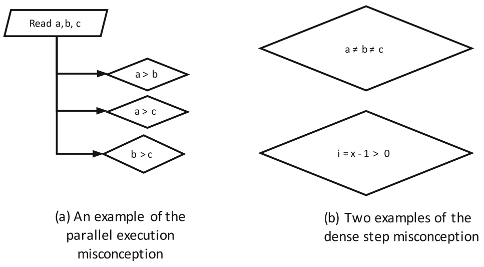

Read a,b, c
a > b
a > c
b >c
a ≠ b ≠ c
i = x - 1 > 0
(a) An example of the parallel execution misconception
(b) Two examples of the dense step misconception

Fig. 5. Examples of construct-based misconceptions for the sequence concept

(iii) *Loop misconceptions*: Figure 6 illustrates an example of the identified construct-based loop misconception. This misconception is concerned with the *simultaneous initialization and check of the loop's control variable.* As can be realized from this example, this misconception might lead to either infinite or zero-repeating loops, depending on the value of *len(word1) - 1.*

(iv) *Assignment misconceptions*: Two construct-based misconceptions related to the assignment operation were discerned in the flowcharts. The most occurring error in the students flowcharts was the *inverted assignment* where the positions of the source and destination variables in the right and left of the assignment operator were misplaced. Figure 7 (a) shows an example of this misconception. In this example the students were asked to find the maximum and minimum numbers among variables a, b, c and store them in variables max and min, respectively. Another type of the observed assignment issues is the *missing value* misconception where a value is processes but not stored in a variable. Figure 7 (b) shows an example of this misconception.

Table 5. The construct-based problems observed in the flowcharts

Category	Misconceptions
Condition misconceptions	*Missing the False part*: conditional statements where the false part (or the else part) is missing
	3-output condition: conditional statements with 3 outputs
Sequence misconceptions	*Dense step (Human interpreter problem)*: situations where more than one operations are processed in one step
	Parallel execution of steps (flow control)
	Issues with Input/output and start/stop steps (flow control)
Loop misconceptions	*Simultaneous initializing and checking of the loop control variable*
Assignment misconceptions	*Inverted assignment*: situations where the positions of the source and destination statements of the assignment operator are misplaced
	3-output condition: conditional statements with 3 outputs
Sub-algorithm misconceptions	*Incorrect use of sub-algorithms*: for instance using a sub-algorithm at the left side of an assignment operator
	Issues of calling/returning from sub-algorithms
Flowchart presentation	*Using incorrect shapes for the flowchart's constructs*

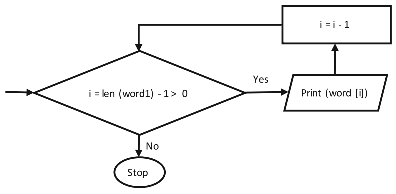
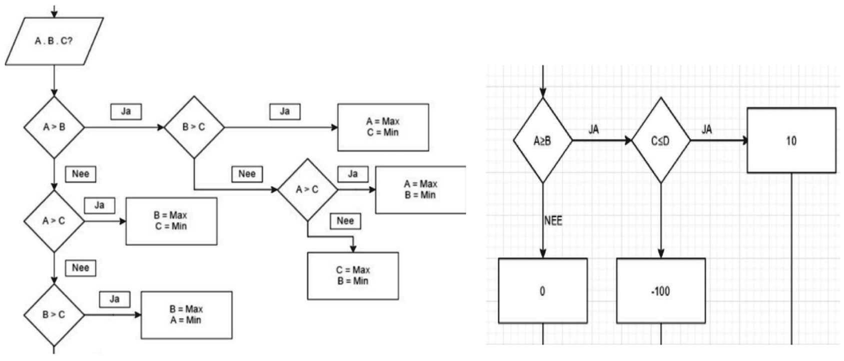


Fig. 6. An example of the loop misconception



(a) An example of the inverted assignment misconception

(b) An example of the missing value misconception

Fig. 7. Two examples of the assignment misconceptions

(v) *Sub-algorithm misconceptions*: The *Incorrect use of sub-algorithms* exhibits an misunderstanding and wrong interpretation of the usage of sub-algorithms. For instance, it was observed that some of the students interpreted a sub-algorithm as a variable or container and used it at the left side of an assignment operator. *Issues of calling/returning from sub-algorithms* represent another sub-algorithm misconception. Examples of this misconception include: not passing input parameters to sub-algorithms and not returning results from sub-algorithms.

(vi) *Flowchart presentation*: This misconception refers to situations where incorrect shapes were used by the students to present various constructs of their flowcharts. For example, several students used rectangle instead of diamond for presenting conditional statements.

5 Conclusions and Discussion

In this study, a flowchart-based analysis approach, adapted from [18,20], was introduced and followed to identify the misconceptions of basic algorithm concepts exhibited by 50 students in upper secondary education in the Netherlands. Various types of plan composition and construct-based misconceptions on basic algorithm concepts have been identified in the analyzed flowcharts. The identified plan composition misconceptions are: *not including equal numbers in plans* (an unexpected case misconception), *incorrect adaptation* (a previous experience misconception), *misinterpreting the condition concept* (an interpretation misconception), *incorrect loop construction*, *composition problem* (i.e. devising an algorithm using sub-algorithms), and *mapping problem*. The discerned construct-based misconceptions include *missing the false part* of conditional statements, *3-output condition*, *parallel execution of steps* (a flow control misconception), *dense step* (a human interpreter misconception), *missing input/output and start/stop* *steps*, *incorrect initialization and checking of the loop counter*, *inverted assignment*, *missing values*, *incorrect use of sub-algorithms*, *issues of calling/returning from sub-algorithms*, and *using incorrect shapes for the flowchart's statements*.

Computer programming consists of two interconnected contexts of high level, abstracted and language-independent algorithmic thinking, and detailed, stepwised, and language-dependent coding. Our findings suggest that some of the programming misconceptions at the abstract level might be filtered by specifications of programming languages and accordingly not be visible for language-based misconceptions detecting approaches. On the basis of our findings, it can be concluded the flowcharts, by inheriting and bridging between the essence and key characteristics of algorithmic thinking and coding contexts, are useful means to detect misconceptions exhibited by students in the solution formulation and algorithmic thinking phases.

As observed in the results section, the students exhibited various sort of plan composition and construct-based problems on basic algorithm concepts in their flowcharts. The inverted assignment was the most occurring misconception encountered by the students. The same finding has been reported by other researchers (see for example [8,17]). Furthermore, the students showed several misconceptions related to the sub-algorithm concept including the composition of an algorithm using sub-algorithms. Arguably, a reason for this misconception stems from this fact that the majority of the students did not have any previous experience of working with abstract constructs such as sub-algorithms. Surprisingly, it has been observed that some of the students first wrote computer programs and then tried to convert their programs to flowcharts.

Given the programming misconceptions presented in Tables 1 and 2 as the analytical framework, the results suggest that there are many overlaps between misconceptions encountered by students in the algorithmic and coding parts of the programming process. For example, human interpretation, previous plan experience, unexpected case, and inverted assignment misconceptions happen similarly in both parts [18]. However, there are two exceptions with regard to the *parallel execution* and *dense step* misconceptions. These misconceptions represent some of the students' assumption that flowchart's steps can run parallel to each other. To the best of our knowledge, these are newly identified misconceptions and have not been reported in other studies. A possible reason for this neglect might be due to this fact that the majority of research on programming misconceptions has been conducted in a specific programming language and the syntactical structure and specifications of the programming language filter emerging this sort of abstract misconceptions.

The use of flowcharts generated by groups of students as data, instead of individually generated flowcharts, can be seen as a limitation of the study. One might claim that group dynamics might amplify or filter misconceptions arising at an individual level. A second limitation stems from conducting the study with students who had previous experience of programming. Repeating the same study by students with less programming experience might result in detection of other kinds of misconceptions. Other possibilities for follow-up research include using the identified misconceptions to develop CS teaching materials for the algorithmic thinking and examining their effectiveness in resolving these misconceptions, and scrutinizing possible links between students' exhibited misconceptions in algorithmic thinking and coding phases of programming. As to the bigger research project on design-based education, our results suggest that using flowcharts to express intermediate design products could provide a more *authentic* way to incorporate formative assessment of fundamental concepts during students' project work.

References

1. Barendsen, E., Grgurina, N., Tolboom, J.: A new informatics curriculum for secondary education in the Netherlands. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 105–117. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_9
2. Bohl, M.: Flowcharting Techniques. Science Research Associates, Chicago (1971)
3. Chetty, J., van der Westhuizen, D.: Towards a pedagogical design for teaching novice programmers: design-based research as an empirical determinant for success. In: Proceedings of the 15th Koli Calling Conference on Computing Education Research, pp. 5–12. ACM (2015)
4. Clancy, M.: Misconceptions and attitudes that interfere with learning to program. Comput. Sci. Educ. Res., pp. 85–100 (2004)
5. Elo, S., Kyngäs, H.: The qualitative content analysis process. J. Adv. Nurs. **62**(1), 107–115 (2008)
6. Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., Settle, A.: Computational thinking in k-9 education. In: Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference, pp. 1–29. ACM (2014)
7. Nuffic: The Dutch education system described (2015). https://www.nuffic.nl/en/publications/find-a-publication/education-system-the-netherlands.pdf. Accessed Sept 2017
8. Putnam, R.T., Sleeman, D., Baxter, J.A., Kuspa, L.K.: A summary of misconceptions of high school basic programmers. J. Educ. Comput. Res. **2**(4), 459–472 (1986)
9. Rahimi, E., Barendsen, E., Henze, I.: Typifying informatics teachers' PCK of designing digital artefacts in dutch upper secondary education. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 65–77. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_6
10. Rahimi, E., van den Berg, J., Veen, W.: Investigating teachers' perception about the educational benefits of Web 2.0 personal learning environments. eLearning Pap. (35) (2013)
11. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: a review and discussion. Comput. Sci. Educ. **13**(2), 137–172 (2003)
12. Scanlan, D.A.: Structured flowcharts outperform pseudocode: an experimental comparison. IEEE Softw. **6**(5), 28–36 (1989)
13. Seppälä, O., Malmi, L., Korhonen, A.: Observations on student misconceptions–a case study of the build-heap algorithm. Comput. Sci. Educ. **16**(3), 241–255 (2006)

14. Sheard, J., Simon, S., Hamilton, M., Lönnberg, J.: Analysis of research into the teaching and learning of programming. In: Proceedings of the Fifth International Workshop on Computing Education Research Workshop, pp. 93–104. ACM (2009)
15. Shneiderman, B., Mayer, R., McKay, D., Heller, P.: Experimental investigations of the utility of detailed flowcharts in programming. Commun. ACM **20**(6), 373–381 (1977)
16. Shulman, L.S.: Those who understand: knowledge growth in teaching. Educ. Researcher **15**(2), 4–14 (1986)
17. Sirkiä, T., Sorva, J.: Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In: Proceedings of the 12th Koli Calling International Conference on Computing Education Research, pp. 19–28. ACM (2012)
18. Smetsers-Weeda, R.: Think... then act() flowcharts as a tool for novice programmers to enhance their problem solving skills. Master's thesis, TU Delft, The Netherlands (2016)
19. Sorva, J.: Notional machines and introductory programming education. ACM Trans. Comput. Educ. **13**(2), 8 (2013)
20. Spohrer, J.C., Soloway, E.: Novice mistakes: are the folk wisdoms correct? Commun. ACM **29**(7), 624–632 (1986)
21. Wing, J.M.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)

A Preliminary Investigation on Computational Abilities in Secondary School

Ugo Solitro[1](✉), Margherita Pasini[2], Debora De Gradi[1], and Margherita Brondino[2]

[1] Department of Computer Science, Università Degli Studi di Verona, Verona, Italy
ugo.solitro@univr.it

[2] Department of Human Sciences, Università degli Studi di Verona, Verona, Italy
{margherita.pasini,margherita.brondino}@univr.it

Abstract. For many years, several countries have been committed to the introduction of computer science and, more recently, computational thinking in primary and secondary schools. Nevertheless, students of the first year of university often encounter difficulties in dealing with introductory courses in computer science and, in particular, programming. One of the main issues is related to the difficulty of thinking about algorithmic solutions, a skill that should be acquired from secondary school. A good knowledge of difficulties and the discovery of efficient strategies to overcome them are vital tasks. In this work, we discuss a first step in this direction. We have prepared a brief test focused on the algorithmic abilities and other processes potentially involved with it. The idea is to build a useful tool that can give assistance to the introduction of computational thinking skills in secondary school education. The test has been proposed to a small sample of secondary school students, aged 11 to 16 and its outcomes analyzed.

Keywords: Computational thinking · Algorithmic thinking · Problem solving · Secondary school · Computing education

1 Introduction

When we observe the performance of a first year introductory course in programming in a non vocational curriculum, we can be really disappointed: difficulties in learning, low performances, high drop-out. One of the most relevant problems is seemingly originated from too little familiarity with algorithm and effective procedures in the previous school experience. Many institutions in the past decades have proposed and developed experimental projects for the introduction of informatics and digital technology in School. This subject is also widely discussed in the community (see for instance [8,18,19]). More recently, informatics and, in particular, computational thinking have been incorporated into school curricula starting from primary schools. In spite of this, the final outcomes are often poor. Schools often do not have access to resources, notably people with adequate experience but also time, computer facilities.

Italy is engaged in a long process of reforming the School. This process involves in particular the introduction of informatics in all degrees, with an emphasis on *digital competences* instead of wide and cross-disciplinary computational skills. An account of this can be found in [8]. In 2015, more recent reform (also known as *"La Buona Scuola"*) has indicated *Computational Thinking* as one of priority objectives of the Italian school.

This work is part of a wider project aimed to study learning skills and difficulties encountered by students - mainly secondary school and university students - related to computational and algorithmic skills. This project also involves university students enrolled at the first degree in Applied Mathematics, and is oriented to drop out reduction. The project includes a deep examination of students' learning difficulties in programming. As a part of the activities, we also changed some teaching methodologies, analyzed motivation, evaluate performance and examined their background [11,12,16,17]. In particular, we discovered that the greater part of students lack of an adequate introduction to computational skills[1]. The project intends also to explore the modalities of introduction to computational thinking, actual problem solving methodologies and algorithmic skills, in middle and high schools. This study aims to present the results of the assessment of a small set of preliminary skills in secondary school students (grades 6 to 11).

This study presents the findings of the assessment of a small set of basic skills with the aim of developing instruments that could help the introductions of computational skills in the education and predicting difficulties.

2 Computational and Algorithmic Thinking

Computational Thinking (**CT**) is a permeating concept in the recent discussions about *Computer Science Education.* Papert in [10] suggests *"to use computational thinking to forge ideas that are at least as explicative as the Euclid- like constructions (and hopefully more so) but more accessible and more powerful."* This expression is more often related to the paper by Wing [20] which describes CT as a *"universally applicable attitude and skill set"* that includes solving problems, designing systems and understanding human behaviour using the fundamental concepts of computer science. On the other hand, Knuth [6] describes *Algorithmic Thinking* using a set of characterizing categories: representation of reality, reduction to simpler problems, abstract reasoning, information structures, algorithms. These two abilities are, in fact, strongly related and, in particular, CT requires the ability to identify appropriate algorithms.

In a recent review [14], Plerou shows the classification of algorithmic thinking difficulties and mathematical learning difficulties, grouping them into 6 categories: visual perception, spatial and geometrical perception, time perception, calculation, calculus perception, algorithmic thinking. Algorithmic thinking

[1] This datum is available by consulting both entrance test results and students' answers to a survey proposed at the beginning of the course.

deficits concern perception, design and description of an algorithm, best solution evaluation and the ability to separate a problem into smaller sub-problems.

In the last years, several studies investigated the skills used in programming. These studies used standardized tests to evaluate the required skills and students' results in programming courses or programming tasks to check for hypothesized verify the correlations.

Erdogan et al. [5] tested psychological predictors of 48 high school students. The predictors tested are creativity, problem solving, computer attitude and general skills, which include analytical thinking, abstract thinking and spatial perception. The results show a significant correlation between programming and mathematics grades, as well as between programming grades and general skills.

Milić [9] investigated on the *IQ* level, computer attitude and gender. This work studies the correlation with two types of programming tasks, a low level and a high level knowledge. The low level performance, which requires to perform a program, is correlated with IQ and computer attitude. The high level performance, which requires the ability to create a program, is correlated only with the low level performance and not with any of the other predictors tested.

Korkmaz's in [7] shows that critical thinking and logico-mathematical intelligence are both correlated with algorithmic skills, intended as problem analysis, solution design and mathematical transfer.

Ambrosio et al. [2] tested general intelligence, spatial reasoning, mathematical reasoning and attention to details. The results suggest that general intelligence and spatial reasoning are factors affecting the learning of programming, as these two factors are related to the final grades of the programming course.

3 Method

3.1 Participants

The data are collected among 261 students, of which 139 middle (i.e. lower secondary) school students (from 6 to 8 grades) and 122 high school students (form 9 to 11 grades).

Middle school students were from a residential district of the city. The school teaching plan prescribes 6 h of mathematics per week and no independent informatics teaching. The "digital" background is heterogeneous, with generally low CT skills. The sample is composed by 75 female students and 64 male students, distributed among the classes as shown in Table 1.

Table 1. Gender distribution among the middle school classes.

Class	1A	1B	1C	2A	2B	3A	3B
M	13	10	9	4	6	11	11
F	8	11	9	9	15	12	11

High school students were from a technical institute (Electronics, Informatics and Logistics) located in the city. The first two years are organized essentially in the same way for all the three curricula and the specialization starts in the third year. In Table 2 is synthetically described the group of participants with respect to sex and specialization. High school students generally show a heterogeneous "digital" that is dependent on middle schools, previous experiences and personal interests. In the first two years, they have to attend specific introductory courses in ICT. From the third year, informatics is considered more seriously as part of a professional training.

Table 2. High school sample data.

Class	Specialization	#	M	F	Courses
1	—	41	36	5	Mathematics (4) Computer technologies (1 + 2)
2	—	35	35	0	Mathematics (4) Applied science and technologies (3)
3	Informatics	24	24	0	Mathematics (3) Complements of mathematics (1) Telecommunication systems and technology (2 + 1) Systems and networks (2+2) Computer science (3 + 3) Telecommunication (1 + 2)
	Logistics	22	20	2	Mathematics (3) Complements of mathematics (1) Automation and electronic (1 + 2) Logistics (3 + 2)

In parenthesis the hours per week dedicated to the course:
(classroom) or *(classroom + laboratory)*.
#: number of students tested.
M: number of male students.
F: number of female students.

3.2 Procedure

The test consists of two different parts:

1. the first part is about the general abilities,
2. the second part regards computational and algorithmic skills.

Both parts were presented during normal class time, so the students were not required to dedicate extra time to the test. The activity was set following the need of the school and the availability of teachers, and hence not necessarily during mathematics or computer science classes. The test required about 50 min with an upper limit of a 1 h, depending on the school schedule and needs.

The first part of the test was performed in about 35–40 min. The students received all the questions and an answer sheet on which they had to report their answers and an identification number assigned randomly.

After the completion of the first part (all the answer sheets have been collected), the second part, consisting in a single sheet of paper, was hand out to students. In this case, the answers were marked directly on the paper, as well as the identification number. The second part of the test was performed in about 15–20 min.

3.3 Instruments

Bebras international contests have come out as a successful way to introduce computational thinking activities in School and *Bebras* [1] activities has been proposed for the introduction of informatics in the curriculum [3,4]. Hence the items in our test are partly based on *Bebras* tasks and partly on more traditional tests.

The First part of the test includes 18 items. The aim is to verify the students general abilities in 8 tasks that could be related to computational thinking. The items involve non verbal aspects, pattern recognition, spatial orientation, temporal order, procedure application, best solution identification, reasoning by analogies and problem comprehension. The items used are based on the *Bebras* tasks, the problem solving tasks and *Raven's Progressive Matrices* (RPM) [15]. RPM is a collection of widely-used standardized intelligence tests consisting of analogy problems in which a matrix of geometric figures is presented with one entry missing, and the correct missing entry must be selected from a set of answer choices.

The Second part acts as a criterion, useful to test the validity of the first part as a set of preliminary skills connected with programming skills. This second part is, in some way, estimates students' programming skills. It is organized in two "specific tasks":

– a *coding* task, again inspired by *Bebras* activities, where the student has to describe a simple procedure in a formalized language;
– an *algorithmic* task, based on the activities described in [13], where the student has find a solution to a problem in an effective way.

Since the algorithmic task is more challenging, it has been organized in subtasks guiding the student toward the solution.

4 Results

In this section we identify the tasks with an acronym that is connected with the specific goal:

Acronym	Description
R	*Non verbal (Raven matrix)*
S	*Pattern recognition*
O	*Spatial orientation*
ORD	*Sequential order*
P	*Procedure application*
SO	*Best solution identification*
A	*Reasoning by analogy*
CT	*Problem comprehension*
AL	*Problem solving*
COM	*Coding*

4.1 Correlations Between General Abilities and the Specific Tasks

To verify the effects of general abilities on the specific tasks, we study the correlation between scores in each general task and scores in the final tasks, algorithmic and coding.

The correlation matrix (Table 3) with Pearson's coefficients shows significant correlations between spatial orientation and coding ($r = .292, p < .001$), and temporal order and coding ($r = .299, p < .001$). For the algorithmic ability, significant correlations are shown with: spatial orientation ($r = .355, p < .001$), temporal order ($r = .342, p < .001$) and best solution recognition ($r = .355, p < .001$).

Table 3. Pearson's correlation matrix with significance level.

	COM	AL
R	0.15*	0.11
S	0.14*	0.24***
O	**0.29*****	**0.36*****
ORD	**0.30*****	**0.34*****
P	0.20**	0.21***
SO	0.20**	**0.36** ***
A	0.12	0.21***
CT	0.07	0.19**
COM		0.20**

$^{*}p < .05, ^{**}p < .01, ^{***}p < .001$

Then we tested two regression models to see which predictors better explain coding task and algorithmic ability. Spatial orientation ($\beta = .22, p < .001$) and temporal order ($\beta = .21, p < .001$) seemed to be the stronger predictors of coding task explaining 12% of variance. For algorithmic ability, spatial orientation

($\beta = .20, < .001$), best solution recognition ($\beta = .21, < .001$), temporal order ($\beta = .20, < .001$) and reasoning by analogies ($\beta = .13, p = .024$) explained 23% of variance.

4.2 Specific Abilities Interaction

Since for the coding task we have only three possible scores, that are just 0, $\frac{1}{2}$ or 1, the results obtained in the previous section about this task are not really reliable. We are therefore interested in running a one way *ANOVA* using as a fixed factor the coding score and as a dependent variable the algorithmic score, this analysis makes sense since we have mentioned that the computational ability includes the algorithmic ability. The ANOVA confirms our inclusion as it shows a significant main effect of the computational coding performance on the algorithmic score ($F(2, 258) = 5.25, p < .01$), in particular the *post hoc* tests show significant difference ($p_{bonf} < .01$) is found between the students who scored 0 in the coding task ($mean_{AL} = .33, st.dev._{AL} = .23$) and students who scored 1 in the coding task ($mean_{AL} = .49, st.dev._{AL} = .31$) (Table 4).

Table 4. Algorithmic task scores grouped by computational scores.

COM *score*	AL *mean*	AL *st.dev.*
0.0	0.33	0.23
0.5	0.40	0.26
1.0	0.49	0.31

4.3 School and Class Effects

School and class effects are analysed using a repeated measure *ANOVA* with 2 factors:

school (SCU): 1 – middle school, 2 – high school;
class (CL): 1 – first year, 2 – second year, 3 – third year.

Tasks main effect. The ANOVA shows there are main effects between the tasks ($F(1, 1785) = 53.12, p < .001$), meaning that at least one of the tasks has significantly lower scores than the others, independently of the school and class. Figure 1 shows that there are drops in the mean scores obtained in spatial orientation, temporal order and best solution recognition tasks.

School main effect. There is also a main effect of the school ($F(1, 255) = 76.27, p < .001$), meaning that there exists a significant difference between the global score obtained from middle school students and the global score obtained from high school students. A *post hoc* comparison confirms the significance of this difference ($p_{bonf} < .001$), and shows that middle school score ($mean = .65, st.dev. = .15$) is lower than high school score ($mean = .81, st.dev. = .14$).

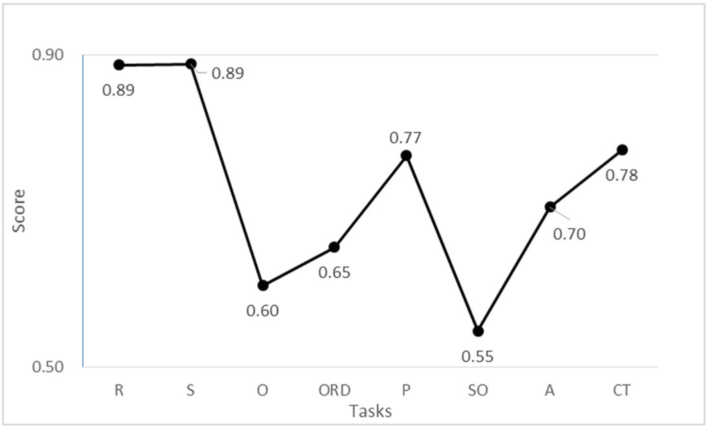


Fig. 1. Tasks main effect

Class main effect. The *ANOVA* also shows a main effect of the class ($F(2, 255) = 3.78, p < .05$) but this effect doesn't have a direct interpretation, as it considers first year students of both middle school and high school in the same group. For this reason we prefer to repeat the *ANOVA* on middle school data and high school data separated.

The results of the *ANOVA* on middle school data show there is no main effect of the class, so the differences between the mean scores of the different classes are not significant.

The results of the *ANOVA* on high school data, instead, confirms the main effect of the class found originally ($F(2, 119) = 3.02, p = .05$) and the *post hoc* comparisons show the only significant difference ($p_{bonf} = .05$) is between first year student scores ($mean = .78, st.dev. = .15$) and third year students scores ($mean = .85, st.dev. = .13$).

Tasks and school interaction effect. The interaction effect of tasks and school is significant ($F(7, 1785) = 3.07, p < .005$), meaning that there is at least one task where the difference between middle school and high school scores is higher than the differences in the other tasks. Figure 2 shows the task where the difference between middle school and high school scores is higher is spatial orientation (O).

Task and class interaction effect. The interaction effect of tasks and class is not significant, nor in the case we consider all of the data, nor in the case we consider middle school and high school data separately. This means that the trend of the classes among the different task is the same for all the classes, as we can see in Fig. 3.

School and class interaction effect. The interaction effect of school and class is again not significant, meaning that trend of the classes among middle school and high school are similar, as shown in Fig. 4.

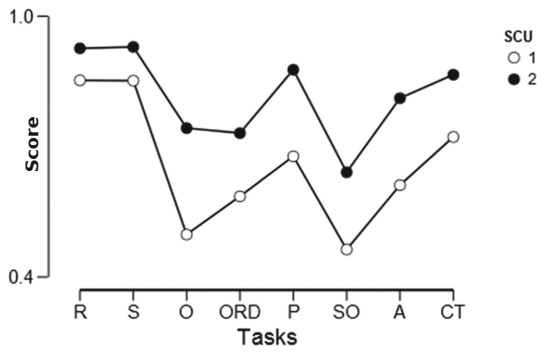


Fig. 2. Interaction effect of tasks and school. SCU: 1 = middle school; 2 = high school

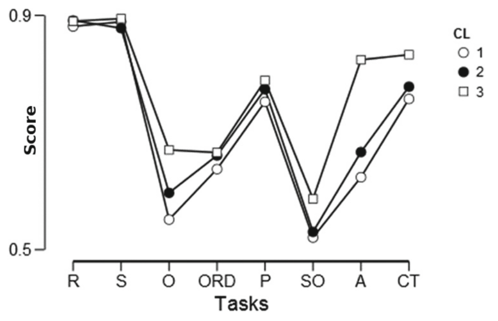


Fig. 3. Interaction effect of tasks and class. CL: 1 = first year; 2 = second year; 3 = third year

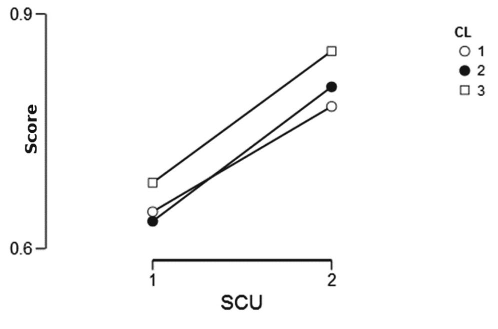


Fig. 4. Interaction effect of tasks and class. SCU: 1 = middle school; 2 = high school. CL: 1 = first year; 2 = second year; 3 = third year.

4.4 Gender Effect

It was possible to mark down the gender only for high school students, but the numbers show there are only 7 female students in the high school sample, so this number is not significant to make any analysis.

Although we don't know the individual gender for middle school students, we know the gender distribution among the classes, see Table 1. From this distribution, we can see that some classes have a balanced number of male and female students, while the class $1A$ has a higher number of female students and classes $2A$ and $2B$ have a higher number of male students. Since there is no significant difference between the classes in middle school, we could analyse our data considering the class $1A$ as a class with female dominance and the classes $2A$ and $2B$ as classes with male dominance, and see if there is a significant difference between classes with different gender dominance. The independent sample t-test shows there is no significant difference in any of the tasks, in fact the mean results of the first part of the test are very close (Female dominance: $mean = .63, st.dev. = .15$; Male dominance: $mean = .65, st.dev. = .17$).

5 Conclusion

The intention of this work was to establish a preliminary instrument that can help to introduce computational abilities in the secondary school.

The results show that the main abilities underlying computational thinking are temporal order, best solution identification, spatial orientation and reasoning by analogies. The school is a factor affecting students' performances so these abilities improve over time, especially after following specific courses, as in the case of third year high school students. The analysis on the gender suggests there is no difference between female and male performances, but this result is not significant as it was not possible to collect individual informations.

This preliminary work has brought some positive results, but has also highlighted the necessity of some improvements. Hence we have in mind to redesign the test to enhance its effectiveness and identify suitable action to improve the introduction of computational thinking in non vocational school curricula.

We also plan to work in cooperation with secondary school teachers in order to find an effective way to introduce computational thinking skills in secondary school. Moreover we intend to deepen our analysis of the effect of school background on the learning of programming and other computing subjects for non vocational university students.

References

1. (2017). www.bebras.org
2. Ambrosio, A.P., Almeida, L.S., Macedo, J., Franco, A.H.R.: Exploring core cognitive skills of computational thinking. In: Psychology of Programming Interest Group Annual Conference 2014 (PPIG 2014), pp. 25–34 (2014)
3. Dagienė, V., Futschek, G.: Bebras international contest on informatics and computer literacy: Criteria for good tasks. In: Informatics Education-supporting Computational Thinking, pp. 19–30 (2008)
4. Dagienė, V., Sentance, S.: It's computational thinking! bebras tasks in the curriculum. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 28–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_3

5. Erdogan, Y., Aydin, E., Kabaca, T.: Exploring the psychological predictors of programming achievement. J. Instr. Psychol. **35**(3) (2008)
6. Knuth, D.E.: Algorithmic thinking and mathematical thinking. Am. Math. Monthly **92**(3), 170–181 (1985)
7. Korkmaz, Ö.: The impact of critical thinking and logico-mathematical intelligence on algorithmic design skills. J. Educ. Comput. Res. **46**(2), 173–193 (2012)
8. Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., Settle, A.: Computational thinking in k-9 education. In: Proceedings of the Working Group Reports of the 2014 on Innovation and Technology in Computer Science Education Conference, pp. 1–29. ACM (2014)
9. Milić, J.: Predictors of success in solving programming tasks. Teach. Math. **22**, 25–31 (2009)
10. Papert, S.: An exploration in the space of mathematics educations. Int. J. Comput. Math. Learn. **1**(1), 95–123 (1996)
11. Pasini, M., Solitro, U., Brondino, M., Raccanello, D.: The challenge of learning to program: motivation and achievement emotions in an extreme apprenticeship experience. In: Proceedings of 27th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2016, pp. 151–156 (2016)
12. Pasini, M., Solitro, U., Brondino, M., Burro, R., Raccanello, D., Zorzi, M.: Psychology of programming: the role of creativity, empathy and systemizing. In: Vittorini, P., Gennari, R., Di Mascio, T., Rodríguez, S., De la Prieta, F., Ramos, C., Silveira, R.A. (eds.) MIS4TEL 2017. AISC, vol. 617, pp. 82–89. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60819-8_10
13. Plerou, A.: Dealing with Dyscalculia over time. Icicte **2008**, 1–12 (2014)
14. Plerou, A.: Algorithmic thinking and mathematical learning difficulties classification. Am. J. Appl. Psychol. **5**(5), 22 (2016)
15. Raven, J.: Raven progressive matrices. In: McCallum, R.S. (ed.) Handbook of Nonverbal Assessment, pp. 223–237. Springer, Boston (2003). https://doi.org/10.1007/978-1-4615-0153-4_11
16. Solitro, U., Zorzi, M., Pasini, M., Brondino, M.: Early training in programming: from high school to college. In: Gaggi, O., Manzoni, P., Palazzi, C., Bujari, A., Marquez-Barja, J.M. (eds.) GOODTECHS 2016. LNICSSITE, vol. 195, pp. 325–332. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61949-1_34
17. Solitro, U., Zorzi, M., Pasini, M., Brondino, M.: A "light" application of blended extreme apprenticeship in teaching programming to students of mathematics. Methodologies and Intelligent Systems for Technology Enhanced Learning. AISC, vol. 478, pp. 73–80. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40165-2_8
18. Sysło, M.M., Kwiatkowska, A.B.: Informatics for all high school students. In: Diethelm, I., Mittermeir, R.T. (eds.) ISSEP 2013. LNCS, vol. 7780, pp. 43–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36617-8_4
19. Webb, M., Davis, N., Bell, T., Katz, Y.J., Reynolds, N., Chambers, D.P., Sysło, M.M.: Computer science in k-12 school curricula of the 2lst century: why, what and when? Educ. Inform. Technol. **22**(2), 445–468 (2017)
20. Wing, J.M.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)

Solving Everyday Challenges in a Computational Way of Thinking

Bernhard Standl(✉)

Institute of Software Technology and Interactive Systems, TU Wien, Vienna, Austria
bernhard.standl@ifs.tuwien.ac.at

Abstract. Over a decade, computational thinking (CT) has been in the focus of educators and researchers in computer science. During this period of time, the term has been developed in different ways, reaching from a fundamental idea for finding a definition what problem-solving in computer science is about to a very particular view that CT is a required skill to code software applications. This paper presents results of the Fulbright project *coThink - Computational Thinking* carried out at the Missouri State University in Springfield, MO, USA which was based on the research question: How can CT be utilized with computer science algorithms for challenging real-life situations? As a result of a literature review, a CT five-step problem-solving process aimed at improving students' awareness to handle everyday life situations was identified. It was further integrated in classroom lessons, where it was applied at four student groups and evaluated mixing qualitative (analysis of worksheets) and quantitative methods (questionnaire) at a sample size of n = 75. Results showed that students frequently discovered a good approximation to solve real-life challenges following computer science algorithms but we also came to the conclusion to revise our problem-solving process.

Keywords: Computational-thinking · Problem-solving · Classroom research

1 Introduction

Seymour Papert initially introduced the term computational thinking (*CT*) in programming computers with LOGO for K-12 in 1980 in his book *Mindstorms* [11], but with problem-solving in mathematics in his mind. On the other hand Papert's problem-solving processes were influenced by Poyla's work from the 1950 s in *How to solve it* [12] who described how mathematical problems can be systematically approached and solved by students. The current movement in CT was initiated when Wing introduced the term again in 2006 [17]. Despite some similarities between Wing's and Papert's definition, Wing emphasized on problem-solving and Papert focussed on using CT to *forge ideas* [11]. Still, both didn't necessarily involve a computer in the process [6,9]. Considering this, we identified in recent literature, CT is for the most part either understood as umbrella term for learning coding concepts as e.g. loops, variables, recursions or as a term describing a problem-solving approach as used in computer science. Just as Nickerson et al. argued in [10]: *Definitions of CT vary*, there is, however, only little consensus in a definition for CT and so far a common understanding can only be identified in Wing's view [19]: *CT is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.* Barr and Stephenson argued in [2] referring to Wing [17], that CT includes seeking algorithmic approaches. This underlines that algorithmic thinking is part of CT and a term for composing different approaches, techniques, and systematics used in computer science for solving problems. Considering Lee, who argued in [8] that algorithmic thinking is a part of CT and further Barr, where CT is defined as the *seeking algorithmic approaches to problem domains* [2] in this paper we are focusing on the algorithmic part of CT. Even if the term CT is popular these days, so broad and deep it has been developed in history over decades. However, Tedre and Denning further described in [14] that some CT strategies can be applied also in different contexts, but they also claim, that *such transfer has never been substantiated.* Taking Yadav's thought, that CT can be integrated for everyday life challenges into account [21], this paper introduces an approach for combining computer science algorithms with real-life applications (as e.g. finding the shortest path to the school) within the theoretical framework of CT aimed at increasing the students' algorithmic problem-solving awareness by addressing these questions: *What is a suitable step-by-step computational-thinking problem-solving process for finding algorithms in real-life applications? How can such process put in practice classroom teaching at high-school?* The paper is structured as follows: First we are discussing related literature and propose a framework for a CT problem-solving processes for real-life applications, then we are presenting the integration of the process in classroom practice followed by an evaluation in practice and close with a conclusion and an outlook for further work.

2 Background

2.1 Algorithms for Everyday Life Applications

Algorithms in computer science have been developed for computers but they also can address human questions of everyday life. Our motivation for finding real-life applications is based on Wing's point of view, that techniques of computer science can support solving problems in other areas of life [17] where students identify learning as more authentic and has a strong relation to their own life. A further reason for connecting algorithms to human living was also, that this approach allows students to focus on the process of problem-solving beyond coding in a programming language. Inspired by popular ideas by Christian and Griffiths in *Algorithms to Live By* in [4] and scientifically confirmed in *Algorithms Unplugged* by Vöcking et al. [15] we selected five algorithms for an application in classroom practice: binary search, bin packing, minimal spanning tree, topological sort, and shortest path. Supposing, that the algorithms itself are known to the reader of this paper, we are presenting here the algorithms with only a brief description of the idea for an application with everyday life problems: **Binary Search:** When searching a CD or a book among many in a shelf, students develop a search strategy to find the item quickly. This leads to search algorithms and to the binary search algorithm in particular. **Bin Packing:** By asking how moving boxes can be filled up effectively with things from closets and shelves, students should find out strategies to master this systematically. The algorithms Next Fit and First Fit will describe the solution in computer science. **Minimal Spanning Tree:** The challenge is to connect seven islands with each other in building bridges. The distance between each bridge is known. Students have to find out the shortest possible connection to each island connecting the mainland. Solutions will lead to Prim's and Kruskal's algorithm. **Topological Sort:** Students write down a to-do list for a day and try to bring this list into order. By doing this, dependencies and preconditions will lead to the aimed algorithm for topological sorting that is used at finding one possible topological order. **Shortest Path:** Students work on a street map and try to identify the shortest path from home to school. The idea of selecting these algorithms was, to show in examples that solutions from complex problems in computer science can be easily transferred to challenge real-life situations as well.

2.2 Our View on Computational Thinking

Beyond all differences in definitions for CT, we identified across almost all contributions, that CT is about an approach for problem-solving in a way, that is used in computer science identified as algorithm. Wing already has stated in her refined definition of CT in [18], that CT is an approach for solving problems that draws on concepts fundamental to computing. Further, Aho described in [1] that the term CT includes *algorithm-design and problem-solving techniques that can be used to solve common problems arising in computing.* This is similar to Wolfram Math World's definition of the term *algorithm* itself as a specific set of instructions for carrying out a procedure or solving a problem. The question arises, if there is an additional value in introducing CT or if CT is just a vague and fashionable term paraphrasing procedures already included in the term *algorithm*? In fact, narrowing the term CT to designing algorithms would be off its underlying idea. Selby et al. explained for instance in [13] CT should include the ability to think in abstractions, in terms of decomposition, algorithmically, in terms of evaluation, and the ability to think in generalizations. Moreover, Xu et al. described in [20] CT as a universal, not limited to computers, fundamental, not only for experts, provides mental tools for thinking processes, has rich contents and is connected to any concept of computer science, and poses deep scientific problems in providing a framework to ask meaningful scientific questions. As Yadav et. al. reminded in [21,22] that Wing's initial paper [17] points out that CT involves three key elements *Algorithms, Abstraction, and Automation*, the term CT has been grown since then to a variety of interpretations. For

Lee et al. [8] CT involves defining, understanding, and abstraction. Barr et al. suggested in [2] CT involves the design of solutions, implementation of designs, testing, running analysing, reflecting, abstraction, creativity, and group problem solving. Grover and Pea [7] stated that CT should include among others abstraction, information processing, structured problem-solving decomposition as modularization, iterative recursive thinking, and efficiency. Again, for Lee et al. [8] CT involves defining, understanding, and solving problems, reasoning at multiple levels of abstraction, understanding and applying automation, and analysing the appropriateness of the abstractions made. In reference to Voogt et al. [16] who suggest, that *...we should not try to give an ultimate definition of CT, but rather try to find similarities and relationships in the discussions about CT,* outcomes of our literature review resulted in a problem-solving approach which goes beyond the skills computer literacy and understanding technology and underlines the need for developing CT skills [5]. Beyond the actual problem-solving process, Barr et al. suggested in [2] also a collection of dispositions a problem-solver should hold, which will be included in our process bellow. Those are confidence, persistence, handle ambiguity, deal with open-ended problems and communication skills in team-work. From teaching practice experiences, we know that the teacher's awareness of such dispositions at the students can be crucial for the success of learning and added them as additional part of the process.

2.3 Problem-Solving Approach for Real-Life Challenges

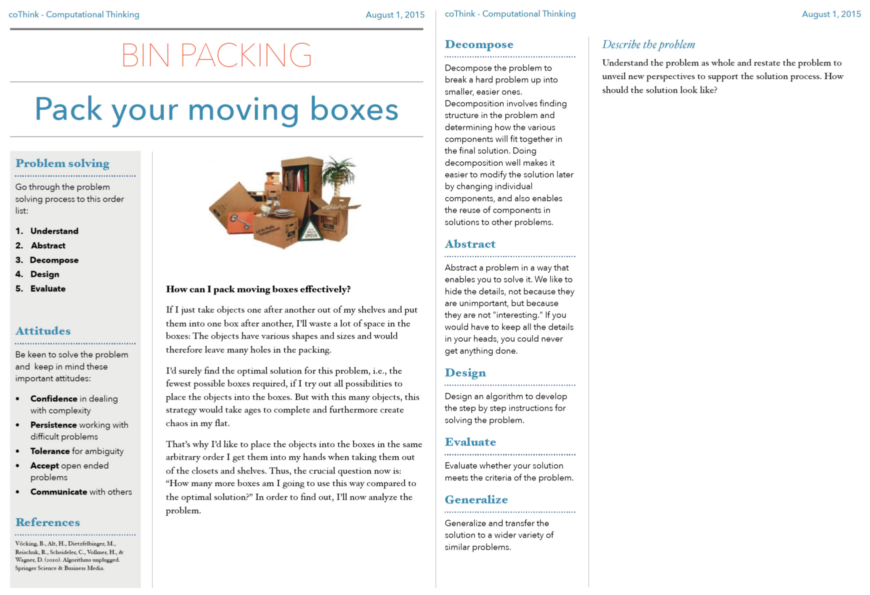
The goal of our CT problem-solving process is, to improve the students' awareness to be able to take advantage of CT problem-solving techniques to solve real-life problems. Therefore, we suggest an arranged step-by-step process for solving a problem. The process includes besides elements as abstraction, decomposition, and design of algorithm also understanding the problem at the beginning of the process and testing the solution at the end of it. **1: Understand** the problem as whole and restate the problem to unveil new perspectives to support the solution process. Also, state clearly what should be achieved with the solution. This part derives from Papert's initial ideas on CT [11], referencing to Poyla's first problem-solving step *First, you have to understand the problem* in *How to solve it* [12]. **2: Abstract** a problem in a way that helps to solve it. If we had to keep all the details in our heads, we could never get anything done. As we have described above, abstraction is mentioned as a keystone of CT and Grover et al. identified it as a core element, which differs CT from other types of thinking [7]. **3: Decompose** the problem to break a hard problem up into smaller, easier ones. Decomposition involves finding structure in the problem and determining how the various components will fit together in the final solution. Doing decomposition well makes it easier to modify the solution later by changing individual components, and also enables the reuse of components in solutions to other problems. This was already suggested by Wing in [17] as modularizing a problem *to make it tractable.* We see the parts decomposing and abstraction are key steps for preparing the actual problem-solving process. Abstraction is a step, where parts found will be removed or reduced in its representation and decomposing is a process, where the remaining parts will be separated and clustered. **4: Design** an algorithm to develop the step-by-step instructions for solving the problem. Start from what already is known and work outward from there. Make a plan how to approach to solve the problem. Selby et al. [13] relate to Wing [17] and further describe this process as a heuristic reasoning to devise a solution. **5: Test** the algorithm whether a solution meets the criteria as testing and debugging as used in software development and in the context of CT suggested by Brennan et al. in [3] to smoothen not working parts of the solution. This process has been planned by us so that it can be carried out in a linear fashion step by step for solving the problem. As mentioned above, we enhanced this five-step problem-solving approach, with these dispositions, as it was suggested by Barr et al. in [2]: **Confidence** in dealing with complexity, **Persistence** in working with difficult problems, **Tolerance** for ambiguity, The ability to deal with **open ended problems**, The ability to **communicate** and work with others to achieve a common goal or solution. Taking this approach for a CT problem solving approach into account, in the next section we will describe, how we integrated it into classroom practice.

3 Method

In order to proof the concept, we carried out our interventions at two high-schools with two classes each. The participants (n = 75) were students between the ages of 15 and 17 from four different high-school classes at two different schools (Greenwood Laboratory School and Willard High School located in Springfield, MO, USA). As the access to high school classes was limited to 2×2 h, we designed our lessons with a strong emphasis on your five-step problem-solving approach. Our goal was to find out, if everyday life problems can be approached systematically using techniques from computational thinking. Therefore, we planned four lessons on two days with the overall intention to let students find a solution to a problem heuristically followed by a reflection compared to the actual algorithm of computer-science. Students not only practiced their problem-solving competencies but also learned how computer science concepts can be useful and meaningful for their own life. We accompanied the classroom activities with mixed methods using a pre-/post-questionnaire and an analysis of the students' work sheets. Even though, both schools offer computer science classes with instruction on using a computer and standard software, students of both schools received no prior computer science education with a focus on computational algorithmic thinking on a regular basis. Hence, we presumed that all participating students had similar pre-experiences with algorithmic problem-solving processes. Due to this fact, we identified all students across both schools as one entity for analysing the data. It should also be mentioned that we did not include any control groups for organizational reasons and because of the research design, which was primarily aimed at evaluating our problem-solving concept. We carried out two separate lessons (2 h each) on two different days with each of the four student groups at two schools, the first lesson introduced CT and our problem-solving approach and in the second lesson students worked on problems individually and in groups collaboratively. In order to provide students a guideline through the problem-solving process, our worksheets were distributed with a clear description of the problem and a path through the process which was the same as stated above from understanding the problem for generalizing the solution.

3.1 Instruments

We designed two research instruments for evaluating our problem-solving process: in order to track the students' problem-solving strategies, we designed worksheets. The worksheets had five sections, divided into the five-step problem solving process. In Fig. 1, the first two pages of a worksheet to bin-packing are depicted. The front page describes the problem to the student and recalls the five-step problem-solving process and the supportive attitudes in the column on left hand side. On the right-hand side, an exemplary part of the work-sheets shows, that students also had to sketch for each problem-solving step their ideas as mentioned above: describe the problem, abstract the problem, decompose the problem, Design the algorithm, test the solution. In order to identify differences in students attitudes towards problem-solving we distributed pre-/post-questionnaire. The questionnaire included 20 items, where each of Barr's dispositions [2] has been assigned to four questions using a Likert-Scale from 1 'strongly disagree' to 6 'strongly agree'.



coThink - Computational Thinking August 1, 2015

BIN PACKING

Pack your moving boxes

Problem solving

Go through the problem solving process to this order list:

1. **Understand**
2. **Abstract**
3. **Decompose**
4. **Design**
5. **Evaluate**

Attitudes

Be keen to solve the problem and keep in mind these important attitudes:

- **Confidence** in dealing with complexity
- **Persistence** working with difficult problems
- **Tolerance** for ambiguity
- **Accept** open ended problems
- **Communicate** with others

References

Vöcking, B., Alt, H., Dietzfelbinger, M., Reischuk, R., Scheideler, C., Vollmer, H., & Wagner, D. (2010). Algorithms unplugged. Springer Science & Business Media.

How can I pack moving boxes effectively?

If I just take objects one after another out of my shelves and put them into one box after another, I'll waste a lot of space in the boxes: The objects have various shapes and sizes and would therefore leave many holes in the packing.

I'd surely find the optimal solution for this problem, i.e., the fewest possible boxes required, if I try out all possibilities to place the objects into the boxes. But with this many objects, this strategy would take ages to complete and furthermore create chaos in my flat.

That's why I'd like to place the objects into the boxes in the same arbitrary order I get them into my hands when taking them out of the closets and shelves. Thus, the crucial question now is: "How many more boxes am I going to use this way compared to the optimal solution?" In order to find out, I'll now analyze the problem.

coThink - Computational Thinking August 1, 2015

Decompose

Decompose the problem to break a hard problem up into smaller, easier ones. Decomposition involves finding structure in the problem and determining how the various components will fit together in the final solution. Doing decomposition well makes it easier to modify the solution later by changing individual components, and also enables the reuse of components in solutions to other problems.

Abstract

Abstract a problem in a way that enables you to solve it. We like to hide the details, not because they are unimportant, but because they are not "interesting." If you would have to keep all the details in your heads, you could never get anything done.

Design

Design an algorithm to develop the step by step instructions for solving the problem.

Evaluate

Evaluate whether your solution meets the criteria of the problem.

Generalize

Generalize and transfer the solution to a wider variety of similar problems.

Describe the problem

Understand the problem as whole and restate the problem to unveil new perspectives to support the solution process. How should the solution look like?

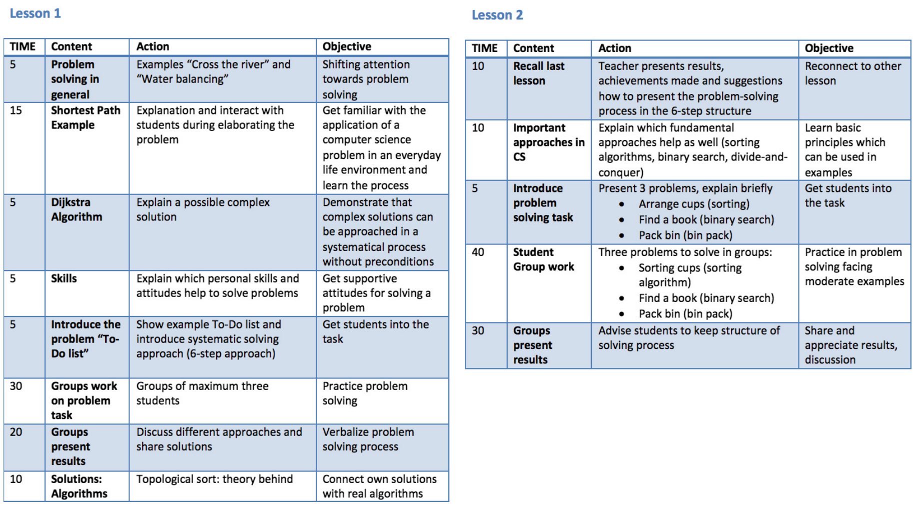
Fig. 1. Worksheet for the task "Pack your moving boxes"

3.2 Analysis

The analysis of the worksheets was a process of interpretation, where we evaluated our linear approach of solving the problem in going through each of the five problem-solving steps. We checked in which parts of the worksheets students were accepted as useful during the problem-solving process by reading through the worksheets and identified, which parts were not edited by the students where it was difficult for students to use it for the problem-solving process. The analysis of the questionnaire was carried out with descriptive statistics and statistical significance tests.

3.3 Classroom Lessons

The lesson plan in Fig. 2 gives an overview on the classroom intervention of day one and is based on findings of our literature research and includes all five steps from understanding the problem to generalizing the solution.



Lesson 1

TIME	Content	Action	Objective
5	**Problem solving in general**	Examples "Cross the river" and "Water balancing"	Shifting attention towards problem solving
15	**Shortest Path Example**	Explanation and interact with students during elaborating the problem	Get familiar with the application of a computer science problem in an everyday life environment and learn the process
5	**Dijkstra Algorithm**	Explain a possible complex solution	Demonstrate that complex solutions can be approached in a systematical process without preconditions
5	**Skills**	Explain which personal skills and attitudes help to solve problems	Get supportive attitudes for solving a problem
5	**Introduce the problem "To-Do list"**	Show example To-Do list and introduce systematic solving approach (6-step approach)	Get students into the task
30	**Groups work on problem task**	Groups of maximum three students	Practice problem solving
20	**Groups present results**	Discuss different approaches and share solutions	Verbalize problem solving process
10	**Solutions: Algorithms**	Topological sort: theory behind	Connect own solutions with real algorithms

Lesson 2

TIME	Content	Action	Objective
10	**Recall last lesson**	Teacher presents results, achievements made and suggestions how to present the problem-solving process in the 6-step structure	Reconnect to other lesson
10	**Important approaches in CS**	Explain which fundamental approaches help as well (sorting algorithms, binary search, divide-and-conquer)	Learn basic principles which can be used in examples
5	**Introduce problem solving task**	Present 3 problems, explain briefly • Arrange cups (sorting) • Find a book (binary search) • Pack bin (bin pack)	Get students into the task
40	**Student Group work**	Three problems to solve in groups: • Sorting cups (sorting algorithm) • Find a book (binary search) • Pack bin (bin pack)	Practice in problem solving facing moderate examples
30	**Groups present results**	Advise students to keep structure of solving process	Share and appreciate results, discussion

Fig. 2. Lesson plan day 1 and 2

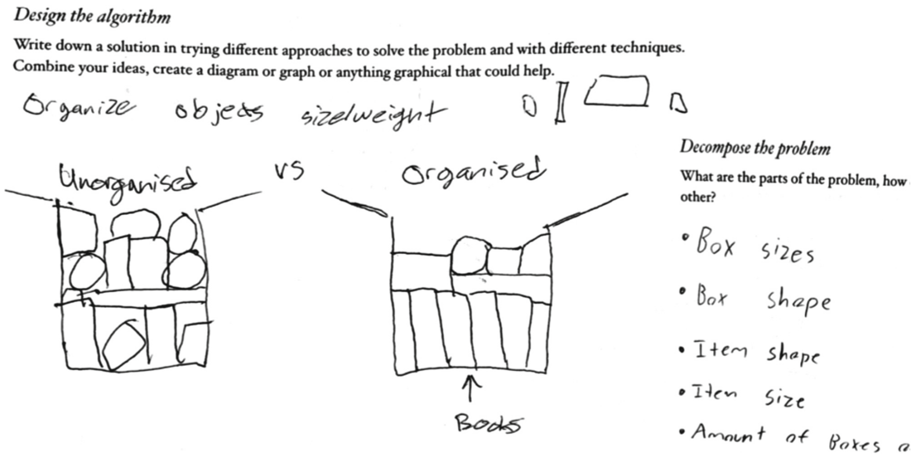
We started with an introductory lecture about problem-solving, CT and algorithms. Next the teacher approached a solution together with the whole student group for the shortest path problem by going through the five-step problem solving approach introduced above. Next, each student solved the example with the minimal spanning tree and shared the solution at the end of the hour together with others in group-work. It is important to note, that the students' attitudes required for a beneficial problem-solving process were mentioned explicitly by the teacher, who facilitated also the group-work for promoting the students' awareness for it. On the second day, the teacher first summarized insights from the first lesson and gave a lecture on some approaches, how computer scientists try to solve problems. This included basic sorting algorithms as well as general strategies as divide-and-conquer. Students subsequently worked in groups on more difficult problems as binary search, bin packing and the topological search as described above. Finally, group-work was followed with group presentations which included a discussion on connecting solutions with the *real* algorithms from computer science.

Example of application: In order to underline how our CT problem-solving process works for real-life challenges in detail, we give here an example based on the worksheet as depicted above. To each problem-solving section, we describe a possible student's answer which is based on Vocking et al. [15]. *Pack your moving boxes: How can I pack moving boxes effectively?* **Possible solution of a student:** By going through the problem-solving process, the students systematically approach the solution to the problem (very short version): **Describe the problem:** The challenge is to place my items into moving boxes in the most efficient way. **Abstract the problem:** To make it easier, I reduce the items to a geometrical representation that each part is described as block. So, a ball becomes for example a block with a edge length of its diameter. **Decompose the problem:** I have moving boxes with a certain size and items with different sizes and forms. **Design the algorithm:** First I tried different approaches in practice and extracted a general solution: 1. Sort the items by size, 2. Take the next available biggest item. 3. If the items fit into the box: Place the item into the box. Else: Open the next box. 4. Go to step 2. **Test the solution:** I organized a small shoe-box and some items and I evaluated my solution. It turned out, that my solution is not perfect yet as I need to many moving boxes. This example demonstrated, how the problem-solving process can be integrated for finding a solution to a problem from a real-life context by using a CT problem-solving approach leading to an algorithmic solution.

4 Results

4.1 Worksheet: Problem-Solving Process

Results showed, that students were successful in developing an approach to solutions for the problems. But they also experienced sometimes difficulties in dealing with the five-step process of solving a problem and were not always sure how each of the steps relates to the problem-solving process. For example, students were more engaged in describing the problem and designing the solution but they had difficulties to fill the worksheets for abstracting the problem, decomposing it and evaluating the solution separately. We found out, that the process of describing the problem already covered parts of the abstraction process. As in Fig. 3 it is shown, students have used different ways to solve the problem, some graphical, some using text. For instance, when students described the problem using their own words, some kind of abstraction has already taken place by most of the students. Next, the part of decomposing the problem in smaller parts apart from problem-solving was experienced as difficult and was rarely edited. Most of the time, decomposition was integrated into the design process or the evaluation process. These outcomes suggested us to reorder and reduce our problem-solving process at the worksheets in future to three steps: **1. Describe, decompose and abstract the problem**, **2. Design the algorithm**, and **3. Test the solution**.



Design the algorithm

Write down a solution in trying different approaches to solve the problem and with different techniques. Combine your ideas, create a diagram or graph or anything graphical that could help.

Organize objects size/weight

Decompose the problem

What are the parts of the problem, how other?

- Box sizes
- Box shape
- Item shape
- Item size
- Amount of Boxes a

Fig. 3. Snippets of filled student worksheets

4.2 Questionnaire: Students' Problem-Solving Attitudes

Even though, the structure of the worksheets was challenging for students, our intervention still had some impact on students' attitudes towards computational problem solving as the pre-/post-questionnaires suggest. The results of the questionnaires showed some differences at certain items, where some of them even differed statistically significant. The items of the diagram below can be clustered in four sections, where each cluster represents a field of dispositions a problem-solver should hold as defined above. These are **1–4:** Confidence in dealing with complexity, **5–8** Persistence in working with difficult problems, **9–12:** Tolerance for ambiguity, **13–16:** The ability to deal with open ended problems, **17–20:** The ability to communicate and work with others to achieve a common goal or solution. The questionnaire was identified as reliable as all two of the measures evidenced a good reliability as Cronbach's alpha ranged from 0.82 for the pre-test and 0.85 for post-test. As the results in Table 1 show, confidence (items 1–4) increased and item 1: "I feel anxious when I have to solve this kind of difficult problems." had also a significant difference in the pre-test scores (M = 3.55, SD = 1.51) and post-test scores (M = 4.20, SD = 1.26); t(140) = −2.77, p = 0.006. Furthermore we identified a significant difference in the pre-test scores (M = 3.55, SD = 1.42) and post-test scores (M = 4.20, SD = 1.12) at item 4: "I feel

Table 1. Aspects from students' attitudes towards problem solving. Scales from 1 'strongly disagree' to 6 'strongly agree' (n = 75)

	Item	∅ pre	∅ post
1	I feel anxious when I have to solve this kind of difficult problems	2.78	2.00
2	I doubt that I could solve this complex problem	2.26	1.73
3	I like to solve challenging problems like this one	3.74	3.55
4	I feel confident in solving this kind of difficult problems	3.70	3.91
5	If I could not solve this problem quickly, I would give up	2.30	2.18
6	I feel impatient when I'm working on this kind of challenging problems	2.87	3.09
7	When a problem like this challenges me, I am keen to solve it	4.17	4.09
8	I would work on this problem until I have found a solution	4.39	3.73
9	I rather prefer to work on problems where the solution is already obvious	3.04	3.09
10	There is possibly only one way to solve this problem	2.70	1.91
11	There are many roads that lead to the solution	4.13	4.64
12	This problem may has more than one solution	4.61	4.73
13	There exists a solution to this problem	5.04	5.55
14	I would be confused, if I wouldn't find a clear solution with this problem	3.78	3.82
15	There can be different approaches to solve this problem	4.78	5.00
16	The solution to this problem can look different from what I have expected	4.43	4.82
17	I don't think that someone could help me to solve this problem	2.13	2.27
18	If too many people are involved in solving this problem, it will not be done well	3.43	3.95
19	Teamwork is the key to solve this problem effectively	3.78	3.93
20	A solution to this problem could be better found together	4.26	3.27

confident in solving this kind of difficult problems.", conditions; t(140) = −3.00, p = 0.003. Items in the second cluster of persistence in the problem-solving process (items 5–8) decreased. The third cluster tolerance for ambiguity (items 9–12) increased and there was a significant difference in the pre-test scores (M = 4.06, SD = 1.14) and post-test scores (M = 4.74, SD = 0.89) at item 10: "There is possibly only one way to solve this problem.", conditions; t(140) = −3.59, p = 0.001. The fourth cluster on the ability to deal with open ended problems (items 13–16) increased and there was a significant difference in the pre-test scores (M = 4.94, SD = 1.51) and post-test scores (M = 5.34, SD = 1.26) at the variable "There exists a solution to this problem.", conditions; t(140) = −2.32, p = 0.022. The fifth cluster on teamwork (items 17–20) is inconsistent in terms of that students think that the solution to a problem can be found better together (item 20) but at the same time state that too many people involved in problem solving will have a negative impact (item 17). In a nutshell, results of the questionnaires suggest that students reduced their anxiety in solving difficult problems and increased their confidence. We further conclude, that students developed their understanding for the possibility of multiple solutions.

5 Discussion

This paper presented results of the project *coThink - Computational Thinking* which was aimed at identifying an approach for a step-by-step CT problem-solving process for real-life applications. The underlying idea was to break problem-solving techniques known from computer science down to support students' awareness for handling real-life challenges. Based on literature research a five-step CT problem-solving approach emerged which was enhanced with dispositions a problem-solver should hold as suggested by Barr et al. in [2]. We designed lessons and material which integrated our ideas and concepts and applied it in practice at four student groups at two different schools. First experiences showed, that students were keen to develop solutions for problems in a real-life context and their approach mostly approximates an algorithmic description or even sometimes an existing algorithm of computer science. Results of the questionnaires evaluating students' attitudes for problem-solving showed, that our intervention had some positive impact, even if it is likely that students didn't adopt their dispositions but rather had an increased awareness which dispositions are required for CT problem-solving. Further work will also include the refinement of new worksheets, the specification of further algorithmic real-life examples and its application in classrooms. It turned out, that students had problems filling all sections of the worksheets and finding a description for each of the five problem-solving steps was perceived as confusing. In fact, only the sections *Understand the problem* and *Design the solution* were accepted and helpful in the process. Some of the students described their solution, which was supposed to part of section *Abstract the problem*, as part of *Describe the problem*. As a consequence, we further redesigned the worksheets and combined fields which will lead to three sections of the worksheets, as described above (1. Describe, abstract and decompose the problem - 2. Design the algorithm - 3. Test the solution) In essence, we reduced the five-step problem-solving approach to three steps in order to simplify the process for students to: understand - solve - analyse. In the future, we will continue refining the teaching material and the research setting for a more detailed analysis focusing more on the actual students' problem-solving process by adding for instance think-aloud interviews as research instrument to gain more detailed insights how our approach has impact on CT problem-solving process.

Acknowledgments. This research was supported by the Fulbright Visiting Scholar Program Austria sponsored by the U.S. Department of State. I would like to thank the Fulbright Office in Vienna, the Council for International Exchange of Scholars (CIES) and the Institute of International Education (IIE). I thank my colleagues at Missouri State University in Springfield, MO, USA who provided assistance in facilitating the research project. I would also thank the school authorities and teachers at Greenwood Laboratory School and Willard High School for support and assistance in organization of classroom lessons and in submitting the IRB proposal.

References

1. Aho, A.V.: Computation and computational thinking. Comput. J. **55**(7), 832–835 (2012)
2. Barr, V., Stephenson, C.: Bringing computational thinking to K-12. ACM Inroads **2**(1), 48 (2011)

3. Brennan, K., Resnick, M.: New frameworks for studying and assessing the development of computational thinking, pp. 1–25 (2012)
4. Christian, B., Griffiths, T.: Algorithms to Live by: The Computer Science of Human Decisions. Henry Holt and Co., New York (2016)
5. Dagiene, V., Futschek, G.: Bebras, a contest to motivate students to study computer science and develop computational thinking. In: Proceedings of WCCE, pp. 139–141 (2013)
6. Dagienė, V., Sentance, S.: It's computational thinking! Bebras tasks in the curriculum. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 28–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_3
7. Grover, S., Pea, R.: Computational thinking in K12 a review of the state of the field. Educ. Researcher **42**(1), 38–43 (2013)
8. Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L.: Computational thinking for youth in practice. ACM Inroads **2**(1), 32 (2011)
9. Mannila, L., Dagiene, V., Mirolo, C., Settle, A.: Computational thinking in K-9 education, pp. 1–29 (2014)
10. Nickerson, H., Brand, C., Repenning, A.: Grounding computational thinking skill acquisition through contextualized instruction. In: Proceedings of ICER 2015, pp. 207–216 (2015)
11. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, Cambridge (1980)
12. Polya, G.: How to Solve It, 2nd edn. Penguin Books Ltd., London (1957)
13. Selby, C.C., Woollard, J.: Computational thinking: the developing definition, pp. 5–8 (2013)
14. Tedre, M., Denning, P.J.: The long quest for computational thinking. In: Koli Calling Conference on Computing Education Research, pp. 120–129 (2016)
15. Vöcking, B., Alt, H., Dietzfelbinger, M., Reischuk, R., Scheideler, C., Vollmer, H., Wagner, D. (eds.): Algorithms Unplugged. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15328-0
16. Voogt, J., Fisser, P., Good, J., et al.: Computational thinking in compulsory education: Towards an agenda for research and practice. Educ. Inform. Technol. **20**, 715–728 (2015). https://doi.org/10.1007/s10639-015-9412-6
17. Wing, J.M.: Computational thinking. Comm. ACM **49**(3), 33 (2006)
18. Wing, J.M.: Computational thinking and thinking about computing. Philos. Trans. Ser. A Math. Phys. Eng. Sci. **366**(1881), 3717–3725 (2008)
19. Wing, J.M.: Computational Thinking: What and Why? The Link - The Magazine of the Carnegie Mellon University School of Computer Science (2011)
20. Xu, Z.W., Tu, D.D.: Three new concepts of future computer science. J. Comput. Sci. Technol. **26**(4), 616–624 (2011)
21. Yadav, A., Hong, H., Stephenson, C.: Computational thinking for all: pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. TechTrends, 1–4 (2016)
22. Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., Korb, J.T.: Introducing computational thinking in education courses. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, pp. 465–470. ACM (2011)

System Papers

Teaching Basic Elements of OOP in School Informatics During Constructing Virtual Micro-worlds

Evgeny A. Eremin(✉)

Perm State Humanitarian Pedagogical University, Perm, Russian Federation
eremin@pspu.ru

Abstract. Object-oriented programming (OOP) is one of the most prevalent techniques of programming now. OO principles are also widely used in comprehensive software including user graphical interface. Although the benefit of early teaching OO concepts is advisable, it meets variety of difficulties and as a result no general consensus on this subject exists. The present paper by virtue of OOP instruction analysis proposes to include OO foundations as a topic into the school informatics course. The specific of the suggested method lies in a separation of OO principles from learning any concrete programming language like Java or C#: the original software tool was developed for these aims by the author. The tool acquaints pupils with OO concepts during construction from objects their own virtual world.

Keywords: OOP · Objects · School · Informatics · Courseware · Micro-worlds

1 Introduction

At present an object-oriented approach became the prevalent paradigm in the professional world of software development. Programming community recognized the value of OOP for designing complex systems and now practically all modern programming languages include support of this technology. Objects are widely used not only for programming, but also for internal organization and interfaces in many software systems. For instance [16] OO approach helps to understand the notion of process in a multitasking system, the message exchange mechanism and even user manipulations with the standard elements of graphic interface. The ideas similar to OO inheritance can be found in artificial intelligence, where hierarchical structures represent human knowledge. The architecture of modern computers is formed from modules with very multifarious internal organization, connected by a common interface. Besides in education we can see that computer-based courses and courseware are often constructed from modular objects [26].

Such wide application of OO ideas in industry, computer science and engineering education requires its wide learning, but this demand comes across many difficulties. First of all OO approach is based on a high-level abstraction, so understanding of this subject is a serious cognitive problem [9, 10, 23, 24, 29]. Furthermore it is usually

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 195–206, 2017.
https://doi.org/10.1007/978-3-319-71483-7_16

connected with a simultaneous learning of programming language constructions. Some other plural subjective factors also complicate OOP teaching [13], for example, an absence of suitable well-developed teaching materials and software tools. After an extensive survey of literature on teaching OOP in [20] the authors also made a conclusion about a lack of provable pedagogical approaches, appropriate books and suitable environments for novices. Although many ideas exist how OOP should be taught to beginners, there is no general consensus on this subject.

This paper proposes to begin learning the fundamental ideas of OO approach in a school informatics course. A new instructional method for these aims is described, which helps to form the main OO concepts in pupil's mind during building some world on a computer display. Software support for this learning activity was written by the author. For those who wish to create more advanced versions of computer support the paper contains a description of the software system organization; using it, a programmer can develop more perfect educational software.

2 Background

"It is clear that novice programmers face a very difficult task. Learning to program involves acquiring complex new knowledge and related strategies and practical skills" [22]. Learning OOP is much more complicated goal [23]. So in this section we discuss the difficulties of teaching OOP and the ideas how to overcome them. As the number of publications about OOP is enormous, only the most related ones are mentioned in the review.

2.1 Teaching Objects First

An experience of teaching OOP immediately showed that it has specific whether student has got preliminary knowledge in "classic" (imperative) programming or not. The phenomenon when learners meet difficulties, studying OOP after other non-OOP languages, got a special name "*paradigm shift*". To avoid this obstacle, a teaching technology called "*objects-first*" was proposed. As follow from the name, it suggests introducing OOP principles from the very beginning. So students start from objects and inheritance, and after experimenting with these ideas only then "goes on to introduce more traditional control structures, but always in the context of an overarching focus on object-oriented design" [1].

Situation with this educational technology is not fully clear yet. An all-around discussion about "objects-first" and "objects-late" technologies ("object vs. imperative") had not leaded to unity and consensus [14]. Psychological studies of a problem are rare and non-systematic, as review [23] argues. Furthermore the experimental research also brings dissonant results. For example publication [7] states the following. "The main result of the study is that there are no differences between OOP-first and OOP-later with regard to learning gain." Conversely, research in [28] leads to the positive conclusion: "it is possible to say that the learners instructed with object-first method achieved higher learning outcomes."

It is worth mentioning that experimental studies clearly show the difference in understanding of the program texts [21]. “Novice programmers comprehending an OO style program form a strong domain model, while novices comprehending an imperative style program form a strong program model”. More detailed review of this subject can be found in [22]. From our point of view this thesis confirms that OOP understanding has higher level and so it is one more argument to learn it.

2.2 Avoiding Misconceptions of Terms

Another problem of OOP teaching is possible misconceptions about the fundamental terms. A large set of such confusions were revealed after the analysis of the distance learning experience in the paper [9]. As it turned out, students often mix even two basic concepts – “object” and its “class”. Later publications [20, 24, 29] confirmed this misconception.

What pedagogical conclusion follows from this surely established fact? The fundamental OOP concepts that do not depend upon any programming language must be taught very carefully. Probably it is rational to learn them *apart* from a real programming language.

2.3 Language Choice; Micro-worlds

Teachers very often debate the topic what language is better to instruct programming. Researchers also persecute many studies making the detailed comparison of different programming languages [15, 17]. A discussion about OOP language for the beginners even more actual because the novices “seem to fixate on language details, losing the big picture, no matter how much we stress all of these concepts” [24].

It is evident that a language for the novices must be laconic, with simple rules and without unnecessary facilities. Such *mini-languages* [5] are often specially created for learning of the programming foundations. “The time required to master a mini-language itself is small, so the students can spend most of their efforts on more important issues” [5].

Except simplicity, educational mini-languages have another essential feature: they are always built on the base of some virtual environs, so a student’s program controls certain object (*actor*). Such teaching environments are usually called *micro-worlds* [1, 5, 17, 27]. Hence using a micro-world in instruction has two essential advantages: the motivation of pupils (introducing elements of game) and the reduction in complexity of a programming language and a development tool.

There are several well-known micro-worlds to study OO approach, for instance Alice [6], Greenfoot [11] and ObjectKarel [30], built on a base of Karel++ [3]. All of them are successfully using as an introduction into one of the most popular OOP programming language Java.

2.4 Visualization

Changes in a micro-world on a computer screen demonstrate the process of a program execution, so it clearly visualizes what program a student has written and how accurate it is done. Modern professional programming environments include well-developed tools for visualization. As it is argued in [12], for introductory courses the environment is possibly more important than the language, because if it is too complex, pupils may loose a sense of learning material, struggling with powerful, but unclear interface.

We must carefully examine what kind of visualization our students need. "The most advanced professional object-oriented development environments, such as Visual C++ or Delphi, use graphical support only to build the user interface of an application, but neglect the internal structure of the program itself" [12]. Besides educational software for learning OOP need appropriate visualization of the basic OO concepts.

For further consideration it is important not to miss one more positive feature of the present-day visual software systems: they allow us to get rid of writing numerous formal declarations. When, using a mouse, you create a new component in programming environment like Delphi, the system automatically generates necessary description and places it into a program text. The present paper expands this pupil friendly idea: it suggests hiding all formal descriptions inside a teaching system and showing them only in the demonstrative visual forms like tree of classes or tables of properties.

2.5 Application of General Didactic Principles

Lastly let us look at a problem of OOP teaching from the general pedagogical positions. From this point of view we have a complex learning material that is difficult to teach. What theory of didactics offers us for these aims? We can use so called "*spiral curriculum*". In his well-known book [4] Bruner wrote about studying the scientific foundations at school the following way. "Experience… points to the fact that our school may be wasting precious years by postponing the teaching of many important subjects on the round that they are too difficult." And later: "the early teaching of science … should be designed to teach these subjects with scrupulous intellectual honesty, but with an emphasis upon the intuitive grasp of ideas and upon the use of these basic ideas. A curriculum as it develops should revisit these basic ideas repeatedly, building upon them until the student has grasped the full formal apparatus that goes with them." As we see from discussed above, OOP, being an approach that is built on several non-language key concepts, is a good candidate to such scheme of learning.

Bergin in his project "Fourteen Pedagogical Patterns" [2] declared similar ideas about building an educational course. "Organize the course so that the most important topics are taught first. Teach the most important material, the "big ideas," first (and often). When this seems impossible, teach the most important material as early as possible" [2]. The argument for such conclusion is that if you delay the learning of important topics students may get many misconceptions (see segment 2.2 about problems with fundamental OOP terms).

Applying these general principles to introducing OOP into a school course of informatics, we can conclude that the starting point for it is not the constructions of some popular programming language, but understanding what an object, its properties and behavior is. Supported with a suit of exercises aimed on manipulation with objects of some micro-world, such lessons will create the solid platform for future programming.

3 Teaching Strategy

The analysis in the previous section showed that for many reasons it is practical to begin an introduction into OO approach from teaching several fundamental categories like object, class, inheritance etc. This part of material has no direct connection with any OOP language. If we support it by some OO micro-world, we can get a complete topic to learn within a framework of school informatics.

To determine the content of a learning material for this topic we may follow the publication [31], based on authors' programming and pedagogical experience, where the six-step approach to teach the OO methodology is presented. First five steps are the following (the sixth one is optional and does not take out below).

1. Discuss fundamental principles of object-orientation with respect to conventional thinking.
2. Introduce an object concept by observing the real world.
3. Acquire the class concept by abstraction of many common objects.
4. Introduce instantiation after the class concept is learned.
5. Illustrate subclasses by adding more details to an existing class and superclasses by finding common things among several classes.

First two steps are general and so should be discussed with the help of illustrations from everyday life. For steps 3–5 the authors suggested to show students (as an example) how correspondent declarations look in C++ language. But an example set is not mandatory, so we can consider that *all* five steps do not depend upon specific computer language. Temporarily compromising learning goals and working on them in isolation is the main way to reduce cognitive load [25].

"After these steps, instructors can start the language teaching, and should be confident that the students can learn the language with correct thinking and with object-oriented methodology in mind" [31].

Similar strategy we find in [19] before an introductory course of Java language. It is reported shorter, but the content is obviously similar. The first lectures explain the object ideas using real world examples to form "mental models" of classes and objects. Theoretical part are followed by several specially constructed for the course computer labs.

In recently printed textbook [18] such way to start OOP is described the following way. "For some time you will not feel that you are learning programming. You will have a feeling that we are only playing with the computer. However, in this initial phase you have to learn a lot of terms, connections and regularities so that later on you could understand further topics much better and quicker."

So we see that an easy method to introduce OOP basics to novices is possible and its content is clear. Now we shall describe its software support, proposed in the present paper.

4 Software Support

To support the theoretical explanations what is an object and how to use it, we should have some learning software to organize labs. In this section a variant of such software, proposed by the author, will be briefly described. It was named *System Builder*

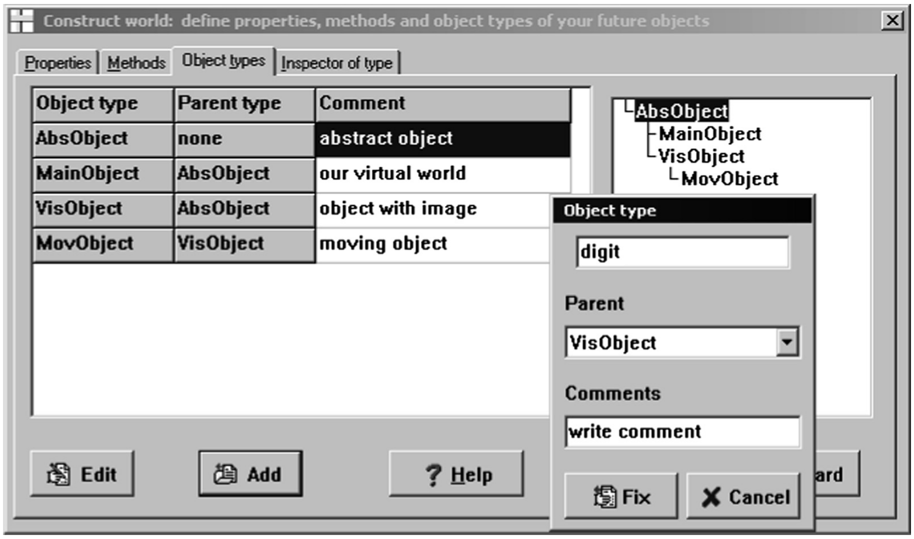
(SB) – a tool to construct a virtual micro-world from objects. First version of this program was presented earlier [8], later on it was extended and improved after testing with the students of pedagogical university (recent pupils and future school teachers).

4.1 Tool's Peculiarity

As it was mention above, the most known micro-worlds are aimed at further learning Java. Unlike them, SB is separated from concrete programming language and is focused on the fundamental OO concept themselves. It is rather a tool for object design, which teaches pupils to see objects with their relations in real life and build an OO model of a situation. During manipulations with created objects (mainly with visual images) pupils can deeper understand what is property, method, inheritance, polymorphism etc. On our opinion, such approach is right for school introductory topic.

Extending ideas of modern visual programming environments, SB has no any descriptions of classes and objects at all. For instance to create a new type of objects (class) you just fill an "electronic blank": input its name into a text field, select a parent class from a list and enter optional comment for yourself (Fig. 1). So we just give answers to short clear questions instead of writing long statements, constantly remembering rules and agreements of language. The system stores all input data into memory and represents it for learners in a form of tables. Table representation, as it turned out, is not only a demonstrative form of output, but also a perfect instrumentation for compilation of a project.

According to its pedagogical aims SB has the easiest language to describe the behavior of objects. It contains simple assessment (e.g. *x.image* = *x.image* + *1*) and calling of object's method (*x._changeimage*). A conventional form of *if*-operator (with possible nesting) is defined. In addition several optional constructions exist (e.g. a cycle


Construct world: define properties, methods and object types of your future objects
Properties
Methods
Object types
Inspector of type
Edit
Add
? Help
ard
Fix
Cancel

Object type	Parent type	Comment
AbsObject	none	abstract object
MainObject	AbsObject	our virtual world
VisObject	AbsObject	object with image
MovObject	VisObject	moving object

```
└AbsObject
  ├MainObject
  └VisObject
     └MovObject
```

Object type

digit

Parent

VisObject

Comments

write comment

Fig. 1. Creating a new type of objects (class) from one of the standard types

for sequential substitution of objects with specified type in order, taken from correspondent table). All these operators are used to write methods and method's body (without any formal title) is the longest text student must type. Minimal text with primary intuitive syntax, fully free of descriptive statements from a concrete professional language (like import, void, static, public etc.), is an essential advantage of the proposed teaching tool.

For maximum reducing of input, SB has predefined library of types, properties and methods. It is read from a system text file when SB starts. For instance, there is a primary hierarchy of objects in this library.

All object types finally are derived from the top parent type called AbsObject (abstract object). Its descendants are MainObject with predefined object World and Vis-Object, which describes a static (motionless) object with image. The last one's child named MovObject in addition can move (see hierarchy in Fig. 1). So a pupil can instantly start to create visual static or moving objects.

4.2 How to Build a Virtual World

After carefully planning of their world, students run SB and do the following steps.

1. **Construct** the world: define object types, methods and properties of future objects. If the standard template is not enough, add new types, properties and methods to it. Try to prepare the maximum, although SB allows correction at any moment. Note that as practice showed just this step often puzzles students unlike two further ones.
2. **Build** the word: using what was prepared on the previous step, create objects and set properties to each of them. On this step a pupil already sees where objects are situated and what images (static yet) they have.
3. **Watch** what was built. At last watch the evolution of world in time. If there are no mistakes, enjoy how your ideas become true. If it is not so – try to find mistakes. SB has special regimes to facilitate this process. Except automatic regime, there are two debugging ones: by time steps and by operations.

4.3 Examples of Projects

Primarily let us briefly discuss as an example how to build in SB a simple project named *Timer* (that shows minutes and seconds). Its view you can find in the top right corner of the collage compounded in Fig. 2.

The sequence of operations may be the following.

1. First of all draw (or take from a clipart) 10 pictures with images of numerals, name these files 'pic1' – 'pic10' and put them into a project folder.
2. Create a new type (called for example *digit*) from *VisObject* standard class and add the only property *max_value* to it (see step 6 also). Let us accent, that timer digits have no special property to keep current numbers – their images do this.
3. Add property *carry* to the standard object *World*. When any digit reaches *max_-value*, it must be forced to 0; in this case *carry* = 1, otherwise *carry* = 0. (We can see the simplest variant of data exchange between objects in this project.)

4. Find in the standard type *MainObject* method *_main* and complete it with the text from listing 1.
5. Add method *_inc* to *digit* type and input its program from the same listing. (Note that this is the longest text to input.) Now our world is **constructed**.
6. Then create 4 objects of *digit* type (4 timer digits): *sec_lo*, *sec_hi*, *min_lo* and *min_hi*. Postfixes *_lo* and *_hi* let us distinguish the low and the high digits in minutes and seconds values. For both high digits *max_value* property must be set to 5 and for the low ones – to 9. Now all preparations are over, the world is **built**,
7. Run SB project and **watch** how timer becomes alive. Correct errors (if any).

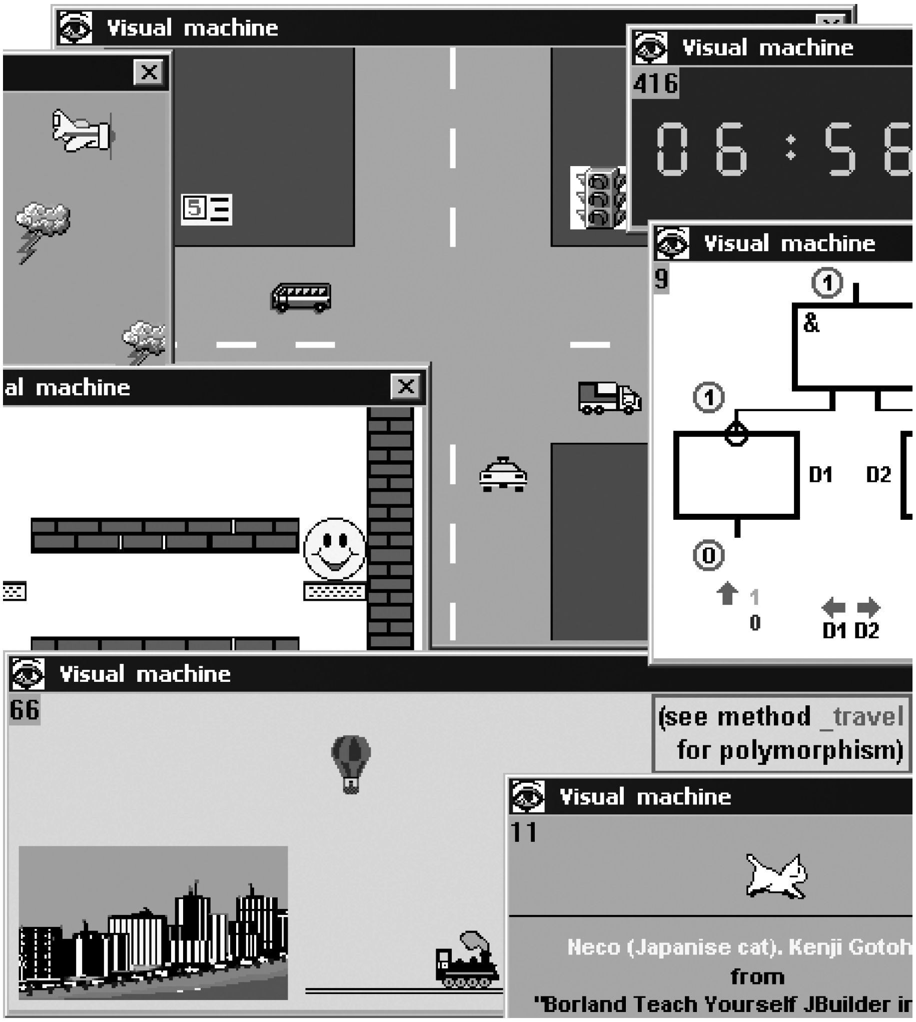


Fig. 2. Examples of object projects, developed in SB

Listing 1. Methods for project *Timer*

```
Method _main

World.timer=World.timer+1
World.carry=1
for OL1: sec_lo..min_hi
if World.carry=1
OL1._inc
endif
endfor
```

```
Method _inc

if .image<=.max_value
.image=.image+1
World.carry=0
else
.image=1
World.carry=1
endif
```

Short notification with cycle *for* in method *_main* is not mandatory; instead of it you can write a longer variant with four similar fragments – for every digit singly.

Omitted name of an object in *_inc* listing is equivalent to *this* construction in Java.

It is necessary to emphasize that we have discussed one of the simplest static projects. Much more complex ones, dynamic and interactive, are also available in SB kit set as demo projects. In Fig. 2 several example are combined together in the collage:

- *Plane* – driving a plane through thundery clouds, using arrow keys (interactive);
- *Traffic* – crossroad controlled with a traffic light;
- *Timer* – timer with minutes and seconds;
- *Lift* – actor goes up and down by elevators, executing its program;
- *Logic* – digital logical scheme (interactive);
- *Polymor* – illustration that different types of objects may travel different ways;
- *Neko* – famous animation, made by Kenji Gotoh.

4.4 How it Works

SB keeps values of all properties for all objects as a matrix. For instance program line *World.timer* = *World.timer* + *1* has internal representation *1.12* = *1.12* + *#1*, which adds direct value 1 to property 12 of object 1.

According to matrix values, visual machine draws the window with our virtual world (Fig. 3). To change a picture we must call any method, which assigns new values to the properties and this will be immediately indicated on a screen. To make our world "alive", SB has timer (its frequency can be adjusted by users). At every tick of this timer method *_main* is called. It does some actions according to our program, changing the values of properties or calling some methods to do it. Finally the visual machine repaints a screen and the process repeats again.

If we press some key or click a mouse button, SB starts one of the special system methods with predefined name, also changing our virtual world through the values of modified properties. This way our virtual world becomes interactive.

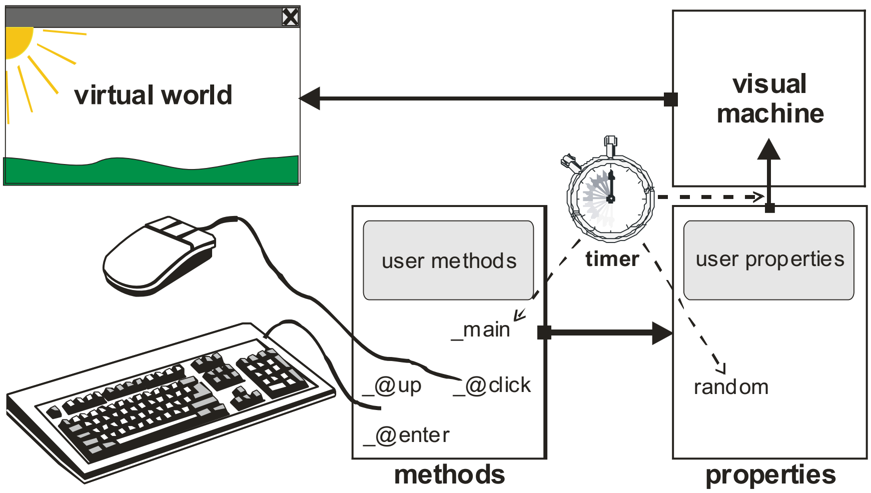


Fig. 3. Functional scheme of the software

5 Conclusion

The analysis made above, confirms that there are convincing reasons to teach OO basics in a school informatics course. Present paper proposes to do it out of direct connection with concrete professional programming language like Java or C++. The matter of the suggestion is to learn this topic during construction of a virtual world from visual objects of different types and describing its behavior by means of the easiest language. The special educational software was developed for these aims; it allows building systems by writing minimal number of program lines.

Easy realization of OO project in SB is achieved by several means. Primarily it is the wide usage of graphical interface. Every object, property or method is creating with the help of "electronic blanks" – the dialog windows with standard graphic controls like text fields, dropdown lists and so on. Instead of writing long statements constantly digressing on syntax rules a pupil just answer clear questions and the software constructs components of OO world automatically. Besides all data about objects is stored and displayed in a form of tables.

Although this paper describes the given author's free software, its main aim is to demonstrate possible way of teaching basic OOP ideas at school. Practice distinctly indicates that students, working with SB, have the ultimate difficulties just when they build an object model but not in the following implementation. So from this point of view the reported method seems very advantageous.

References

1. Bennedsen, J.: Teaching and learning introductory programming – a model-based approach. Ph.D. thesis, Oslo University, Oslo (2008)
2. Bergin, J.: Fourteen Pedagogical Patterns. http://csis.pace.edu/~bergin/PedPat1.3.html

3. Bergin, J., Roberts, J., Pattis, R., Stehlik, M.: Karel++: A Gentle Introduction to the Art of Object-Oriented Programming. Wiley, New York (1996)
4. Bruner, J.S.: The Process of Education. Harvard University Press, Cambridge (1960)
5. Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., Miller, P.: Mini-languages: a way to learn programming principles. Educ. Inform. Tech. **2**(1), 65–83 (1997). https://doi.org/10.1023/A:1018636507883
6. Daly, T., Wrigley, E.: Learning Java Through Alice 3, 2nd edn. CreateSpace Independent Publishing Platform, Charleston (2014)
7. Ehlert, A., Schulte, C.: Empirical comparison of objects-first and objects-later. In: 5th International Workshop on Computing Education Research (ICER 2009), pp. 15–26. ACM, New York (2009). https://doi.org/10.1145/1584322.1584326
8. Eremin, E.: Software system to learn objects. In: 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), p. 188. ACM, New York (2000). https://doi.org/10.1145/343048.343212
9. Holland, S., Griffiths, R. Woodman, M.: Avoiding object misconceptions. In: Miller, J.E. (ed.) 28th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1997), pp. 131–134. ACM, New York (1997). https://doi.org/10.1145/268084.268132
10. Hubwieser, P.: Analysis of learning objectives in object oriented programming. In: Mittermeir, R.T., Sysło, M.M. (eds.) ISSEP 2008. LNCS, vol. 5090, pp. 142–150. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69924-8_13
11. Kölling, M.: Introduction to Programming with Greenfoot Object-Oriented Programming in Java with Games and Simulations, 2nd edn. Pearson, Boston (2016)
12. Kölling, M., Rosenberg, J.: An object-oriented program development environment for the first programming course. In: Klee, K.J. (ed.) 27th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 1996), pp. 83–87. ACM, New York (1996) https://doi.org/10.1145/236452.236514
13. Kölling, M., Rosenberg, J.: BlueJ – The Hitchhiker's Guide to Object Orientation. Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, Technical report no. 2 (2002)
14. Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C., Whalley, J.L.: Research perspectives on the objects-early debate. In: Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2006), pp. 146–165. ACM, New York (2006). https://doi.org/10.1145/1189215.1189183
15. Mannila, L., de Raadt, M.: An objective comparison of languages for teaching introductory programming. In: 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006 (Baltic Sea 2006), pp. 32–37. ACM, New York (2006). https://doi.org/10.1145/1315803.1315811
16. Meyer, B.: Towards an object-oriented curriculum. In: Ege, R., Singh, M., Meyer, B. (eds.) 11th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 1993), pp. 585–594. Prentice-Hall, Upper Saddle River (1993)
17. Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., Paterson, J.: A survey of literature on the teaching of introductory programming. In: Carter, J., Amillo, J. (eds.) Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2007), pp. 204–223. ACM, New York (2007). https://doi.org/10.1145/1345443.1345441
18. Pecinovský, R.: OOP – Learn Object Oriented Thinking and Programming. Eva & Tomáš Bruckner Publishing, Řepín-Živonín (2013)

19. Proulx, V.K., Raab, J., Rasala, R.: Objects from the beginning - with GUIs. In: 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002), pp. 65–69. ACM, New York (2002). https://doi.org/10.1145/544414.544436
20. Ragonis, N., Ben-Ari, M.: On understanding the statics and dynamics of object-oriented programs. In: 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2005), pp. 226–230. ACM, New York (2005). https://doi.org/10.1145/1047344.1047425
21. Ramalingam, V., Wiedenbeck, S.: An empirical study of novice program comprehension in the imperative and object-oriented styles. In: Wiedenbeck, S., Scholtz, J. (eds.) 7th Workshop on Empirical Studies of Programmers (ESP 1997), pp. 124–139. ACM, New York (1997). https://doi.org/10.1145/266399.266411
22. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: a review and discussion. Comput. Sci. Educ. **13**(2), 137–172 (2003)
23. Sajaniemi, J., Kuittinen, M.: From procedures to objects: a research agenda for the psychology of object-oriented programming in education. Hum. Tech. **4**(1), 75–91 (2008). https://doi.org/10.17011/ht/urn.200804151354
24. Sanders, K., Boustedt, J., Eckerdal, A., McCartney, R., Moström, J.E., Thomas, L., Zander. C.: Student understanding of object-oriented programming as expressed in concept maps. In: 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2008), pp. 332–336. ACM, New York (2008). https://doi.org/10.1145/1352135.1352251
25. Sorva, J., Seppälä, O.: Research-based design of the first weeks of CS1. In: 14th Baltic Sea Conference on Computing Education Research: Koli Calling 2014 (Baltic Sea 2014), pp. 71–80. ACM, New York (2014). https://doi.org/10.1145/2674683.2674690
26. Tait, B.: Object orientation in educational software. Innov. Educ. Train. Int. **34**(3), 167–173 (1997). https://doi.org/10.1080/1355800970340302
27. Tomek, I.: Microworlds for teaching concepts of object oriented programming. J. Univ. Comput. Sci. **6**(1), 423–434 (1995). https://doi.org/10.1007/978-3-642-80350-5_35
28. Uysal, M.P.: The effects of objects-first and objects-late methods on achievements of OOP learners. J. Soft. Eng. Appl. **5**(10), 816–822 (2012)
29. Xinogalos, S.: Object-oriented design and programming: an investigation of novices' conceptions on objects and classes. Trans. Comput. Educ. **15**(3), 1–21 (2015). Article 13. https://doi.org/10.1145/2700519
30. Xinogalos, S., Satratzemi, M., Dagdilelis, V.: An introduction to object-oriented programming with a didactic microworld: objectKarel. Comput. Educ. **47**(2), 148–171 (2006). https://doi.org/10.1016/j.compedu.2004.09.005
31. Zhu, H., Zhou, M.: Methodology first and language second: a way to teach object-oriented programming. In: 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003), pp. 140–147. ACM, New York (2003). https://doi.org/10.1145/949344.949389

Real-Time Data Analyses in Secondary Schools Using a Block-Based Programming Language

Andreas Grillenberger(✉) and Ralf Romeike

Computing Education Research Group,
Friedrich-Alexander-Universität Erlangen-Nürnberg,
Martensstraße 3, 91058 Erlangen, Germany
{andreas.grillenberger,ralf.romeike}@fau.de

Abstract. Data management is central to many CS innovations: Smart home technologies and the Internet of Things, for example, are based on processing data with high velocity. One of the most interesting topics emphasizing several challenges in this field is real-time data analysis. In secondary CS education, it is only considered marginally. So far, there are no tools suitable for general-purpose real-time data analysis in school. In this paper, we discuss this topic from a secondary CS education perspective. Besides central concepts and differences to traditional data analysis using relational databases, we describe the development of a general-purpose Snap*!* extension that allows accessing and processing data from various sources. Thereby, students are enabled to conduct data analyses using, for example, sensor data or web APIs. With the example of a weather station, we outline how this tool can be used in school for analyzing sensor data generated in the classroom.

Keywords: Real-time data analysis · Data stream systems · Data management · Sensor data · Physical computing · Secondary CS education

1 Introduction

In previous years, several innovative topics of computer science became pervasive in our daily lives. The increasing possibilities when capturing, processing, analyzing and visualizing data are, for example, central to home automation ("smart home") and when connecting various devices in the Internet of Things. Also, data analyses are often used for decision-finding, even in contexts in which this would hardly be expected in, such as autonomous cars or when addressing voters in election campaigns. The emerging field *data management* comprises several developments originating from the challenge to store and analyze so-called *big data*, i.e. large amounts of data that are generated and analyzed with high velocity and have strongly varying structures [10]. Despite its highly innovative character, various practices and principles of this field seem promising for general educative CS education. For example, partitioning and replication of data are increasingly relevant, not only when developing data management systems, but also when synchronizing data between different devices and services:

Understanding why data are sometimes duplicated or lost and how to prevent this is hardly possible without getting to know the underlying concepts, such as redundancy. One exemplary topic emphasizing several principles and practices of data management, is real-time data analysis. In practice, such analyses are often conducted using data stream systems. With these, immediate analysis of large amounts of data are possible. As today we are often confronted with the results and implications of real-time data analyses, getting to know their basic principles becomes increasingly relevant for understanding such results, for evaluating their quality and relevance, but also for making decisions based on such data [7]. For developing and fostering competencies related to capturing, storing and processing data and for estimating and evaluating the consequences and implications of these possibilities, real-time analyses are an ideal starting point. Hence, in this paper we will outline the functionality and central principles of real-time data analysis in general and of data stream systems in particular. Based on this, we will describe how central ideas of real-time data analysis can be included in secondary education using a general-purpose data stream system extension for the block-based programming language Snap*!* [8].

2 Related Work

Despite their high relevance, not only in computer science, neither the topic data management in general, nor real-time data analyses in particular, have been examined in detail as topics for secondary CS education yet. Although related contexts, such as the Internet of Things, have already been discussed as topic for CS education (cf. e.g. [11]), courses and projects presented are typically designed for higher education. As they pursue different goals, only few aspects can be transferred to our work, such as the practical orientation and the hands-on approach. Also in robotics, processing streams of sensor data and events is central from a technical perspective, but has not yet been discussed as a topic for CS education. As we have shown in a qualitative analysis of various curricula and educational standards, today most data management topics, including real-time data analysis, are typically not included in secondary CS teaching [5]. In consequence, as of today, there are no suitable tools for general-purpose real-time data analysis in school. To bring aspects of modern data analyses to school, we already developed and described a tool for analyzing the Twitter data stream [6], which has been presented to teachers and discussed with them at various opportunities. The advantages in comparison to data analyses using databases were convincing for most teachers. Yet, there was always one concern: To work with the Twitter API, students need to have an account on this platform. As a workaround, we offered the possibility to work offline using a cached data set, but this was obviously less motivating for the students than working with live data. Also, teachers were often concerned that privacy issues might arise and noted the restricted flexibility of the tool, as it only supported the Twitter data stream as a data source. Hence, for tackling these issues, in this paper we describe a new general-purpose approach which allows using various data sources and nearly all possibilities of the continuous query language[1].

3 Real-Time Data Analyses

Today, real-time analyses are being used for several purposes: analyzing credit card transactions in order to prevent fraud (cf. [12]), reacting to temperature changes in smart home environments, or monitoring the environment of smart cars. Although in many use cases, providing results of data analysis immediately is not essential, in most modern use cases there are at least weak restrictions on the reaction times of a system: Typically, users want data to be available as soon as possible, hence "soft real-time"[2] becomes increasingly important. In other use cases, such as industrial robots, the restrictions on data analysis are even harder: they need to fulfill firm or hard real-time requirements[3].

In order to meet a deadline even when analyzing large amounts of data, traditional data analyses are not sufficient. Instead, typical real-time data analyses are in several parts completely different. While traditional data sources are rather finite and discrete, for example when sensor data streams are analyzed, the analysis system needs to handle infinite and continuous data sets (cf. [13]). Also, the amount of data sources (in particular sensors) is growing, with wireless communication the data are available faster than years ago, and the data rate is increasing drastically [13]. Yet, when regarding current CS teaching, analyzing data in schools, if at all, typically takes place using databases: For example, in popular weather station projects, the sensors data are often stored in relational databases and queried using the query language SQL. While this approach is suitable for traditional data analyses, several challenges occur when trying to conduct real-time analysis that way: In particular when data are generated continuously, analysis jobs would need to be started frequently in order to ensure up-to-date results, as determining distinct points in time for starting the analysis is not possible when analyzing data streams[4]. In consequence, the database (and the analysis) gets overloaded or stuck because of the continuous and parallel read and write operations. Additionally, using databases for this purpose also results in enormous amounts of data being stored, because of the high rate of

[1] The "continuous query language" (CQL) is similar in syntax to SQL, but in particular allows using "sliding windows", which makes it suitable for data stream analysis (cf. [1]).

[2] *Soft real-time* allow even frequent misses of the deadline, as only service quality is being influenced (cf. e.g. [3]).

[3] *Hard real-time* strongly requires adherence to a deadline, as exceeding it results in a system failure. *Firm real-time* tolerates missing the given deadline infrequently, but the analysis results become irrelevant after the deadline and the quality of service is degraded.

[4] "A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety." [4].

data generation and as it is not possible to delete outdated values, as information would get lost. Hence, although databases are a suitable tool for storing large amounts of rather static data for a long time and analyzing such data, they can hardly be used for real-time data analysis. Today, this problem can be solved by using data stream systems, which are designed as general-purpose systems for fast analysis of continuous data streams.

In several characteristics, these systems are the opposite of databases: In particular, data stream systems do not store data permanently, but instead process them on-the-fly by applying queries that have been previously defined [9]. Thus, data stream systems are appropriate when data are to be analyzed only once, immediately after being generated. This is the case in particular for sensor data: As every value is only up-to-date until the next value is generated, immediately analyzing them is important. In addition, in such use cases it is typically reasonable to drop values if they cannot be analyzed immediately and to continue with the next value which is then the most relevant one, as significant changes typically do not influence single values only. Data stream systems can, in particular, show their potential when "sliding windows" are used, i.e. when a defined number of recent values or all values from a defined time span are analyzed (cf. [2]). With databases, during every execution of a query, all data stored in the database would be re-evaluated (whether or not they were changed before), while using "sliding windows" allows to cache and evaluate only the relevant data, ensuring that they are deleted from the cache when out-of-date.

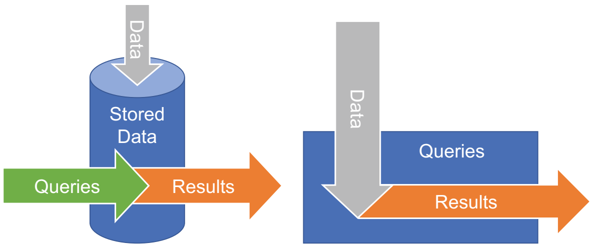


Fig. 1. Functional principles of databases and data stream systems.

The different functional principles of databases and data stream systems are visualized in Fig. 1: When using databases, storing and analyzing data are independent processes. In contrast, in data stream systems, data is directly analyzed after they were received by the system. Hence, many modern systems processing continuous data streams are based on characteristics of data stream systems rather than of databases: For example, when measuring the wind velocity in a smart home project for triggering the closure of windows, it is neither important to process every single value nor to store these data permanently. Hence, data stream systems are ideal for this purpose. To summarize, Table 1 shows a comparison of central characteristics of both, databases and data stream systems.

Table 1. Comparison of the main characteristics of databases and data stream systems.

Characteristics	Database system	Data stream system
Data storage	Permanent storage, often relational but also other paradigms	No storage except caching of values needed for further processing
Data processing	When executing a query	Immediately after new data were received
Typical data	Data that should be stored permanently and that are relevant for long term; often used for multiple queries/purposes; often rather static data	Continuous data streams consisting of data which quickly become outdated; typically only processed for one purpose
Query language	Typically descriptive, often SQL	Typically descriptive, often CQL
Popular implementations	Professional implementations: MySQL, PostgreSQL, MongoDB ...	Research implementations: STREAM, AURORA, TelegraphCQ, Odysseus ...
Central concepts	Integrity, consistency, redundancy	Availability, concurrency

4 Real-Time Data Analyses in Current Secondary CS Teaching

When the topic “data” is addressed in secondary CS teaching, the focus is typically on (relational) databases [5]. This has not been changed since the early 1990s. Thus, because of the tremendous developments in recent years, several topics that are strongly related to the students’ daily life today are only considered marginally, such as data security, privacy, encryption or meta-data. Also, “real-time data analysis” is typically not considered in secondary CS teaching at all: Although such analyses are pervasive in our daily life, understanding their underlying principles and functioning is not a goal of typical CS education. Yet, to understand their relevance, opportunities and threats, getting to know basic aspects of real-time data analysis is essential. For example, it is hard to imagine how large amounts of data captured by CCTV may be analyzed in real-time without knowing the underlying analysis methods. But when they know about the restrictions and differences of real-time analysis in comparison to data analysis in general, students also become familiar with the involved opportunities and threats.

Recently, aspects of data management and real-time data analysis can be found in school teaching when data is acquired and evaluated in physical computing projects. For example, sensors are used for measuring and evaluating environmental influences. In simple projects, data is typically processed directly on the microcontroller, e.g. when just reacting on concrete values or measuring and displaying values such as the current temperature. But in more complex projects, for example calculating average temperatures for defined time frames, analyzing the data directly on the microcontroller is often not possible due to memory and performance restrictions. Hence, data is often sent to a computer for further storage and analysis. This typically involves multiple tools, such as the programming environment for the microcontroller, one for controlling data storage and analysis, and a database such as SQLite for data storage.

Currently, a popular project in secondary schools is building weather stations. With several sensors, data is gathered continuously. Often, the goal is more complex than just showing the current values, as for example, average or extreme values are more interesting. When calculating such values for defined time frames (i.e. for the last 24 h), the analysis becomes too complex for typical microcontrollers used in school. When trying to use traditional approaches for storing and analyzing the sensor data, all values in this time frame need to be stored in a suitable data storage, deleted from it as soon as they are outdated and replaced with other values. This would lead to numerous read/write operations. When using databases for this purpose, as it is common in such projects, the problems described before arise. Yet, professional data stream systems are hardly suitable for use within CS education because of their high complexity.

5 Challenges for Teaching Real-Time Data Analysis

5.1 Data Sources

As analyzing data in real-time is an especially suitable for dynamically changing data, data sources that are traditionally used in CS education are not suitable for this purpose. Instead, there are in particular two approaches for accessing "live data":

Web APIs. With several web applications providing free access to their application programming interfaces (APIs), lots of data can be acquired. For example, social media platforms such as Twitter and Facebook provide access to large parts of their data. There are two approaches for accessing the data: While Twitter uses a "push" method, i.e. the client subscribes to the stream and is notified by the server about new tweets (comparable to the observer pattern), most other APIs use a "pull" approach, i.e. the client frequently requests information. Although pulling produces more unnecessary communication and delays, it has an advantage for teaching: As the connections are not kept open all the time, typically multiple users can use the same credentials. Also, as most APIs are based on the REST principle and hence are accessed via HTTP calls, using them is relatively easy, particularly if they are based on a "pull" approach. Hence, these interfaces are ideal data sources for real-time data analysis, as the data are (depending on the application) highly dynamic and as access is typically only limited by rate limits of the APIs.

Sensor Data. Another data source is sensor data. With physical computing projects gaining popularity in CS education, using sensors for measuring data from the environment of a system has already become common in CS teaching. This is also a suitable approach for gathering data for data management lessons: The main advantage of using sensor data as a basis for data analysis is that it is generated in real-time in the classroom environment, without hurdles of using web APIs, such as mandatory user accounts or privacy issues. Also, as nearly all common programming languages allow communication with microcontrollers, from traditional object-oriented languages such as Java right up to block-based programming environments designed for educational use such as Scratch or Snap*!*, finding a suitable tool for obtaining data is not a problem. Another advantage of using sensor data in data management education is the shift of focus from just processing and analyzing data, to the whole data life cycle from their acquisition and modeling, through processing and analysis, to visualization and (perhaps) deletion of data. This depicts the real usage of data and shows how different CS fields come together in innovative topics like the "Internet of Things", which is based particularly on data management technologies and (interconnected) microcontrollers. As capturing sensor data can be done in various ways, there are also very different types of sensors that might be used, and hence also different ideas for data processing and analysis.

5.2 Development of a Real-Time Data Analysis Extension for Snap*!*

Design Decisions. For accessing both, sensor data and REST APIs, several tools suitable for CS education already exist. In particular, the block-based programming language Snap*!* [8] has high potential for data management teaching: Not only does it provide blocks for sending HTTP requests that are suitable for accessing REST APIs, it cam also communicate with Arduino (and compatible) microcontroller boards in the fork Snap4Arduino[5]. Hence, it is possible to access sensor data captured with such boards. It is also highly expandable, so that processing data is possible using the same tool with which they are captured, and it is easy-to-use for students, even without previous experiences using it. Thus, for conducting real-time data analysis in CS teaching, we implemented the central features of data stream systems and CQL [1] in the Snap*!* programming environment. Using Snap*!* as a basis clearly contributed to our main goals: The tool should be flexible enough to be used with various data sources, it should be easy-to-use in a school context and it should be easy-to-extend by teachers and (ideally) also by the students themselves. To allow the data stream system extension to be used in Snap*!* and all its forks, we only used functionalities that Snap*!* provides in its end-user interface and avoided to directly modify its source code. Hence, the extensions is completely based on primitive Snap*!* blocks and JavaScript functions (which can also be called in Snap*!* out-of-the-box by mapping them into blocks). In addition to extending Snap*!* for allowing data stream analyses, we also extended it for data visualization purposes in the same way.

[5] http://snap4arduino.org.

Implementation of the Data Stream Analysis Extension. In contrast to our tool for analyzing the Twitter data, which was based on imperative queries, for the Snap*!* data stream system extension ("Snap*!*DSS") we use declarative queries. This approach facilitates handling the queries as one unit in the programming environment and allows nesting queries like is possible, for example, in typical query languages such as SQL. Also, as professional data stream systems typically use declarative languages such as CQL, Snap*!*DSS conveys the function of professional data stream analysis tools better than SnapTwitter. For making the extension as flexible as possible, Snap*!*DSS offers the following functionalities:

– Creating a new data stream from any data source ("reporter block" in Snap*!*).
– Combining data from several sources in one stream.
– Running queries on data streams, using aggregate functions, projections, selections and sliding.
– Using analysis results as a data source for a new data stream (nesting).
– Continuous evaluation of queries in the background.

To implement these functions, we created a data structure which is represented in Snap*!* as an unevaluated block. This internal data structure is rather complex: The data stream system, data streams and queries are represented as lists with various fields (cf. Fig. 2). In particular, a data stream stores a list of its queries and information on its data source; each query stores its parameters as well as a cache of current values. When a data stream system is started, each query of all of its streams is executed in the background about once per second[6]. Hence, the values stored in the data stream system are updated permanently, as it is common in data stream systems. Despite its complex structure in the background, Snap*!*DSS can be easily used with the following blocks:

– `create data stream system` : Creates a new data stream system, to which streams can be added for continuous processing.
– `new data stream from ( ) ◂▸` : Creates a new data stream, which is based on the data given in an input field.
– `add stream ☰ to data stream system ☰` : Adds a data stream to a data stream system.
– `select ▾ ( ◂▸ ) from stream ☰ [ last ● values ▾ ]` : Executes queries on the data streams. After the first use of a query, the system continuously captures the relevant data for the query while running. In a query, aggregate functions (average, minimum, maximum, sum) and windows may be used.
– `stop data stream system ☰` : Starts continuous data processing in the background.
– `stop data stream system ☰` : Stops background data processing.

Besides allowing high flexibility and all relevant functionalities, our approach also gives the teacher the opportunity to simply adapt the tool for concrete lessons and thus to further reduce complexity (e.g. by hiding blocks or by creating new ones), without losing the flexibility for complex analyses.

[6] We limited processing of queries to one time per second to prevent performance issues. Yet, this limit can easily be changed by modifying the block in Snap*!*.

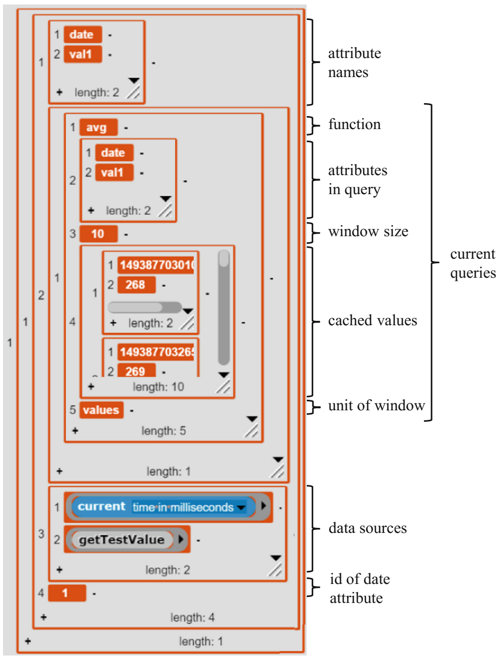


Fig. 2. Internal representation of a data stream in Snap*!*.

Implementation of the Data Visualization Extension. To enable students to visualize data they captured and processed using the data stream system, we also extended Snap*!* in order to allow graph visualizations. Therefore, we use the JavaScript library *plotly.js*[7], which is used in offers diverse plot styles. For using this library in Snap*!*, we use the JavaScript block to load and initialize the API directly in Snap*!*. This implementation is relatively easy, shows how Snap*!* may be extended that way and is prototypical for including JavaScript libraries in Snap*!*. In our extension, the plotly.js library is loaded on-the-fly in the background, thus the user does not have to cope with loading or initializing it and can use it like any other block. Currently, the functions shown in Fig. 3 are implemented: The C-shaped "plot" block prepares the "plotly.js" library and (after defining the traces to plot) generates the image shown on the Snap*!* stage. Within the "plot" block, users can define multiple traces as well as horizontal lines (which are, for example, useful for indicating average values).

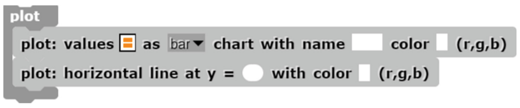

plot
plot: values as bar chart with name color (r,g,b)
plot: horizontal line at y = with color (r,g,b)

Fig. 3. Blocks for creating data visualizations using plotly.js.

[7] https://plot.ly/javascript/.

6 Example Project: Weather Station

As mentioned before, a common example when capturing and processing sensor data in school, is building a weather station. This project allows students to capture lots of data with sensors using almost any typical physical computing platform[8]. In order to avoid the problems of using the typical database-based approach, in the following we will describe, how a weather station can be implemented using our Snap!DSS extension.

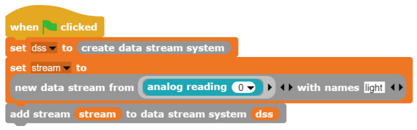
In this project, students can recognize the main functional principles and concepts of real-time data analysis. Hence, they can carry out their own data analysis and understand how easily several data sources can be combined for achieving better analysis results, e.g. by making recorded values more accessible through aggregation and by creating simple forecasts (e.g. using pressure differences as an indicator for weather changes). Also, by including API data, the results could be compared to professional analysis for evaluating them. For capturing and analyzing the data, Snap4Arduino on the software side (with Snap!DSS loaded) and an Arduino Uno microcontroller board on the hardware side can be used. We use a combined sensor for temperature, pressure and humidity, which we connected using the "grove shield"[9]. Yet, our extension does not restrict using other sensors, as long as their values can be read using reporter blocks in Snap!, as those are needed for using the data stream extension. Using our extension, the values read by the reporter blocks are interpreted as a data stream. After adding this stream to the data stream system and starting it, the sensor values are evaluated according to the queries defined. As no queries have been defined yet, nothing would happen when running the program; first, at least one query has to be defined (which can even happen on-the-fly while the system is running). As we are interested in the minimum, average and maximum values of both, pressure and temperature, we can define three queries, as it is depicted in Figs. 4 and 5 for calculating the average value of a light sensor.

The results of the queries can directly be shown, e.g. by clicking on a query block, but they can also be used as input for other blocks in Snap!. Hence, we are, for example, able to show them on a LCD display or to visualize them using the data visualization extension, as it was done in the example shown in Fig. 5. For displaying the data of one sensor and the corresponding average value, the code and the result is shown in Fig. 6. The visualizations may be changed without losing information, because the data from all sensors are analyzed in the background as defined in the queries after starting the system.

This example project (Fig. 7) is easier to understand and implement than the typical implementation in schools using databases, in particular as fewer different systems are involved. Due to the design of the tool, it is clearer and allows to be extended flexibly, and it can also be used for more complex analyses. The project can be realized with students with or without prior knowledge

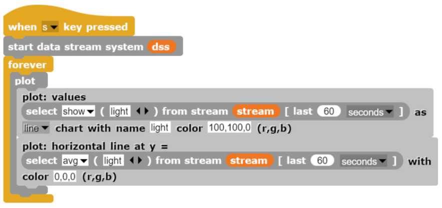
[8] An exemplary project was described by the Raspberry Pi Foundation: https://www.raspberrypi.org/blog/school-weather-station-project/.

[9] http://wiki.seeed.cc/Grove_Starter_Kit_v3/.



```
when clicked
set dss to create data stream system
set stream to
  new data stream from analog reading 0 with names light
add stream stream to data stream system dss
```

Fig. 4. Analyzing a sensor data stream.



```
when s key pressed
start data stream system dss
forever
  plot
    plot: values
    select show ( light ) from stream stream [ last 60 seconds ] as
    line chart with name light color 100,100,0 (r,g,b)
    plot: horizontal line at y =
    select avg ( light ) from stream stream [ last 60 seconds ] with
    color 0,0,0 (r,g,b)
```

Fig. 5. Visualizing sensor data.

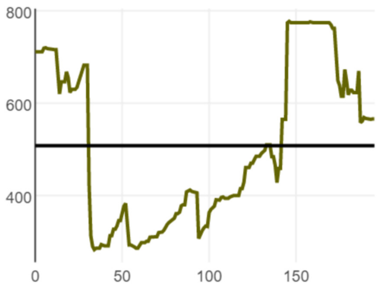


Fig. 6. The line chart represents the actual values, the horizontal line shows the average value.

on data management or physical computing, and with no additional material or systems in comparison to typical physical computing projects. The concept can be transferred to other projects, such as building a smoke detector (which could also take into account temperature increases, in order to prevent false alerts), realizing smart home projects (e.g. switching on the light when the environmental brightness becomes lower than the average in the last five minutes, ensuring that temporary peaks are neglected), or even for quantified self projects (such as realizing a sleep quality measurement device or for measuring heart rates).

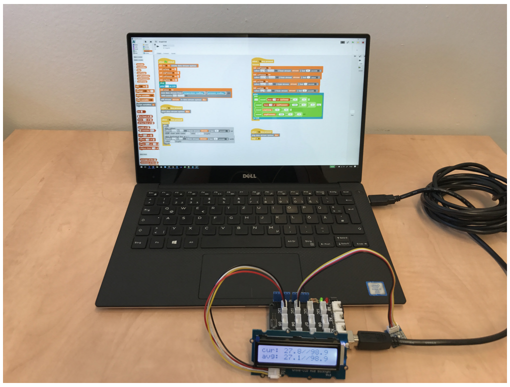


Fig. 7. Exemplary weather station project using a combined temperature and pressure sensor as well as a LCD attached to a grove shield.

7 Summary

In this paper, we discussed central characteristics of real-time data analyses. Based on these, we proposed a tool, which is suitable for conducting general-purpose data stream analysis in secondary CS education. In an example, we have shown how this tool can be used in school in combination with physical computing. Although the Snap!DSS extension reduces complexity, it preserves characteristics and functionalities of real-time data stream analyses. It is oriented at the syntax and semantics of CQL, allows its typical operations and implements central aspects and functionalities of general-purpose data stream systems. In combination with accessible hardware, students are enabled to pursue their own ideas and to conduct analyses on their own, which are otherwise only possible for professionals in the field. Thus, this approach opens up new opportunities for the students to benefit from the innovations in data management. Yet, by preserving the typical character of data stream systems and real-time data analyses, it also fosters a better understanding of several central concepts of data management.

References

1. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. **15**(2), 121–142 (2006)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 1–16. ACM, New York (2002)
3. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems. MIT Press Ltd., Cambridge (2017)
4. Golab, L., Özsu, M.T.: Processing sliding window multi-joins in continuous queries over data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), vol. 29, pp. 500–511. VLDB Endowment (2003)
5. Grillenberger, A., Romeike, R.: A Comparison of the field data management and its representation in secondary CS curricula. In: Proceedings of WiPSCE 2014. ACM, Berlin (2014)
6. Grillenberger, A., Romeike, R.: Analyzing the Twitter data stream using the snap! Learning environment. In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 155–164. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_14
7. Grillenberger, A., Romeike, R.: Teaching data management: key competencies and opportunities. In: Brinda, T., Reynolds, N., Romeike, R. (eds.) Proceedings of KEYCIT 2014. Universitätsverlag Potsdam (2014)
8. Harvey, B., Mönig, J.: Snap! Reference manual (2014). http://snap.berkeley.edu/SnapManual.pdf. Accessed 09 July 2017
9. Krämer, J.: Continuous Queries over Data Streams - Semantics and Implementation. Philipps-Universität Marburg (2007)
10. Laney, D.: 3D data management: controlling data volume, velocity, and variety. Technical report, META Group, February 2001
11. Mäenpää, H., Varjonen, S., Hellas, A., Tarkoma, S., Männistö, T.: Assessing IoT projects in university education: a framework for problem-based learning. In: Proceedings of the 39th International Conference on Software Engineering, pp. 37–46. IEEE Press, Piscataway (2017)
12. Mayer-Schönberger, V., Cukier, K.: Big Data: A Revolution That Will Transform How We Live, Work, and Think. Houghton Mifflin Harcourt (2013)
13. Nittel, S.: Real-time sensor data streams. SIGSPATIAL Spec. **7**(2), 22–28 (2015)

XLogoOnline: A Single-Page, Browser-Based Programming Environment for Schools Aiming at Reducing Cognitive Load on Pupils

Juraj Hromkovič[1], Giovanni Serafini[1], and Jacqueline Staub[1,2](✉)

[1] Department of Computer Science, ETH Zürich,
Universitätstrasse 6, 8092 Zürich, Switzerland
[2] Pädagogische Hochschule Graubünden, Scalärstrasse 17, 7000 Chur, Switzerland
{juraj.hromkovic,giovanni.serafini,jacqueline.staub}@inf.ethz.ch

Abstract. For more than twelve years, our chair has been introducing primary school children to algorithmic thinking by teaching them how to program in Logo. The key element of the proposed didactic approach consists in reducing the extraneous cognitive load on the pupils. We developed and stepwise refined the required teaching materials that allow for introducing only a few instructions in a programming language, which is gradually extended simply relying on modular design. XLogoOnline is our new browser-based, single-page programming environment for schools which is perfectly attuned to our curriculum. We argue that the platform reduces the extraneous cognitive load on the pupils thanks to a heavily-simplified workflow, appropriate for young children, and present evaluations that confirm high usability and acceptance across ages.

1 Introduction

It is crucial that young pupils leave school not only as passive users of computers but also with the ability to think algorithmically and solve problems by programming. This form of learning is constructive, enriches creativity and teaches precision but is strenuous and demanding by its very nature. In this work, we reflect on the difficulties children encounter while learning to program and introduce our two-pronged approach of a programming environment and curriculum as technical and didactic solutions to reduce cognitive load on novice programmers.

1.1 At Long Last, Computer Science in Swiss Schools

Research and development of programming concepts and languages have a long and rich academic tradition in Switzerland. Pascal, Modula-2, and later Oberon were conceived by Swiss computer science pioneer and Turing-award recipient Niklaus Wirth, with his main objectives being that university students are introduced to programming using modern and didactically appropriate teaching material.

It may seem paradoxical, however, that Switzerland is still debating about the opportunity to introduce computer science as a mandatory subject at primary, lower and higher secondary school. While the German speaking part of the country slowly seems to agree on computer science for all children up to 15 years, the so-called "Plan d'étude Romand" in the French part of the confederation still completely neglects computer science [10]. At present, higher secondary school students all around the country are allowed to choose computer science as an optional subject, usually in the last year of the curriculum. The Swiss Conference of Cantonal Ministers of Education is finally considering to introduce Computer Science as a compulsory subject at higher secondary school. For the ongoing debates a decision is expected for October 2017.

For more than twelve years, our chair at ETH Zurich – together with several regional partners, among them Pädagogische Hochschule Graubünden [9] and University of Basel – have been conducting projects introducing primary school children and their teachers to programming in Logo [3–5,11] using our didactic approach. In the last two years, in the greater area of Basel alone, 96 primary school classes (1862 pupils) and their teachers were taught how to program using our teaching materials. Currently, we are even active outside the Swiss borders and are introducing 8 primary school classes (141 pupils and their teachers) to programming in the Principality of Liechtenstein.

In the rest of this paper we present *XLogoOnline*, our new browser-based programming environment for introducing primary school children to programming.

1.2 Organization of the Paper

In Sect. 2, we reflect on the contribution of programming on algorithmic thinking with a focus on preserving pupils' working memory. In Sects. 3 and 4, we highlight technical and didactic design principles of our curriculum before honing in on implementation details of XLogoOnline. Section 5 introduces objectives, setup and results of three experiments after which we finally summarize the key insights of the paper in Sect. 6.

2 Algorithmic Thinking and Cognitive Load

We consider the expressions *algorithmic thinking* and *computational thinking* as two equivalent and interchangeable ways to state the same concept [5]. Knuth already reflected 30 years before Wing about the relations between *algorithmic thinking* and *mathematical thinking*, and informally described *algorithmic thinking* as his "perception of the most typical thought processes used by a computer scientist". Aho captured the nature of algorithmic thinking as follows: "We consider computational thinking to be the thought process involved in formulating problems so their solutions can be represented as computational steps and algorithms" [7]. Irrespective of whether we refer to it in one way or another, we believe that computer science should be introduced at school "as an independent scientific subject" and that it should be "studied both for its intrinsic intellectual and educational value and for its applications to other disciplines" [2].

2.1 Programming is a Key to Algorithmic Thinking

Following the definition of Aho, we are firmly convinced that learning how to develop well-structured programs is a key element to mastering algorithmic thinking, and an ideal way to experience scientific methods at an appropriate level of abstraction. In our curriculum, pupils first learn to write small pieces of code, which then are tested and revised for improvement. We mainly focus on programming-in-the-small and the curriculum trains pupils to implicitly formulate a hypothesis (write a program to solve a given problem), verify their hypothesis (run and test the program), and successively refine the hypothesis as necessary.

Modular design is both a technique to develop well-structured programs and an overall problem solving strategy that can be used to tackle any complex tasks, even outside of computer science. In our programming classes, pupils are explicitly taught to seek geometric figures for repetitive patterns. They rapidly discover the advantages of programming and naming such small bite-size elements, and of reusing them in a proper modular fashion. The pupils learn to successively decompose problems into smaller tasks which are each solved independently, and finally reassembled to solve the original assignment.

Learning to program is generally considered to be a complex cognitive activity. In our didactic approach, programming-in-the-small and modular design require the pupils to organize their programs into conceptually independent units. This modularization process is core to teaching computer science: First, it helps to approach challenging problems by breaking them down into smaller tasks, which can be solved independently by applying top-down or bottom-up problem solving strategies. Second, thinking about problem instances in an abstract way helps identifying structural properties inherent to the problem, and therefore allows the pupils to classify them into abstract problem classes, with common properties that can be solved relying on one single parameterized program. This results in a high intrinsic cognitive load. Educational research highlights, that this kind of cognitive load is specific to the contents and considers it as immutable [14].

2.2 How We Reduce the Extraneous Cognitive Load

The educational materials we developed consist of the German textbook *Einführung in die Programmierung mit Logo* [6], the booklet *Programming in LOGO* [1], and Logo programming environments that are tailored for young pupils and programming exercises with a visual feedback [1]. The main subject of our paper is XLogoOnline, the most recent programming environment in this suite [12,13].

We strongly believe that Logo – despite its age – still is one of the most suitable languages for introducing novices of all ages to programming. Moreover, we are convinced, that Logo together with our teaching materials and the programming environments allow for a noticeable reduction of extraneous cognitive load. Regarding knowledge dimensions presented in the Revision of Bloom's Taxonomy [8], we argue that our proposed didactic setting lower the mental effort and preserve the working memory of the pupils:

1. We massively reduce the impact of *factual knowledge*: Instead of introducing pupils to all the instructions and features of a specific programming language, we deliberately focus on a very limited set of basic instructions and teach the kids how to gradually extend the programming language by naming their programs and making them available for reuse. Furthermore, the chosen syntax is adequate to age and abstraction skills of school children. The pupils are not merely using a programming language but they are developing their own, personalized vocabulary to actively interact with the computer.
2. *Conceptual knowledge* is introduced in a compact and didactically effective way: in the eyes of the pupils, a program simply is a sequence of unambiguous instructions that the computer understands and predictably executes. Furthermore, we focus on only one looping control structure that avoids variables, and, later on, we introduce the simpler notion of a static parameter, as a preparation step to the challenging concept of variable. Variables are postponed to lower secondary school.
3. The *procedural knowledge* and to some extent also *metacognitive knowledge* developed by children mainly relies on modular design. We use this design principle as a straightforward problem solving strategy and, moreover, as a mean to hide nested loops.

Readers who are interested in further details and reflections on our teaching approach should refer to our prior publications [3–5,11]. With the exception of the textbook, all teaching materials and environments are freely available to all.

2.3 Programming Languages for School Education

The choice of a specific programming language for school education strictly depends on the targeted educational objectives. We mainly aim at designing (and not only at using) a language together with the children, and strongly believe that programming classes are a unique chance to learn being precise in a semantic but also in a syntactic way. A text-based programming language such as Logo is inherently well-suited for teaching both precision and modular design. Block-based programming languages assume that the syntax is a major hurdle while learning how to program, and therefore try to lower the extraneous cognitive load by using graphical elements and a drag-and-drop programming approach. Although the general objectives of programming classes with block- and text-based languages seem to be similar, the methods adopted for reducing the cognitive load on the pupils considerably differ and make separate research on this topic necessary.

3 Technical and Didactic Design Principles

XLogoOnline was designed as a browser application to meet the requirements of being an easily deployable programming environment which may be used across platforms. At schools we encounter a plethora of old and modern computer infrastructures that vary from stationary computers to laptops and tablets, that have different operating systems, various screen sizes, and all may or may not be connected to the Internet.

We had to decide upon numerous technical and didactic design alternatives. In the following, we present the most influential choices we have taken and describe their impact on the implementation of our platform:

1. Built-in diagnostics that offer smart hints and pinpoint errors are automatically generated in XLogoOnline and play an important role in the interaction between computer and pupil. They help to foster a mostly autonomous and self-driven learning process, and moreover alleviate pressure on the teacher.
2. XLogoOnline adopts a two-pronged strategy to reduce distractions in the user interface. First, the chosen single-page approach, which makes all information for developing and understanding a program visible at one glance, prevents obstructively layered windows. Second, exhibiting nothing but the most important features makes for a simple and clear user interface.
3. Our reactive, browser-based, client-side application relies on state-of-the-art technologies and runs on a local web-server in offline mode. It uses the local file system to store and retrieve code from earlier sessions. This enables the platform to be deployed independent of whether an internet connection is available or not.
4. Our programming environment relies on Canvas, a native dot-matrix data structure for drawing supported by most modern browsers, which does not pollute the Document Object Model and, moreover, which provides a handle for conveniently downloading pictures. Our tool makes efficient use of the canvas by discerning whether flushing the entire drawing to the canvas is possible, or whether it should be split into sequences (e.g. in case of a *wait* command) as the parser is running.

4 Implementation of XLogoOnline

In the following, we explain what goes on behind the scenes when a Logo instruction is executed by touching upon the parser, the editor and the canvas and explaining what actions are triggered by a simple press of the <enter> button in the input field.

4.1 Information Flow

Pupils interact with four main components (Fig. 1): they issue commands to the *input field* in a line-by-line manner and each line contains either built-in or user-defined commands. These programs are defined in the *code editor* and retrieved from it during execution. Once execution has finished successfully, the result is visible as a drawing in the *rendering component* and the command is added to the *history* which keeps track of all recently issued commands. Next, we zoom in on the pertinent implementation aspects of each component.

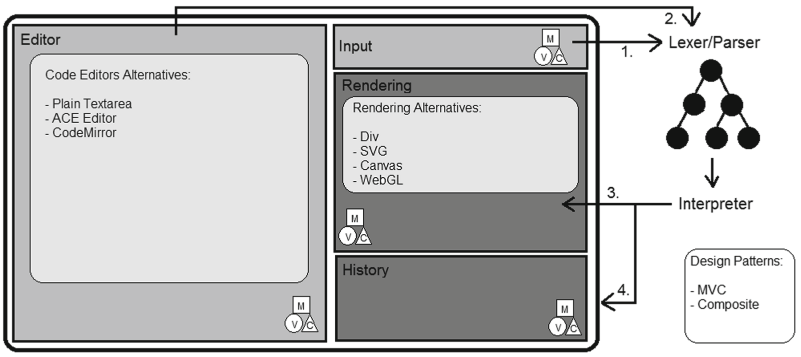


Fig. 1. Information flow

4.2 Editor

While our curriculum only introduces a very basic set of Logo primitives, pupils use the editor to extend the set of commands by defining new customized programs. Programming demands precision and mistakes are inevitable, therefore we provide helpful suggestions in terms of diagnostics which highlight token classes directly in the editor by using different colors and inline markers to pinpoint syntactical errors. Programs are typically declared in the following characteristic form:

```
TO PROGRAMNAME :param1 :param2 ... :paramN
body
END
```

Two especially common mistakes we see in pupils' programs are due to mistaken declarations or invocations of programs. The former usually appear as parameters with missing double-colons, which are interpreted as invocations to non-existing programs. The latter often manifest as missing spaces between program names and their arguments, leading to a single program without any parameters. Calling such a program later causes a run-time error because the according signature cannot be found. We help identify these classes of mistakes using syntax highlighting and checking, which we will explain in the following.

Syntax Highlighting. We assign unique colors to each class of syntactic elements and reflect those in the editor. For instance, built-in commands like *fd* (for forward) or *cs* (for clear screen) are rendered in a different shade than numbers, parameters and mathematical constants. Over time, pupils become acquainted to this coloring scheme and common cases of invalid input such as missing spaces or double colons, become evident. This simple, yet surprisingly effective mechanism helps pupils resolve basic mistakes autonomously and further enhances their understanding of grammatical concepts by revealing how the code is interpreted.

Syntax Checking. As the users type in the editor, their programs are continuously and automatically interpreted in the background after a short period of inactivity. Programs which fail to parse are analyzed further to locate missing keywords and erroneous numbers of parameters. The exact location and type of an error is signaled by adding a red cross directly beside the faulty code which shows a hint when hovered over. Doing so aids pupils and frees them from concentrating on syntactical details and let them focus on semantics instead. Next, we discuss the underlying concepts of our interpretation process.

4.3 Parser and Interpreter

The parser starts by validating that programs satisfy the structural requirements imposed by the language specification. Using a grammar, the sequence of characters found in the program is transformed into a sequence of tokens by the lexer. Next, a parser derives a parse tree from the tokens. We label these two processes as *parsing*. The grammar is defined hierarchically as a set of recursive rules. XLogoOnline parses programs starting at the top-level rule and this starting rule differs in two cases: in the input field only program invocations are allowed; in the editor, however, the starting rule only permits program declarations. Once parsed, a decision is made and ill-typed programs are rejected and otherwise execution continues with an interpretation phase which traverses the parse tree using a visitor. Each node corresponds to a Logo instruction and execution entails drawing to the canvas. These two steps we label *drawing*.

4.4 Canvas

Once a program has successfully executed, the result is shown on a drawing surface which allows the user to drag with their mouse and explore the painting. To support panning, we draw onto a large backing buffer consisting of the visible view port in the middle and its eight neighboring cells. The view port can fill at most the whole screen, leading to a backing buffer nine times the actual screen size. The view port consists of two canvases; one used to display the turtle, the other to draw strokes onto. Strokes are not rendered to the display until explicitly flushed. This is a costly operation because the whole frame needs to be redrawn and so, where suitable, we limit these operations and defer the flush to the end of the execution.

5 Classroom Studies and Performance Evaluation

We evaluated XLogoOnline based on three experiments. First, we measured the cognitive load experienced by pupils and teachers across several age ranges while programming in Logo. Next, we compared XLogoOnline against one of its predecessors in terms of usability. Last, we benchmarked three programs in a performance sanity test measuring elapsed time for drawing and compilation.

5.1 Programming and Cognitive Load Across Age Groups

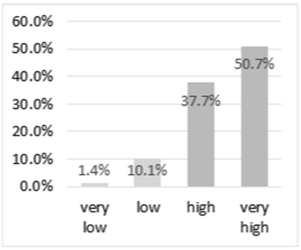
In a typical classroom, a single teacher will guide and assist the entire class; pupils learning how to program, however, progress at differing rates. Ideally pupils should work autonomously while not experiencing cognitive overload. In order to observe children programming in a real classroom environment, we ran several introductory lessons, measured pupils' and teachers' exhaustion levels using a questionnaire and examined the differences in cognitive load across ages.

Setup and Participants. We introduced 75 children (23 girls and 52 boys) and 12 teachers to programming in Logo. In groups of 12, the (mostly) novice programmers spent 75 min solving exercises using sequences of basic movement commands. Each group consisted either of (i) children aged 6 to 9 being exposed to geometry for the first time, (ii) children aged 10 to 13 more used to text-style exercises, geometry and abstract thinking, or (iii) teachers who have progressed even further in their education but mostly lack prior knowledge in computer science. Our hypothesis is that with increasing age students become more accustomed to abstract thinking and their cognitive load decreases. Cognitive load, however, cannot be measured in absolute terms; so we instead asked participants about their level of exhaustion by having them fill out a questionnaire after programming. Answers were provided along a 4-option Likert-type scale featuring smiley faces. This subjective, indirect measure was proposed in prior work by Paas et al. [15], which assumed that the felt complexity of a task and the cognitive load are correlated.

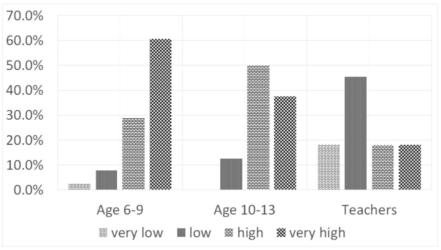
Results. The combined student data confirmed that children indeed feel cognitively exhausted while programming (Fig. 2a) reflecting the need for a tool which eases their learning whilst also not imposing too much extraneous cognitive load.

The more detailed breakdown by age in Fig. 2b confirms our hypothesis; the shifting peaks reflect that teachers are least exhausted, followed in turn by the older and the younger pupils respectively who showed increased exhaustion.

Since exhaustion levels proved to be similar and rather high in both student groups, we require an objective measure which helps differentiating more finely. For this reason, we plan to run a follow-up study examining cognitive load by measuring the number of exercises solved in total. Furthermore, we will explore the influence of different exercise types on the cognitive load of the programmer.



(a) Overall pupil exhaustion



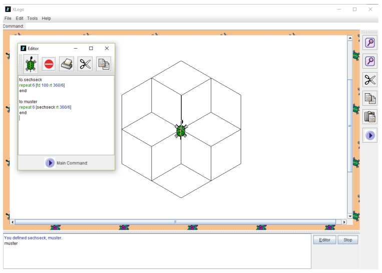
(b) Exhaustion across three age groups

Fig. 2. Questionnaire indicate particularly high cognitive load for young pupils.

The experiment confirmed that most children experience rather high cognitive load while programming, which reinforces the need for tailored teaching material to enable better learning experiences. Children aged 6–9 showed the highest exhaustion levels, while children aged 10–13 were not only less exhausted, but we also observed them to be more progressive and able to work without direct guidance.

5.2 Usability Comparison of XLogo and XLogoOnline

To understand whether our design choices had a positive impact on user experience, we ran a second user study and examined the children's interaction with XLogoOnline, their understanding of the purpose of selected components therein and general impression of the user interface. Figure 3 shows a side-by-side comparison of what solving the same exercise would look like in XLogoOnline and its predecessor XLogo.

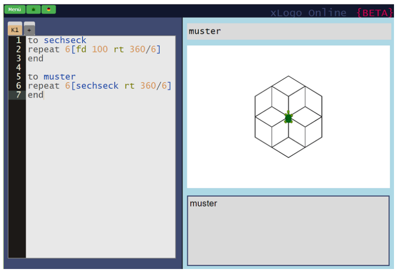


```
to sechseck
repeat 6 [fd 100 rt 360/6]
end

to muster
repeat 6 [sechseck rt 360/6]
end
```

```
You defined sechseck, muster.
muster
```

(a) Predecessor (native): XLogo



```
to sechseck
repeat 6[fd 100 rt 360/6]
end

to muster
repeat 6[sechseck rt 360/6]
end
```

```
muster
```

```
muster
```

(b) Our contribution: XLogoOnline

Fig. 3. XLogoOnline features an uncluttered single-page layout with bigger fonts and a reduced set of GUI features for reducing the extraneous cognitive load and in support of a young audience whose fine motor skills have yet to fully develop.

Setup and Participants. A fifth-grade class consisting of 18 pupils, generally aged 11 or 12 years, participated in this on-site experiment. The class was first introduced to the traditional programming environment XLogo for three weeks where they learned about the basic commands as well as the concepts of repetition and modularization in Logo.

After getting used to programming in XLogo for three programming sessions which lasted two hours each, we introduced XLogoOnline while covering regular polygons. After a short introduction, we left the children to solve exercises for two hours before finishing the experiment by handing out feedback questionnaires covering three main questions: first, we checked their understanding of the use of the four main UI components by letting pupils draw connections between pictures of both user interfaces and descriptions of the components' purpose; second, we asked pupils to grade the four components in XLogoOnline on both their visual appearance and their functionality; last, we asked them to tell us about their general impression working with the platform.

Results. A vast majority (83% of all pupils) understood the usage and meaning of the four UI components correctly and could easily transfer from XLogo to XLogoOnline. We received strongly positive feedback (praises for the syntax checker, error pinpointing, layout, clarity and comprehensibility) which give reassurance that simplifying the user interface in XLogoOnline, adopting a flat design and equipping it with additional diagnostic tools like syntax highlighting and syntax checking are indeed beneficial. Some pupils missed certain features XLogo provided and others found the space usage not optimal, which we subsequently adjusted according to their needs. We conclude, that the transition between XLogo and XLogoOnline posed no significant burden on the students.

5.3 Performance Test

It is easy to build complex shapes involving several tens of thousands of drawing instructions by simply repeating sequences of basic patterns many times over. To ensure that even these complex exercises in our teaching materials are executed in reasonable time, we chose three benchmark programs (shown in Fig. 4) which represent a range from short to more time-intense programs and checked their performance.

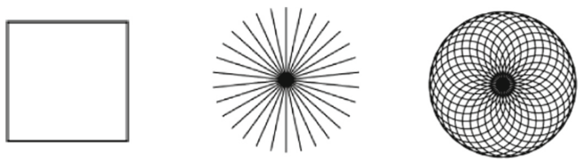

(a) Cheap (b) Medium (c) Expensive

Fig. 4. Benchmark programs ranging from cheap through expensive

The respective programs were re-run 1000 times in XLogoOnline while measuring their overall execution times. We found an average performance of 20000 drawing instructions per seconds on a four-year old off-the-shelf laptop. Consequently, the running time for simple shapes like the ones shown in Fig. 4a and b add up to merely a few milliseconds. Moreover, even the most time-intense exercises in our curriculum run to completion on the order of seconds, which we consider satisfactory for the typical classroom setting.

Finally, we investigated the breakdown of execution time between drawing and parsing to determine the cost of diagnostics, which rely on repeated parsing of the input. We found drawing to be the more influential factor between the two phases and that, as programs get bigger, it increasingly dominates the overall execution time (shown in Fig. 5). Diagnostics like error classification and additional hints are potentially expensive, however, the time spent in parsing constitutes a negligible fraction of the total time and significantly helps pupils recover from their mistakes.

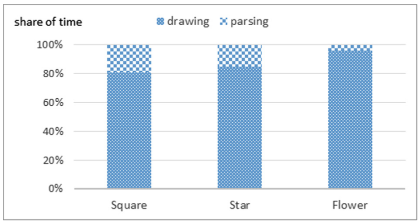


Fig. 5. The larger the program, the more time we spent traversing the parse tree (i.e. drawing) while building the parse tree (i.e. parsing) takes only a fraction of the time.

In summary, we found XLogoOnline able to execute all exercises in our teaching material in reasonable times and parsing to be relatively cheap which allows for further diagnostics as an additional help for the pupil, supporting a more autonomous diagnosis and remedy of syntactic errors.

6 Conclusion

Teaching how to program in hundreds of schools, we found algorithmic thinking and programming to be exciting but exhaustive activities, especially for young pupils. We introduce XLogoOnline, our new browser-based programming environment that aims at reducing the extraneous cognitive load on school children. It is tailored around a curriculum with a deliberately small set of basic commands. Pupils steadily learn to expand the language with custom commands while solving algorithmic problems. As a distinguishing attribute, our programming environment features diagnostics which provide useful feedback to the programmer while having an uncluttered and easy to use single-page user interface. The programming IDE can be deployed at the click of a button using only a web browser and, thanks to its reactive design, can be run on a broad set of devices with different screen sizes and platforms. We ran classroom studies investigating cognitive load during introductory lessons and found a clear need for appropriate didactic settings which reduce the load on pupils' working memory. XLogoOnline was shown to be intuitive and generally approved by primary school pupils.

Future Work: Our broader goals are to understand what troubles young programmers the most. Web environments are especially well-suited for collecting rich data sets, for instance, containing student mistakes. Analyzing this data may reveal patterns which can be tied back to conceptual error classes and tackled more explicitly in our curriculum. Further, we are investigating social effects in programming and have built an early prototype extension for XLogoOnline that allows pupils to collaborate on exercises by sharing code and drawings with their classmates.

References

1. Logo-Uunterrichtsmaterialialien & -Programmierumgebungen. http://www.abz.inf.ethz.ch/logo. Accessed 18 Sept 2017
2. Gander, W., et al.: Informatics education: Europe cannot afford to miss the boat. Informatics Europe & ACM Europe Working Group on Informatics Education (2013)
3. Hromkovič, J., Kohn, T., Komm, D., Serafini, G.: Combining the power of Python with the simplicity of logo for a sustainable computer science education. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 155–166. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_13
4. Hromkovič, J., Kohn, T., Komm, D., Serafini, G.: Examples of algorithmic thinking in programming education. Olympiads Inform. **10**, 111–124 (2016)
5. Hromkovic, J., Kohn, T., Komm, D., Serafini, G.: Algorithmic thinking from the start. Bull. EATCS **121**, 132–139 (2017)
6. Hromkovič, J.: Einführung in die Programmierung mit LOGO, 3rd edn. Springer, Wiesbaden (2014). https://doi.org/10.1007/978-3-8348-2266-6
7. Knuth, D.E.: Algorithmic thinking and mathematical thinking. Am. Math. Mon. **92**(3), 170–181 (1985)
8. Krathwohl, D.R.: A revision of Bloom's taxonomy: an overview. Theor. Pract. **41**(4), 212–218 (2002)
9. Matter, B.: Projekt "Programmieren in der Primarschule". http://www.abz.inf.ethz.ch/wp-content/uploads/2015/03/InfobroschAug2010-3.pdf. Accessed 18 Sept 2017
10. Parriaux, G., Pellet, J.-P.: Computer science in the eyes of its teachers in french-speaking Switzerland. In: Brodnik, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 179–190. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_15
11. Serafini, G.: Teaching programming at primary schools: visions, experiences, and long-term research prospects. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 143–154. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_13
12. Staub, J.: XLogoOnline. http://ethz.ch/xlogo. Accessed 18 Sept 2017

13. Staub, J.: Xlogo online - a web-based programming IDE for Logo. Master Thesis (2016). https://e-collection.library.ethz.ch/view/eth:49742?lang=en
14. Sweller, J.: Cognitive load theory. Psychol. Learn. Motiv. **55**, 37–76 (2011). Academic Press
15. Paas, F.G.W.C., van Merriënboer, J.J.G., Adam, J.J.: Measurement of cognitive load in instructional research. Percept. Motor Skills **79**(1), 419–430 (1994)

Introduction to Bebras Challenge Management: Overview and Analyses of Developed Systems

Valentina Dagiene, Gabriele Stupuriene(✉), Lina Vinikiene, and Rimantas Zakauskas

Vilnius University Institute of Mathematics and Informatics, Akademijos Street 4, 08663 Vilnius, Lithuania
{valentina.dagiene,gabriele.stupuriene,lina.vinikiene}@mii.vu.lt
rimantas.zakauskas@gmail.com

Abstract. The paper analyses contest management systems used by the multicultural Bebras challenge community. Starting as a single annual contest in 2004 in a few countries, the Bebras challenge on Informatics and Computational Thinking has spread to over 60 countries and successfully developed a worldwide network. A requisite tool to perform the challenge is a Contest Management System (CMS). Countries have been developing various systems, depending on their practice, the number of participants, financial issues, etc. The overview and analyses of the Bebras CMS used by various countries are presented here. The main focus concerns the Lithuanian Bebras CMS, its development, architecture, functionalities, and thoughts for future improvements.

Keywords: Bebras challenge · Computational thinking · Contest management system · CMS · Informatics education

1 Introduction

The Bebras challenge[1] on informatics and computational thinking (CT) is an educational community network which brings together teachers and scientists who are interested in teaching informatics at primary and secondary schools. The aim of the Bebras challenge is to engage students in informatics, deepen their understanding in computer technologies, and promote learning by solving short informatics concept-based tasks [1]. The tasks are a crucial point of the challenge: they should involve at least one informatics concept, arranged with an interesting story, be attractive, short, and interactive or dynamic [2].

In general, the challenge is divided into six age categories: Pre–Primary (grades 1–2), Little Beavers (grades 3–4), Benjamin (grades 5–6), Cadet (grades 7–8), Junior (grades 9–10), and Senior (grades 11–13). The core part of the Bebras challenge is a contest organized by each country in the second and third weeks of November. The Southern Hemisphere countries (Australia, New Zealand, and Malaysia) have a different time schedule. Additional rounds of the

[1] http://bebras.org.

V. Dagiene and A. Hellas (Eds.): ISSEP 2017, LNCS 10696, pp. 232–243, 2017.
https://doi.org/10.1007/978-3-319-71483-7_19

contest and other problem-solving activities are organized in various countries. In Lithuania, the Bebras challenge is organized in two rounds: a school-wide contest in November and a nation-wide competition for selected high school students in January of the following year. During the school-wide round students perform tasks using computers and are supervised by teachers in their schools. The second round of the challenge is supervised by universities or colleges in various regions. In Lithuania, over 33 thousand participants performed tasks in the first round (2016) and 385 of them were invited to the second round in January 2017.

A contest management system (CMS) is an essential environment for managing the challenge effectively and efficiently. It should support simple tools, which enable task development, users' management, grading, announcement areas, records of solutions, reports, and data storage. In order to implement the Bebras challenge, several contest management systems have been developed and adapted to users and their needs (described in Sect. 3).

In this paper, we focus mainly on the system that supports the Lithuanian Bebras challenge on informatics and computational thinking. The paper is organized as follows: Sect. 2 describes some examples of CMSs used in various countries. In Sect. 3 we review a survey that was provided to the Bebras community. Main aim of this survey was to gather information about management and implementation of the Bebras challenge, as well as about the systems used, and to get an understanding of the differences of the basic challenge management principles. Section 4 presents a more detailed description of the Lithuanian Bebras contest management system, with the improvements from its first deployment until today. We end with a Conclusion.

2 Examples of the Contemporary Contest Management Systems

The International Olympiad in Informatics[2] (IOI) is one of the most popular competition in Computer science. Students work individually on a set of Computer Science assignments. At the IOI the use of a grading system (contest management system) is an absolute necessity. This system provides all the necessary content for the students to successfully prepare and participate in the programming contests, as well as all the necessary capabilities for the organization of programming contests, including submission of tasks and automatic grading [3]. The CMS (used for IOI 2012) was organized in a modular way, with different services, running potentially, on different machines. When applicable, services can be replicated to allow scaling for larger contests [4]. Whatever the system offers, it should be designed such that it is simple, robust, secure and flexible [5].

The Bebras challenge is not a programming contest, so requirements for this system are different. A short presentation of the existing Bebras systems was published in the paper [6].

[2] http://www.ioinformatics.org.

A Finnish CMS[3] was built using *Rails*, a rapid web development framework using the programming language *Ruby*. In time, features were added to the system by different developers. Some of these features break the separation between models, views, and controllers. For example, although the system was designed to be database-agnostic, its current version occasionally accesses the *SQL* database directly and is therefore strongly tied to *MySQL*.

The CMS of France[4] is one of the most optimal and well-developed: in this system, an optimized front-end is used which reduces the number of requests from clients to the server and consequently the load on a server. While the web workload is distributed, all the web servers access a single relational database. The sessions are implemented using *Memcached* technology, reducing the load on a database while still maintaining a single session across all the web servers. If the web servers become unavailable, competitors are provided with a coded message at the end of their competition. They can send this coded message to the organizers using e-mail and have their results entered into the system. To minimize the communication between the web server and the clients, results are only submitted at the end of the competition, with no backup in case a competitor web browser crashes.

In 2013, Slovenia started to develop a new Bebras CMS[5] that have to be [6]: high performance, scalable, and fault-tolerant. To achieve that, a three-layer architecture consisting of a front-end layer, a business logic layer, and a distributed database back-end layer was applied [7]. As shown in Fig. 1, the front-end was implemented using *HTML*, *CSS* and *JavaScript*, the business logic layer was implemented in *PHP* using the *Yii* web development framework, while *MySQL* Cluster was used as the database back-end.

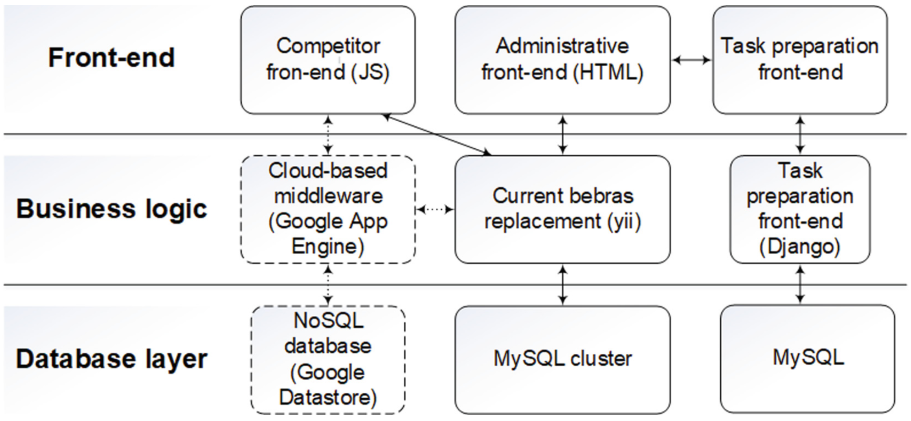

Front-end
Competitor fron-end (JS)
Administrative front-end (HTML)
Task preparation front-end
Business logic
Cloud-based middleware (Google App Engine)
Current bebras replacement (yii)
Task preparation front-end (Django)
Database layer
NoSQL database (Google Datastore)
MySQL cluster
MySQL

Fig. 1. The architecture of the Slovenian Bebras CMS [6]

[3] http://www.majava-kilpailu.fi.
[4] http://concours.castor-informatique.fr/.
[5] http://bober.acm.si/.

The Bebras challenge in Russia and Belarus is supported by distance contests execution/educational system. The system is implemented using *Java*, *Play framework*, *MongoD*B, and can support different types of competitions, such as the contest, “Construct, Test, Explore”. All tasks are uploaded to the system by filling fields, such as a statement, question, a set of answers or interaction, and an explanation containing the correct answer, “It’s Informatics”. Two types of tasks are used:

– Multiple-choice questions with the opportunity to select one of four variants.
– Dynamic tasks. These tasks were implemented in *JavaScript*. There are libraries that allow the creation of drag-and-drop, text input field tasks.

Both types of questions provide the opportunity to select “I don’t know”. Participants in the challenge, school organizers who manage participants from their schools, higher-level organizers (who manage the school organizer), etc. are roles supported by the system [8].

Pikl et al. characterize the ideas for the competition systems and websites, such as logical layout, important information visible at all times, frequently asked questions, discussion between visitors, and a dynamic web page [9]. The history, results, information about the competition, who the competition is for, contact form and additional information should be provided at the competition site.

3 Survey of the Bebras CMS in Various Countries

In this section we describe the process of collecting the data and the results from analysing these data. A questionnaire with fourteen open questions was announced in May of 2016 and was accessible until the end of February (2017). The aim of this questionnaire was to collect information about CMSs in different Bebras community countries, to understand the real situation about CMSs (what is common, what is different), and to discover valuable suggestions for others. Thirty-two countries answered the questionnaires regarding the Bebras challenge. Table 1 shows the distribution of countries by responsibilities of CMS support (organizers in countries themselves create and develop the system; support of the system is trusted for the private company[6]; organizers use platforms available on the Internet).

Sweden and Finland collaborated in the development of the same system. Their system is implemented using *Ruby* and interactive tasks are created using *JavaScript*. Serbia uses Slovenia’s well-developed system (from 2013). Turkey uses LMS *Moodle* (however, they do not have interactive tasks). This is easy to use, although the interface should be changed and there should be a special template for the contest [10]. Croatia also uses LMS *Moodle*. Teachers there take care of editing, publishing tasks, and managing participants with *Moodle*, since every student has their own access to the system. Belarus uses the

[6] https://www.eljakim.nl/project/beverwedstrijd/.

Table 1. Countries distribution by responsibilities of CMS support

CMS developed by organizers	Belgium, Bulgaria, Cyprus, Estonia, Finland, France, Hungary, Indonesia, Italy, Latvia, Lithuania, Macedonia, Russia, Slovakia, Slovenia, Spain, Taiwan, Ukraine
CMS developed by a company	Canada, Germany, Ireland, Japan, The Netherlands, Romania, Singapore, Switzerland, USA
Platforms	Croatia (Moodle), Turkey (Moodle), Belarus (Yandex Contest)

Yandex.Contest system[7]. It is similar to the ACM-like contest system where it is possible to check test-like questions. Macedonia prefers to use a web-based system, developed with *node.js*. The main features of this system are: multiple browsers and devices support, statistics collection, data backup, and it can also sustain the loss of a server or database.

The crucial point for CMSs is the number of people who participate at the same time. It partially depends on the settings of the system. However, small countries with fewer participants do not measure this. For example, for France (more than 470000 participants) and for Germany (more than 290000 participants) the maximum number of participants is 10 000. The French platform is designed to handle much more, if needed. For Belarus, Croatia, Lithuania, and Turkey, the maximum number of participants at the same time is 1000, and for Bulgaria, Finland, and Italy, it is 500 participants. Slovenia is able to connect the largest number of participants at the same time (more than 20,000). There is no limitation for contestants in Ukraine because they work offline, their answers are recorded as files, and are collected afterwards.

In general, the Bebras challenge is held online and participants solve tasks using computers, but in some countries students have the opportunity to solve tasks using other devices. For example, in France, Germany, Italy, The Netherlands, Macedonia, Spain, and USA students can solve tasks using any computer with a browser, an Android, or iOS tablets. In Indonesia, Japan and Latvia students use desktop computers, laptops, tablets, and mobile phones. In Belarus, Canada, Estonia, Hungary, Serbia and Singapore only computers are used. In some cases, only computers are recommended due to interactive tasks, because tasks may not work correctly, or the site is not optimized for mobile phones. In Slovenia, from second grade to fifth grade, tasks are completed on paper and from sixth grade, computers are used.

Using CMS options or additional analytics tools, organizers can provide data about their system, devices, technical details, the number of participants, and answer statistics for those participants. For example, France and Russia collect information about devices or browser versions through *Google Analytics*. Belarus has installed *Yandex.Metrica*. Twenty of thirty-two countries gather information about gender and seven countries are not interested in the gender

[7] https://contest.yandex.ru.

issue. In some cases, such as Italy and Singapore, the possibility of indicating gender is optional. All countries collect the number of participants by age group, but in some CMSs, such as in Belarus, France, and Romania, participants are listed with their precise ages. Personal information, such as name, surname, school, or language is presented in individual cases. It depends on the country's attitude to privacy rules and data publication. Some privacy rules forbid using the student data for statistical research.

The questionnaire confirms that the Lithuanian Bebras CMS has the most developed data collection and statistics analysis. Nevertheless, sixteen countries indicated the need to gather data about gender, devices used, etc., in each country. Discussions were also recorded about a privacy policy, depending on what kind of data were recorded.

Bebras tasks can be of any type, from very basic multiple-choice tasks, to interactive tasks where students manipulate objects visually, or even to advanced interactive tasks where students can write and run small programs. Interactivity is very typical for computers; thus, it is clear that a computer-oriented contest should apply interactive elements to explain or solve tasks. These interactive tasks are often "funny" and easy to understand. However, the interactive elements require a lot of effort by the programmer to implement them. According to the answers from the questionnaire, twelve countries use only multiple choice or open-ended questions for the Bebras tasks. Nineteen countries provide and interactive tasks during the Bebras challenge. Eighth countries out of the nineteen surveyed countries, that use interactive tasks, use the *Bebras Lodge* tool. This is a special tool for creating and developing interactive (dynamic) tasks[8]. France has developed its own special Bebras CMS, which manages all interactive Bebras tasks during the challenge. They use *JavaScript* with *RaphaelJS* library, as well as their own set of libraries[9]. Other technologies used for interactive task implementation are *HTML*5 (Italy) and *JavaScript* (Estonia, Finland, Japan, and Russia). Seven countries (out of thirteen, which do not have interactive tasks) plan to introduce interactive tasks in their CMS. Nevertheless, interactive tasks installation depends on their systems' facilities.

Experience with contest management inspires those interested to think about new system features or improvements for the present CMS. French Bebras organizers would like to have a tool for teachers for creating interactive tasks (something similar to the *Bebras Lodge* tool, but with a different approach). Germany would like to have an API to import, store, and export Bebras tasks including their complete interactivity (the Bebras Pool). Lithuania has a plan to collect data regarding how many times participants have a second look at the same task or return to resolve that task, and how many times they have changed the answer. Slovakia wants to measure the time spent on each task. Ukraine would like to do compatibility with mobile devices and developer-friendly animations. Belarus, Canada, Croatia, Indonesia, Serbia and Turkey are planning to add different types of interactive tasks or develop more interactive task features. Macedonia

[8] http://bebras.licejus.lt/login.
[9] https://github.com/France-ioi/bebras-modules/tree/master/pemFioi.

would like to add more functionality to its CMS to create a friendlier environment for the teacher (for example, see information about students). Macedonia also emphasized the importance of testing as ensuring the performance of participants. Singapore is planning to introduce reports on results for schools and students into their CMS.

4 Bebras CMS Development: Lithuanian Experience

At the beginning of the Bebras challenge in Lithuania (2004), we used PDF technology. Organizers were faced with the basic problem of software installation, students' answers upload, and delivery to the central administration. The Lithuanian Bebras CMS was developed in 2010 and named as the Bebras challenge field[10]. The Lithuanian Bebras CMS was is as a tool to organize the contest, manage information about participants (students) and teachers, to gather data on task solutions, to manage the challenge, and provide detailed statistics and reports. The system was tested by exploitation (more than 30 000 student's entered the system during last year) and efficiency-designed for managing challenge. More than 5 500 new accounts were created for primary and secondary school students last year. The number of new user accounts is growing by similar additional accounts each year.

4.1 System Description

The Lithuanian Bebras CMS is dedicated to create, manage, and administer the challenge online. It works as a web application. The system is based on three-level architecture (similar to the CMS of Slovenia) (Fig. 2). CMS works as a web application and consists of a set of subsystems, which has a well-design interface:

– System administration (security, back up, resource monitoring, etc.),
– User management (registration system, authentication, user profile),
– Challenge management (creation, administration and monitoring),
– School administration (official list of all schools in Lithuania). A school list is updated in accordance with cooperation with the Centre of Information technologies in Education every year.
– Tasks management (create, import tasks),
– Results and communication management (participants and teachers can discuss about particular tasks, preview statistical data).

The Lithuanian Bebras CMS is developed by using the *MySQL* relational database management system (DBMS), *Apache HTTP* server and *PHP* programming language. The operational system for web server is *Linux*. CMS uses the Model-View-Controller structural pattern, which means that an application should be separated from its presentation into three main parts: the model, the view, and the controller. In Model-View-Controller, the view displays information to the user and together with the controller comprises the application's user

[10] http://lt.bebras.lt.

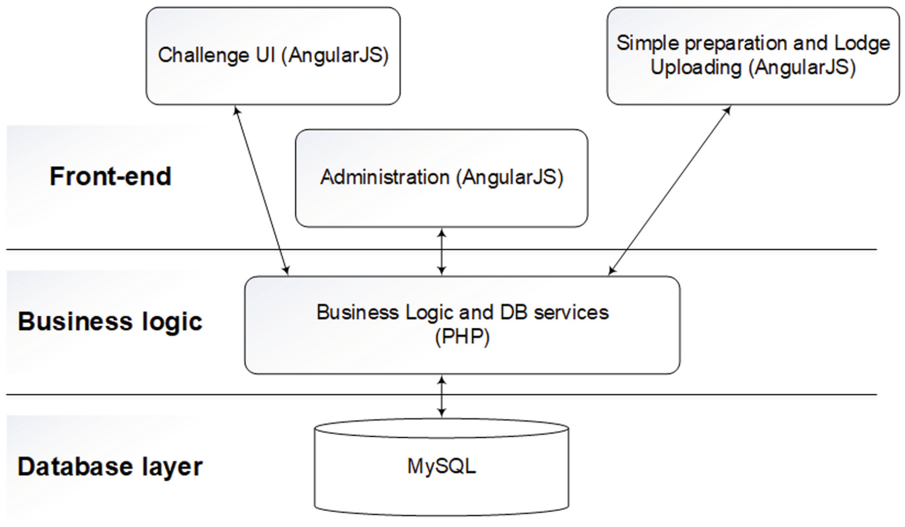

Challenge UI (AngularJS)
Simple preparation and Lodge Uploading (AngularJS)
Front-end
Administration (AngularJS)
Business logic
Business Logic and DB services (PHP)
Database layer
MySQL

Fig. 2. The architecture of the Lithuanian Bebras CMS

interface [11]. A relational database is created to store details of tasks, students and teachers. The data are stored in different tables and relations are established using primary keys or other keys known as foreign keys. CMS is built to be compatible with all operating systems and the latest versions of browsers, although the use of Internet explorer or Edge browsers are not recommended.

During the week of the challenge various statistical data are collected in order to gather information about students' abilities to solve tasks. In addition, this information helps to improve informatics education and improve teacher professional development.

4.2 Functional Requirements

In this section we discuss the Bebras CMS functionality using Use case diagram (Fig. 3). System administrators have full access, including management of task: creation and importation from the *Bebras Lodge* tool. Teachers are provided with challenge access to their school's students and have access to the results of their students. Teachers register their students and system administrators enroll them to the system (it helps to avoid cheating). Registered teachers can confirm students' participation in the challenge during the school-wide challenge in November. Furthermore, teachers have the opportunity to preview the tasks, participate in the discussion, and print certificates for their own students. Students have access to the challenge during the Bebras week and can preview the tasks, and comment on or discuss the particular tasks after the challenge week. Students can see their results only after completing the challenge.

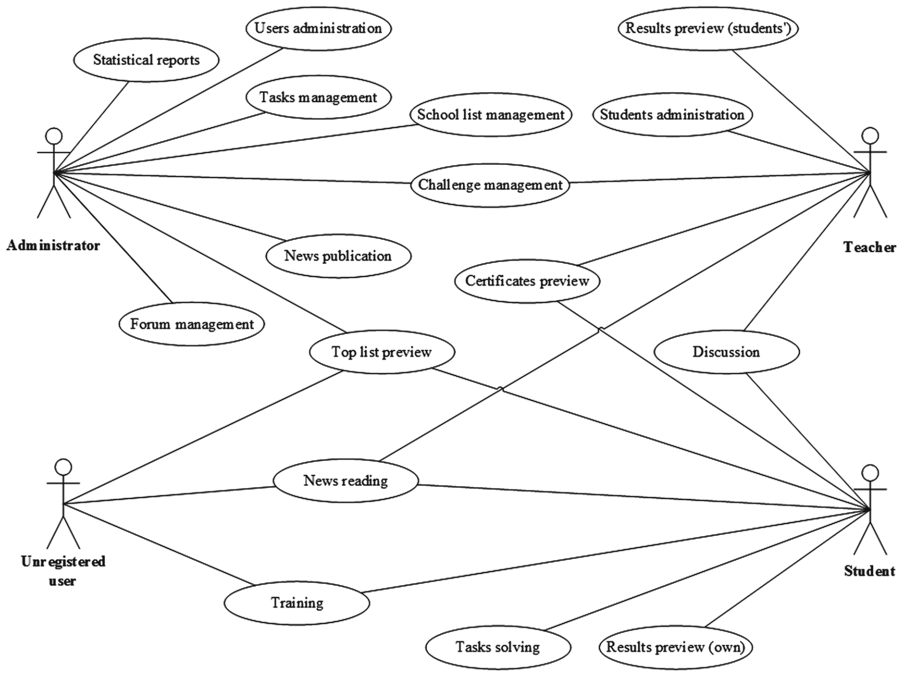


Fig. 3. Use case diagram of the Lithuanian Bebras CMS

CMS design is modular (consists of 11 modules) according to its functionality. **Users management module** includes actions such as create, edit, confirm, delete user, preview or edit data, preview the list of participants, archive participants, give a new password, move student to a higher class, search user, send news and reminders for the participant, give permission for the teacher to coordinate the challenge, and generate the reports for students' solutions.

Challenge management module involves these actions: create and edit the task set, set the challenge date and time, preview task in the task set, task testing, manage permission to solve task set, participation in the challenge, and answer survey about particular tasks.

System administration module consists of previewing the list of schools, editing participants' and teachers' data, searching (by school title, teacher name/surname, student name/surname), and creation of survey for participants.

Tasks management module is designed to create, edit, copy, upload, delete, preview, export task, filter task by tags, create tags for the tasks, and search tasks by name or ID. System supports multiple-choice questions with the opportunity to select one correct answer from four. Other questions, such as animated tasks, text input field tasks, and drag and drop tasks are imported from the *Bebras Lodge* tool. For multiple-choice tasks, the administrator fills these fields: task ID, title, task description and possible answers, answer comment if it is needed, chosen task difficulty, age group, and language.

Results preview module provides the opportunity to revise the desired student, class solutions. Tasks are shown with the marked correct/incorrect answer, students' choice, and point. Challenge results for registered participants are stored and teachers can view them.

Statistical and report module is used to review data about the time taken to solve the tasks, answers, and the number of participants. We collect the following data about participants: name, surname, gender, grade, and school. Also, we gather data about the type of devices and browsers the participants use in the challenge; how much time spent when solving the task for the first time; how much time in seconds the participant spends on every task (the sum of seconds is counted if the participant returns to solve the task). Each solution is separated by the student ID number generated by the system, therefore all statistics can be compared. In addition, data about the students' numbers in Lithuanian schools are saved in the system as well, so we can compare how many students in a particular school participated.

Front-page. Module includes login to the system, a preview of participants' results according to the municipality, age group, and a report about participants' numbers in Lithuania. Training is available on the front-page without any special registration. The time of training is limited to sixty minutes and does not depend on the student's age. There is no limit to the number of answer submissions. Training results for unregistered users are displayed immediately after finishing the session.

Authentication. Module implements the user registration, login and log-out function, and password reminder. Participants are able to login with a *Facebook* account. When a user registers with the system for the first time, the system automatically sends an email with the link, and the user has to approve it. An email is a required field on the system's registration form for purposes of authentication and personal data. Primary school pupils are faced with the problem of not having a personal email account. For this reason, their teachers can upload the list of participants to the system. The system generates passwords and user names automatically from the uploaded file.

Certificate module. The system automatically generates the filled certificate that is prepared by an administrator after the challenge is completed. Teachers get certificates for organizing and coordinating the challenge. Students who participated in the first or second round of the challenge receive certificates for participation. About 10% of the best students from each age group get winners' diplomas. These diplomas are printed and students receive them during the awards ceremony.

Language module enables localization and adapts the system for a specific language by translating resources. Using **discussion module** users can follow discussions (read, comment, delete, edit). As helpdesk email, phone, and *Facebook* are used for communication purposes.

The special system feature is the management of two challenge rounds. The differences between the first and second round is the permission to solve tasks managed by the system administrator. During the first round, the local organizer (teacher) is responsible for the students and marks students who can solve tasks. It can be done only during the scheduled time. Before the second round, the system administrator has to mark the students who are involved in the challenge. Students get email with the confirmation link and students, who confirm their email, solve tasks during the second round. In addition, task sets of the first and second round are separated, therefore the system administrator can easily follow the history of a task set (when and which tasks were used in the challenge). The Bebras CMS is used as the questions pool. About 2000 tasks are collected in the system.

4.3 Practicalities

The Lithuanian Bebras CMS is used as a tool for contest management, but sometimes we demonstrate the system for teachers or systems developers to provide them with practice, understanding the challenge policy, and teacher training. Unfortunately, the Lithuanian system does not provide teachers with the possibility of preparing their own challenge to use in the educational process. If they want to practice tasks development, we recommend a *Bebras Lodge* tool.

Each year a systems developer adds new functionalities to the system or improves existing components. For example, there was a new opportunity in 2016 to upload the participants' list. Next year we are planning to introduce the badge policy as a participation engagement tool. Students will be able to earn badges depending on the collected points. We hope to have four types of badges: gold, silver, bronze medals, and a certificate of participation. Badges do not always affect learners' motivation as an inspiration to reach better results or be involved in the learning process, but we hope it will work as an award that all students will receive. The second addition to the system will be a two-dimensional system, which will enable the classification of tasks according to computational thinking and informatics concepts. Detailed analysis of the new Bebras tasks classification is provided by [12].

5 Conclusions

This paper discusses CMSs as a tool to manage the Bebras challenge in various countries. Answers from thirty-two (out of a total of 39) countries running the Bebras challenge were collected and analysed. The results show that 18 counties developed their own CMSs. 12 countries used only multiple choice or open-ended answers for the Bebras tasks, and 18 countries provided interactive tasks. In the future, some countries would like to use more interactive task features, as well as develop compatibility with mobile devices. The Lithuanian Bebras CMS was developed in 2010. The system consists of eleven modules, and has a three-layer architecture. One of the future plans is integration a two-dimensional system for enabling the classification of tasks according to computational thinking and informatics concepts.

Acknowledgement. We acknowledge the international Bebras Community as well as Centre of IT Development of Vilnius University for support. Thanks Carol Sperry, our longtime colleague at informatics education, for proofreading the paper and making suggestions.

References

1. Benaya, T., Zur, E., Dagiene, V., Stupuriene, G.: Computer science high school curriculum in Israel and Lithuania - comparison and teachers' views. Baltic J. Modern Comput. **5**(2), 164–182 (2017)
2. Dagiene, V., Stupuriene, G.: Bebras-a sustainable community building model for the concept based learning of informatics and computational thinking. Inf. Educ. **15**(1), 25–44 (2016)
3. Kostadinov, B., Jovanov, M., Stankov, E.: A new design of a system for contest management and grading in informatics competitions. In: ICT Innovations 2010, Web Proceedings, pp. 87–96 (2010)
4. Maggiolo, S., Mascellani, G.: Introducing CMS: a contest management system. Olympiads Inf. **6**, 86–99 (2012)
5. Jovanov, M., Kostadinov, B., Stankov, E., Mihova, M., Gusev, M.: State competitions in informatics and the supporting online learning and contest management system with collaboration and personalization features MENDO. Olympiads Inf. **7**, 42–54 (2013)
6. Kristan, N., Gostiša, D., Fele-Žorž, G., Brodnik, A.: A high-availability bebras competition system. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 78–87. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_8
7. Gostisa, D.: Doctoral dissertation: Visoko razpolozljiv tekmovalni sistem v oblaku za tekmovanje Bober-CaaS, Univerza v Ljubljani (2014)
8. Pozdniakov, S.N., Kirynovich, I.F., Posov, I.A.: Contest "Bebras" on informatics in Russia and Belarus. Olympiads Inf. **10**(Special Issue), 55–65 (2016)
9. Pikl, B., Kristan, N., Ham, Z., Brodnik, A.: Developing integrated ACM competitions website. In: International Electrotechnical and Computer Science Conference (2012)
10. Kalelioğlu, F., Gülbahar, Y., Madran, O.: A Snapshot of the first implementation of Bebras international informatics contest in Turkey. In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 131–140. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_12
11. Leff, A., Rayfield, J.T.: Web-application development using the model/view/controller design pattern. In: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2001), pp. 118–127. IEEE Computer Society, Washington (2001)
12. Dagiene, V., Sentence, S., Stupuriene, G.: Developing a two-dimensional categorization system for educational tasks in informatics. Informatica **28**(1), 23–24 (2017)

Author Index