

**Erik Barendsen
Christos Chytas (Eds.)**

LNCS 13057

Informatics in Schools

Rethinking Computing Education

**14th International Conference on Informatics in Schools:
Situation, Evolution, and Perspectives, ISSEP 2021
Virtual Event, November 3–5, 2021, Proceedings**

 **Springer**

Founding Editors

Gerhard Goos

Karlsruhe Institute of Technology, Karlsruhe, Germany

Juris Hartmanis

Cornell University, Ithaca, NY, USA


Editorial Board Members

Elisa Bertino

Purdue University, West Lafayette, IN, USA

Wen Gao

Peking University, Beijing, China

Bernhard Steffen 

TU Dortmund University, Dortmund, Germany

Gerhard Woeginger 

RWTH Aachen, Aachen, Germany

Moti Yung 

Columbia University, New York, NY, USA

More information about this subseries at <http://www.springer.com/series/7407>

Erik Barendsen · Christos Chytas (Eds.)

Informatics in Schools

Rethinking Computing Education

14th International Conference on Informatics
in Schools: Situation, Evolution,
and Perspectives, ISSEP 2021
Virtual Event, November 3–5, 2021
Proceedings

Editors

Erik Barendsen 
Radboud University
Nijmegen, The Netherlands

Christos Chytas
Radboud University
Nijmegen, The Netherlands

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-030-90227-8 ISBN 978-3-030-90228-5 (eBook)
<https://doi.org/10.1007/978-3-030-90228-5>

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© Springer Nature Switzerland AG 2021

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains selected papers presented at the 14th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2021). The conference was hosted by Radboud University (Nijmegen, the Netherlands). It took place during November 3–5, 2021 and was held online due to the COVID-19 pandemic.

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 aspects of teaching and learning in informatics in a broad sense, including the various ways of implementing informatics education in schools.

The conference topics include informatics curricula, programs, and courses, pedagogies for teaching and learning informatics, teacher education in informatics, digital literacy and computational thinking, connecting informatics education in formal, non-formal and informal learning environments, informatics and digital literacy in contexts, (e.g., STEAM education), digital citizenship, engagement and empowerment strategies for broadening participation and diversity in computing education.

The ISSEP conference started in 2005 in Klagenfurt. It was followed by meetings in Vilnius (2006), Torun (2008), Zürich (2010), Bratislava (2011), Oldenburg (2013), Istanbul (2014), Ljubljana (2015), Münster (2016), Helsinki (2017), Saint Petersburg (2018), Larnaca (2019), Tallin (2020), and this year's ISSEP held virtually in Nijmegen (2021).

This year's ISSEP conference received 42 submissions in various categories. Only the full paper categories (country reports and research papers) were considered for publication in these LNCS proceedings. At least three reviewers evaluated the quality, originality, and relevance to the conference of the 29 full paper submissions. The review was double-blind. The committee selected 12 papers for publication in these proceedings, leading to an acceptance rate of 41%.

We would like to express our gratitude to all authors, reviewers, and local organizers for their valuable contribution to this year's ISSEP conference.

September 2021

Erik Barendsen
Christos Chytas

Organization

Steering Committee

Erik Barendsen	Radboud University and Open University, The Netherlands
Andreas Bollin (chair)	University of Klagenfurt, Austria
Valentina Dagiene	Vilnius University, Lithuania
Yasemin Gülbahar	Ankara University, Turkey
Juraj Hromkovič	Swiss Federal Institute of Technology Zurich, Switzerland
Ivan Kalas	Comenius University, Slovakia
Sergei Pozdniakov	Saint Petersburg Electrotechnical University, Russia

Program Chairs

Erik Barendsen	Radboud University and Open University, The Netherlands
Christos Chytas	Radboud University, The Netherlands

Program Committee

Andreas Bollin	University of Klagenfurt, Austria
Andrej Brodnik	University of Ljubljana, Slovenia
Špela Cerar	University of Ljubljana, Slovenia
Valentina Dagiene	Vilnius University, Lithuania
Christian Datzko	University of Applied Sciences and Arts Northwestern Switzerland, Switzerland
Monica Divitini	Norwegian University of Science and Technology, Norway
Gerald Futschek	Vienna University of Technology, Austria
Juraj Hromkovic	ETH Zurich, Switzerland
Mile Jovanov	Ss. Cyril and Methodius University, Serbia
Dennis Komm	PH Graubünden and ETH Zurich, Switzerland
Mart Laanpere	Tallinn University, Estonia
Peter Larsson	University of Turku, Finland
Marina Lepp	University of Tartu, Estonia
Inggriani Liem	STEI ITB, Indonesia
Birgy Lorenz	Tallinn University of Technology, Estonia
Piret Luik	University of Tartu, Estonia
Maia Lust	Tallinn University, Estonia
Kati Mäkitalo	University of Oulu, Finland
Tilman Michaeli	Freie Universität Berlin, Germany

Mattia Monga	Università degli Studi di Milano, Italy
Arnold Pears	KTH Royal Institute of Technology, Sweden
Hans Põldoja	Tallinn University, Estonia
Ralf Romeike	Freie Universität Berlin, Germany
Joze Rugej	University of Ljubljana, Slovenia
Giovanni Serafini	ETH Zurich, Switzerland
Tomas Šiaulys	Vilnius University, Lithuania
Gabrielė Stupurienė	Vilnius University, Lithuania
Reelika Suviste	University of Tartu, Estonia
Maciej Sysło	UMK Torun, Poland
Michael Weigend	WWU Münster, Germany

Local Organization

Erik Barendsen	Radboud University and Open University, The Netherlands
Christos Chytas	Radboud University, The Netherlands
Nataša Grgurina	University of Groningen, Radboud University, and SLO, The Netherlands
Simone Meeuwsen	Radboud University, The Netherlands
Mara Saeli	Radboud University, The Netherlands

Contents

Fostering Computational Thinking

- Computational Thinking in Context Across Curriculum: Students’ and Teachers’ Perspectives 3
Nataša Grgurina and Sabiha Yeni
- Towards Classification of Interactive Non-programming Tasks Promoting Computational Thinking. 16
Tomas Šiaulys and Valentina Dagienė
- Tell, Draw and Code – Teachers’ Intention to a Narrative Introduction of Computational Thinking. 29
Karin Tengler, Oliver Kastner-Hauler, and Barbara Sabitzer

Programming Education

- First Programming Course in Business Studies: Content, Approach, and Achievement 45
Djordje M. Kadijevich
- Why Young Programmers Should Make Game Art: A Study from a Game-Making Course. 57
Mária Karpielová and Karolína Miková
- Teaching Recursion in High School: A Constructive Approach. 69
Dennis Komm

Advancing Computing Education

- A Multi-dimensional Approach to Categorize Bebras Tasks 83
Christian Datzko
- Girls’ Summer School for Physical Computing: Methodology and Acceptance Issues 95
Gabrielė Stupurienė, Anita Juškevičienė, Tatjana Jevsikova, Valentina Dagienė, and Asta Meškauskienė
- Towards a Compulsory Computing Curriculum at Primary and Lower-Secondary Schools: The Case of Czechia. 109
Jiří Vaniček

Teachers' Professional Development

Professional Development for In-Service Teachers of Programming:
Evaluation of a University-Level Program 123
*Majid Rouhani, Miriam Lillebo, Veronica Farshchian,
and Monica Divitini*

Encouraging Task Creation Among Programming Teachers
in Primary Schools 135
*Jacqueline Staub, Zaheer Chothia, Larissa Schrempp,
and Pascal Wacker*

Problems, Professional Development and Reflection: Experiences
of High-School Computer Science Teachers in Serbia 147
Vojislav Vujošević

Author Index 161

Fostering Computational Thinking



Computational Thinking in Context Across Curriculum: Students' and Teachers' Perspectives

Nataša Grgurina^{1,2}  and Sabiha Yeni³ 

¹ Radboud University, Nijmegen, The Netherlands

natasa.grgurina@ru.nl, n.grgurina@rug.nl

² Teaching and Teacher Education, University of Groningen, Groningen, The Netherlands

³ Leiden University, Leiden, The Netherlands

s.yeni@liacs.leidenuniv.nl

Abstract. Integration of computational thinking (CT) into numerous disciplines across the K-12 curriculum is gaining increased attention. In this study, based on the technology integration framework, we investigated how students' understandings, difficulties, and attitudes towards learning subject matter varied at different levels of CT integration. We implemented six different case studies by integrating CT into six different subjects: science, traffic, language, biology, geography, and physics. Two primary and four secondary school teachers, 38 primary school students, and 113 secondary school students were involved in the study. We categorized these lessons according to the technology integration model: unplugged activities are grouped as augmentation level; robotic and two modeling activities are labeled as modification level; modeling and digital storytelling activities are labeled as redefinition level. Our findings indicate that students reported a very positive attitude toward redefinition level activities. Teachers stated that compared to standard instructional activities, students can go deeper and understand the subject content better in CT integrated lessons.

Keywords: Computational thinking · SAMR model · Integration · K-12 · Students' attitude · Teachers' attitude · Computational modeling

1 Introduction

One of the increasing trends in the current educational systems is the call for the integration of digital literacy, and in particular its aspect Computational Thinking (CT), into the curricula [6, 27]. CT provides students with a new set of skills for problem-solving and thinking about the world. Computer Science courses are often considered as regular lessons for teaching CT, however there is an increasing interest to see CS as “a truly interdisciplinary undertaking” [26] for other disciplines [14]. Wing [28] defined the CT term in this context as “... 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 [...]. These solutions can be carried out by

any processing agent, whether human, computer, or a combination of both” [10]. While there are many definitions of the CT problem solving process, they all share a common view that it involves three steps: translating the problem under scrutiny into computational terms, constructing a computational solution, and finally, using that computational solution in the domain where the problem originates [3]. Even individuals who will not become programmers are expected to acquire some universal competencies such as the ability to solve problems in daily life, break down complex problems into components, and generalize solutions.

There are various approaches that guide the development of students’ computational thinking and subject skills. We used the elements of active learning in all cases of our study. Integrating CT concepts through **digital storytelling** is an effective approach that is used particularly in language classrooms to promote subject-related skills and CT skills [11, 18, 23]. Another approach for integrating CT into an interdisciplinary field is through **computational modeling** which is used in different contexts such as science, social sciences etc. [4, 15]. Computational modeling has been used in computer-based learning environments to model learners’ subject related knowledge, cognitive skills, and customize their experiences in the environment based on this information [4]. **Robotics** construction kits have been designated as a promising way to introduce CT processes to students in K-12 schools because it motivates students and scaffolds the learning of programming—novices create programs by initially manipulating visual blocks on the computer screen and can then be guided to create more complex programming [2, 13]. A meta-analysis of studies where robotics is used to promote STEM learning revealed that, generally speaking, the use of educational robotics increased students’ learning of specific STEM concepts [5]. Finally, **unplugged activities** can be a good approach to introduce students, especially young learners, to CT [20]. Regarding the integration to curriculum, teaching science concepts using CT unplugged approach may provide an opportunity for the students to strengthen their foundation in CT skills [7] and improve their understanding in science concepts [24] simultaneously.

The effectiveness of the CT integrated lessons was examined in educational intervention studies with different aspects such as students’ comprehension level about CT skills [8] or level of understanding about subject related objectives [25] or attitude and satisfaction of students toward CT integrated lessons. With respect to the attitude of students, for example, studies explore (increased) interest in and perception of STEM and CS [8, 17] and interest to pursue a CS degree [16]. Further examples focus on affective aspects such as enjoyment in the activity [12], learning experience [22], motivation [19], ownership [21], and difficulties [1].

In this study, in line with the different levels of technology integration framework, students’ progress from completing challenges posed by teachers or modifying existing products to taking an active role as developers of their own designs that make use of computational tools. Puentedura [25] proposes SAMR framework that classified technology integration into learning activities into four levels. The substitution degree is the lowest step at the enhancement level, where CT concepts, models, or associated technologies are directly substituted for more traditional ones. The next degree is the augmentation degree where the technology acts as a direct technology substitute with functional improvement. At the higher level, i.e., *transformation level*, the *modification*

degree signifies that the technology use allows significant task redesign. Finally, at the highest level in this framework—the *redefinition degree*—the use of technology allows for the creation of new tasks that were previously inconceivable.

This study is part of a larger project aiming to examine the characteristics of a successful continuous learning trajectory for the integration of computational thinking into the K-12 curriculum across the disciplines of sciences, humanities, and languages. In cooperation with participating teachers, we developed lesson series and carried them out during the first phase of the project. In these lessons, students engage in active learning with the different levels of integration of the SAMR framework. The aim of this study was to characterize the perceived understanding, difficulties, and attitudes of the students participating in CT integration studies at various degrees of technology integration. We look at these issues from the students' and the teachers' perspectives.

2 Methods

2.1 Participants

In this study, the participants from both primary and all four secondary schools participating in the larger project were involved: 38 primary school students, 113 secondary school students and one teacher from each school. The teachers are male. Two primary school teachers, (T2, T4 and T5) are project leaders for digital literacy projects. The age range of students is between 9 and 14. There are 59 female students and 88 male students, four students did not mention gender.

2.2 Lesson Design

In each school, a different lesson was implemented. The participating teachers followed a workshop where they were presented with examples of lessons integrating CT with the subject matter of disciplines across the breadth of the curriculum: science, humanities, languages etc. Then each teacher individually worked with the researchers to develop a lesson series about a topic they planned to teach. These lessons were then taught instead of traditional lessons. We categorized these lessons according to the SAMR model. The lessons start with the augmentation stage.

Augmentation Degree

Climate case. In the geography lessons, the 8th grade students were learning about the Köppen climate classification¹. During three lessons, they learned about the characteristics of the main climate groups and how to determine the climate type from the temperature and precipitation data. The teacher chose an unplugged approach and students were asked to write down the climate determination process in the form of a decision tree. Due to Covid measures, the lessons were carried out as online lessons.

Modification Degree

Self-driving cars case. In this primary school, the 5th grade students were learning about

¹ <http://koeppen-geiger.vu-wien.ac.at/koeppen.htm>.

self-driving cars and corresponding ethical questions. In the first lesson, the students drew a car with sensors needed to participate in traffic safely, and were asked to write a short algorithm for each sensor. Additionally, they were asked to program a proximity sensor on a mBot car by putting the provided code blocks in the correct order. In the second lesson, the students discussed the advantages of self-driving cars and a number of ethical dilemmas arising from traffic situations. Additionally, they were asked to extend their mBot program to have it follow a line and play a sound—again by putting the given program blocks in the correct order.

Gas exchange case. The 8th grade students were learning about the human body, and in particular, about gas exchange. The researchers provided a simple gas exchange model in Scratch. During three lessons, the students were using it to learn about the oxygen saturation of blood which depends on the type and quantity of the gas being inhaled and the exertion rate. Additionally, the students were asked to adjust the model's program code.

Stopping distance case. The 8th grade students were learning about the stopping distance of a moving vehicle which depends on the road conditions and the reaction speed of the driver. The first lesson was dedicated to theory. In the subsequent two lessons, the students used a simple model in Scratch provided by their teacher and researchers to explore the influence of icy roads, drinking and distraction on the stopping distance. Additionally, the students were asked to adjust the model's program code.

Redefinition Degree

Cell division case. In this primary school case, the 5th grade students learned about the cell division. During the two introductory lessons, they learned about the theory of cell division, watched videos and observed cells under a digital microscope. In the subsequent two lessons, students worked in pairs on a practical assignment: they wrote down all they knew about cell division, they described the corresponding algorithm and developed computational models—i.e. visualizations—of cell division in Scratch. During programming, their class teacher and their programming teacher were helping them when necessary. At the end of the last lesson, each pair of students presented their program in front of the class.

Digital storytelling case. In the course of English as foreign language, the 7th grade students studied to apply the acquired grammar and vocabulary in their digital stories. In the introduction lessons they learned to ask and answer questions about themselves. In the subsequent two lessons, they created their own digital stories to introduce themselves. Students are given an explanation about the goals of the digital story and they made planning. They worked in pairs to create storyboards, got feedback from the teacher about the scripts, and created their digital stories in Scratch. They recorded their voices and added them to their digital stories in order to vocalize the dialogues of the characters. During the programming, their class teacher, researcher and their friends helped where necessary.

2.3 Data Collection

Before starting their work on the projects, the students filled in a **profile survey** where they were asked about their technology ownership, programming background, and experience regarding their familiarity and experience with specific types of programming languages. Furthermore, they were asked about their self-efficacy related to technology and programming, and finally their age and gender. The **practical assignment** consists of a series of questions that guide and document students' activities during their work on the project. The students answer the questions individually to express their problem in computational terms and to construct the computational solution. Additionally, they reflect on the project. **Exit tickets** are small questionnaires given to students at the end of each lesson or at the end of a lesson series. There, the students indicate on a three-point Likert scale whether they liked the lessons (series), whether they found the lessons interesting, whether they understood what they had to do in the lessons, whether they understood the teaching materials, and finally, whether they found the lessons difficult. Additionally, students could write comments about the lessons. **Interviews.** At the end of the project, we interviewed a number of students and all the participating teachers. During the semi-structured interviews which lasted about ten minutes, we asked about the programming practices they employed while working on their projects—which are reported elsewhere—and about their attitude towards technology use, which we report here. We conducted semi-structured interviews with each teacher individually. These interviews lasted about twenty minutes. We asked the teachers to compare their experiences teaching the lessons during the project—thus, making use of CT—to their standard manner of teaching the same content, and about their attitude towards the integration of CT in the lessons. In this report, we focus on their views on students' conceptual understanding, their attitudes and difficulties. All the interviews were recorded and transcribed verbatim.

2.4 Data Analysis

We performed quantitative and qualitative data analysis. The data from exit tickets and some questions from the practical assignments were analyzed through descriptive statistics. For other data, we used Atlas.ti software as a qualitative data analysis tool to code our data. The data were first categorized deductively according to our research goal, in line with the issues the research instruments were concerned with. Within these categories, we applied inductive coding to further characterize the data. In an axial coding process [9], the codes were grouped and merged where necessary under new categories.

3 Results

3.1 Students

Student Profiles. The most commonly used programming languages are block-based languages that 89 students (59%) indicated to use; in the second place are robotic tools that 41 students (27%) stated that they used. The least popular programming languages are text-based programming languages with 30 students (20%). The most commonly

reported programming languages are Scratch, Microbit and Python. As a whole, 119 students (79%) had programming lessons in the past, 12 students (8%) have never had programming lessons but they tried to learn programming by themselves and 20 students (13%) have never had programming lessons before. They were asked how long they had taken programming lessons. In overall, 48 students (32%) had programming lessons less than one month; 43 students (28%) had programming lessons for one year; 32 students (21%) had 2 years to 3 years, and only 3 students (2%) had programming lessons longer than 3 years. Judging their programming experience on the 10-point Likert scale from 1 (no experience) to 10 (very experienced), they score a minimum of 1, a maximum of 9 and a mean of 5,9. Overall, about half of the students (42%) feel comfortable with programming, the other (55%) of students would need at least some help, and 3% of students see themselves as a professional regarding programming (Table 1).

Table 1. The programming experience of students according to different cases

	Cases (teachers)	Programming lesson			Prog. lesson duration				Prog. experience
		Yes	No	Learn by myself	<1 month	1 m to 1 year	2 to 3 years	>3 years	
Aug	Climates (T1)	95	0	5	58	26	11	5	6,2
Mod	Self-driving cars (T2)	100	0	0	63	25	0	13	4,7
Mod	Gas exchange (T3)	63	25	13	65	35	0	0	5,3
Mod	Stop distance (T4)	47	47	6	67	11	11	11	4,8
Red	Cell division (T5)	100	0	0	8	0	92	0	6,7
Red	Digital story (T6)	80	9	11	17	68	15	0	6,5

Exit Tickets. Students filled their exit tickets at the end of each lesson or at the end of the lesson series. We analyzed the following categories: Enjoyment: I enjoyed the lesson; Interesting: The lesson was interesting; Clarity: I knew what to do; Comprehension: I understood the subject matter; and Difficulty: The lesson was difficult (Table 2). The answers were on a three-point Likert scale: Yes (Y), Maybe (M) and No (N). We calculated the relative frequency of the answers for each case. According to the students'

answers, the most enjoyable lessons are the case of cell division (92%) and digital story (91%) and most of the students in these cases (88% and 77% respectively) also found these lessons interesting. The answers of students about clarity are similar to each other between groups (change between 83% and 66%); only in the case of gas exchange, 43% of students declared that they didn't know what to do. With respect to comprehension, the most remarkable answer belongs to the case of climates. While most of the students (91%) in this unplugged case state that they understand the subject (climates in geography) well, even if the number of those who have fun and find the lesson interesting is not very high compared to other cases. Regarding the difficulty of the lessons, students in the case of climates found the lesson series easy (70%).

Practical Assignments. Students from three schools turned in their practical assignments: those working on the stopping distance project, on the cell division projects, and on the digital storytelling projects. We report on students' reflecting on their projects.

Table 2. The mean percentages of exit tickets

	Cases	Enjoyment			Interest			Clarity			Compr			Difficulty		
		Y	M	N	Y	M	N	Y	M	N	Y	M	N	Y	M	N
Aug	Climates	55	43	2	51	40	4	66	30	4	91	6	2	4	26	70
Mod	Self-dr. cars	85	13	3	72	26	3	77	18	5	69	28	3	8	38	54
Mod	Gas exchange	57	35	9	46	45	10	49	28	15	45	41	14	16	43	40
Mod	Stopping dist	50	25	25	38	44	19	69	19	19	69	19	6	19	25	44
Red	Cell division	92	8	0	88	13	0	83	13	4	79	21	0	0	63	33
Red	Digital story	91	9	0	77	16	8	75	21	3	70	25	5	11	25	64

Regarding the satisfaction with their projects, the majority of the students reported they were satisfied. One student is satisfied with some parts of the project but also mentions aspects they are not satisfied with. Some students offer suggestions to improve their project, for example, by testing their program more thoroughly, and some comment they had not finished their projects. We asked students what they enjoyed most in the lessons. The most frequently reported issue is programming or editing an existing program, mentioned by more than half of the students. Only students working on the cell division project mentioned subject matter, such as, for example, using the microscope to look at cells. In each class, there were students reporting they enjoyed cooperating with each other. Two students found nothing to be enjoyable in these lessons and found them to be boring. When asked what they enjoyed least, 15 students reported they actually enjoyed everything. Overall, about half of the students actually report they disliked something

in the lessons, and about half of those refer to organizational aspects of the lessons such as the quality of the instruction, but also having to wait for too long before they were allowed to begin with programming. Furthermore, the students mention programming issues: six students reported they disliked programming while another three reported had too few opportunities to program. Eleven students reported disliking subject-related issues, such as working with a microscope or having to record their voices. Finally, one student dislikes having to articulate and write down their thoughts.

Regarding what the students found to be the most difficult, one student reported everything was difficult and ten students found nothing to be difficult. Out of 24 students who found programming to be difficult, 11 consider it to be the most enjoyable aspect of the project as well. A total of 10 students experienced difficulties related to subject matter—for example, working with a microscope. Eleven students report difficulties related to their creativity (for example, coming up with sentences for their stories) and to thinking (for example, implementing an idea exactly as imagined in their mind).

We asked the students whether they think working on this type of a project (i.e., with computational thinking embedded in the context of the subject matter) helped them to learn subject matter better. Students told us that, for example, this work helped them because learning this way is more fun and so they worked better; but also, that it did not help, “because it was totally different than a school subject”. Finally, we asked whether they would like to work on such a project in the future and if so, what subject matter they consider suitable. The students mention math, languages, physical education, and a few believe it would suit all subjects. The elementary school students show a more positive attitude on both of these issues.

Interviews. After the lesson series finished, we conducted 24 student interviews with students from five schools: three from the self-driving cars project, nine from the gas exchange project, three from the stopping distance project, two from the cell division project, and seven from the digital storytelling project. In the data analysis, we focused on their own perspective on the lesson activities and their learning. A common theme many students mention is their appreciation of collaboration with other students. The primary school students (those working on the self-driving cars and the cell division projects) unanimously agreed they *liked* this type of lesson and believed they *learned subject matter better*. One of the students explained that he *liked* working this way and therefore was *more motivated to learn* about cell division. Another student said they *gained more insight* into the inner workings of a self-driving car after having programmed a sensor for such a car. The students working on the stopping distance project are divided on the issue. While one student said, “Now I *understand* the stopping distance, because you *learn in a fun way* and then you remember it better”, another one was *critical about using a Scratch model* to help them learn: “Of course I know you shouldn’t look at your phone while driving, but I *haven’t learned much*.”

The students working on the gas exchange project are divided on the issue as well and place a number of thoughtful remarks. Several of them said *it was nice to do something new* for a change; however, some warned that this sort of game-like activities could be a *distraction*. Regarding their learning, one student commented that *this way of learning helps* because they get to examine the issue at hand from various perspectives. Another student adds that it was nice to be able to change the values of the parameters in the

model and see the effect on the outcomes. Yet, as one of the students notices, while the *model helped them to learn* about gas exchange, the visual representation of gas molecules and blood cells in the model was not realistic.

Finally, the students working on the storytelling project reported that they *liked working this way*, for example, because working from books only is boring. One of the students said programming a conversation in Scratch *helped them learn English better* because it made them speak in English; however, another student remarked their English was already good and they *did not learn much*.

3.2 Teachers

After the lesson series finished, we interviewed all six teachers. We present the findings regarding students' conceptual understanding, their attitudes and difficulties. The results regarding all of the teacher's replies regarding all pedagogical content knowledge (PCK) elements are reported in another extended study [29].

Students' Conceptual Understanding. This refers to teachers' knowledge and understanding of students' conceptual knowledge and knowledge gaps. Four teachers [T1, T3, T4, T5] reflect on their knowledge regarding the conceptual understanding of students. Regarding the students' understanding of the subject related concepts, the common view of four teachers is that students understood the subject more deeply in CT integrated lessons than in the traditional lessons.

For the augmentation level of the climate case, the teacher said, "*The students generally have quite a lot of trouble with that [to understand climate related learning objectives] The penny does not drop quickly, and that often takes a lot of time. The learning objective was actually to do that in a special way [CT integration], and we did to make it clear.*" [T1] For the modification level of the "gas exchange" case, the teacher commented, "*I think most of them, 80 to 90% knew what they were doing [...] it was not only Scratch but also biology.*" [T3] In the case of stopping distance, "*A whole group actually got to work very fanatically to adapt it [model]*" [T4]. For the redefinition level of the "cell division" case, the teacher remarked, "*If you look at the grade 5 level [of the cell division], I think they have learned enough for this year [...] I thought this [CT integration] might be a way to make this indeed more understandable for the kids [Regarding the CT concepts] Everyone had programmed something in Scratch, they could do that quite well. No matter how basic it was. Programming in Scratch and making the connection between the steps you have to take in Scratch, well I thought that went pretty quickly.*" [T5].

Students' Reaction and Perception. This refers to the teachers' knowledge and awareness of students' reactions and perceptions toward CT integration lessons. The teachers involved in the lesson of all integration levels stated that students' perceptions were encouraging and positive, and they said that with these terms: like, enjoy, enthusiastic, exciting, interesting, fun, educational, voluntary [T1, T2, T3, T4, T5, T6]. Only the teacher of the climate case (the augmentation level) [T1] noticed that two girls who went to the same primary school, and they indicated that they didn't like it. The teachers said,

“I think they [students] liked it. [...] I think what I saw was that they were working on it with enthusiasm.” [T5] “Actually, in my experience, kids always like it. They enjoy working with such a robot. It is often new to them too [...] They often get excited about that anyway.” [T2] “A lot of them said that they liked it a lot. They thought it was hard but they said that was interesting in their opinions [...] I think they did enjoy it.” [T6] “You can really see that they have a lot of work on it but they also seemingly had a lot of fun doing it” [T1] “A whole group actually got to work very fanatically to remix it.” [T4] “I had the idea that they [students] liked it. [...] I do have the feeling that it has been educational and fun.” [T3].

Students’ Difficulties. According to teachers, students experience various sorts of problems during the lessons, including those related to: understanding the programming concepts (conditions, design of algorithm, decision tree) [T1, T2, T5], requiring a lot of skills/knowledge to accomplish task (such as subject-matter knowledge, programming knowledge, collaboration skills, planning skills etc.) [T6], creating new things [T4], and understanding instruction [T3].

For the augmentation level of the climate case, the teacher said, *“They all found it easy. That is unusual, you always have many questioners among students... Questions were asked about the content, and that was quite unusual. They really had some difficulty with the concept of the decision tree, because I had introduced it fairly quickly.” [T1].* For the modification level of the self-driving car case, the teacher commented, *“They found the assignment more difficult to design a short algorithm for the sensors. They just had to write that down [...] They had a lot of trouble with that.” [T1]* For the redefinition level of the digital storytelling project, the teacher said, *“There are a lot of required skills you need to know besides the English stuff. You also need to make a plan, to be able to work together and also then the whole Scratch such as how you program those things. I guess that was the hardest part for them” [T6].* For the cell division case, the teacher remarked, *“They [students] found it very difficult to get conditions for cell division to actually get there [...] They did not progress beyond one condition to achieve cell division.” [T5].*

4 Conclusion and Discussion

In this study, we explored the perceived understanding, difficulties and attitude of the students participating in CT integration studies at various degrees of technology integration, from the students’ and teachers’ perspective. Regarding students’ understanding and difficulties related to the subject matter concepts, the teachers indicated that students understood the subject more deeply in CT integrated lessons than in the traditional lessons. The students themselves indicated that, for the most part, they did not find the lessons difficult and very few of them indicated having had difficulties understanding the subject matter. We conclude that in the lessons where CT was integrated with subject matter, the students did not experience a lot of difficulties and that participating in this type of lessons was beneficial for the deeper understanding and learning of the subject matter.

Regarding the students' understanding and difficulties related to the CT, the teachers reported about their students' specific difficulties related to programming concepts. The students themselves indicated having found programming to be difficult in less than half of the cases. Furthermore, some students experience difficulties related to other issues such as understanding instruction, planning their work and cooperation.

Regarding the students' attitude and their perception of these types of lessons, the teachers report a positive attitude of their students, and the students themselves corroborate this finding. The students reported enjoying these lessons and being satisfied with their work. Notably, a number of students realized that they were better at programming than what they initially thought, or that they actually enjoyed programming. A detailed analysis of all findings reveals that the students who actively engaged in programming (i.e. those working on the projects involving self-driving cars, cell division and storytelling) enjoyed the lessons more and found the lessons more interesting than the students working on other projects. Yet, this outcome is not reflected in the students' opinion on the issue of whether these types of lessons help them to learn the subject matter better. While the primary school students (those working on the cell division project) almost unanimously believe that it does, and would like to engage in similar projects in the future, the students in secondary schools (those working on the stopping distance and storytelling projects) are divided and a majority of those answering this question disagrees.

Relating our findings to the integration levels of the SAMR framework [25] reveals that student engaging in lessons at the transformation level—i.e., modification and redefinition degrees—taken as a whole, tend to have a positive attitude towards this type of lessons. Again, in the projects where students actively engage in programming, they tend to express a positive attitude more often, and these students are exactly the ones who more frequently indicate that they have had programming lessons in the past. Their teachers were aware of the students' programming experience and prepared the lessons accordingly. This finding confirms the importance for students to learn computational thinking: the students who are adept at it have better chances to profit from the integration of CT with the learning of subject matter and to gain deeper understanding of subject matter. Indeed, such students engage in active learning and seem to experience multiple benefits when studying subject matter: expressing the problem under scrutiny in computational terms requires deep understanding of the problem, as does the construction of a computational solution too. Finally, using that computational solution in the original domains allows students to gain new insights [3].

In this study, we saw that integration of CT with subject matter benefits students' learning. Future research within the larger project where this study is a part of is going to expand the focus on measuring learning gains in learning of both subject-matter related and CT-related learning objectives.

Acknowledgements. This work is supported by The Netherlands Organisation for Scientific Research grant nr. 40.5.18540.153.

References

1. Almeida, R., et al.: iProg: getting started with programming: pilot experiment in two elementary schools. In: 2017 International Symposium on Computers in Education (SIIE), pp. 1–5 (2017)
2. Baek, Y., et al.: Revisiting second graders' robotics with an understand/use-modify-create (U2MC) strategy. *Eur. J. STEM Educ.* **4**, 1 (2019)
3. Barendsen, E., Bruggink, M.: Het volle potentieel van de computer leren benutten: over informatica en computational thinking (2019)
4. Basu, S., Biswas, G., Kinnebrew, J.S.: Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Model. User-Adap. Inter.* **27**(1), 5–53 (2017). <https://doi.org/10.1007/s11257-017-9187-0>
5. Benitti, F.B.V.: Exploring the educational potential of robotics in schools: a systematic review. *Comput. Educ.* **58**(3), 978–988 (2012)
6. Bocconi, S., et al.: Developing computational thinking in compulsory education. *Eur. Comm. JRC Sci. Policy Rep.* (2016)
7. Brackmann, C.P., et al.: Development of computational thinking skills through unplugged activities in primary school. Presented at the (2017)
8. Burgett, T. et al.: DISSECT: analysis of pedagogical techniques to integrate computational thinking into K-12 curricula. In: 2015 IEEE Frontiers in Education Conference (FIE), pp. 1–9. IEEE (2015)
9. Cohen, L., et al.: *Research Methods in Education*, 6th edn. Routledge, Abingdon (2007)
10. Cuny, J., et al.: Demystifying computational thinking for non-computer scientists. *Work in Progress* (2010)
11. Curzon, P., et al.: Introducing teachers to computational thinking using unplugged storytelling. In: *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, pp. 89–92 (2014)
12. Djurdjevic-Pahl, A., Pahl, C., Fronza, I., El Ioini, N.: A pathway into computational thinking in primary schools. In: Wu, T.-T., Gennari, R., Huang, Y.-M., Xie, H., Cao, Y. (eds.) *SETE 2016*. LNCS, vol. 10108, pp. 165–175. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52836-6_19
13. Grover, S., Pea, R.: Computational thinking in K–12 a review of the state of the field. *Educ. Res.* **42**(1), 38–43 (2013)
14. Guzdial, M.: *Computing for Other Disciplines*. Cambridge, UK (2019)
15. Arastoopour Irgens, G., et al.: Modeling and measuring high school students' computational thinking practices in science. *J. Sci. Educ. Technol.* **29**(1), 137–161 (2020). <https://doi.org/10.1007/s10956-020-09811-1>
16. Jawad, H.M., et al.: Integrating art and animation in teaching computer programming for high school students experimental study. In: 2018 IEEE International Conference on Electro/Information Technology (EIT), pp. 0311–0317. IEEE (2018)
17. Kafai, Y.B., et al.: A crafts-oriented approach to computing in high school: introducing computational concepts, practices, and perspectives with electronic textiles. *ACM Trans. Comput. Educ. TOCE.* **14**(1), 1–20 (2014)
18. Kordaki, M., Kakavas, P.: Digital storytelling as an effective framework for the development of computational thinking skills. In: *EDULEARN 2017*, pp. 3–5 (2017)
19. Lévano, M., et al.: Methodological framework for the development of computational thinking and programming through experiential learning: case study from schools Juan Seguel and Allipen in Freire, Chile. In: 9th IADIS International Conference, p. 107 (2016)
20. Looi, C.-K., et al.: Analysis of linkages between an unplugged activity and the development of computational thinking. *Comput. Sci. Educ.* **28**(3), 255–279 (2018)

21. Lytle, N., et al.: Use, modify, create: comparing computational thinking lesson progressions for STEM classes. In: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, pp. 395–401 (2019)
22. Nygren, E., et al.: Quantitizing affective data as project evaluation on the use of a mathematics mobile game and intelligent tutoring system. *Inform. Educ.* **18**(2), 375–402 (2019)
23. Parsazadeh, N., et al.: Integrating computational thinking concept into digital storytelling to improve learners' motivation and performance. *J. Educ. Comput. Res.* **59**(3), 470–495 (2021)
24. Peel, A., et al.: Learning natural selection through computational thinking: unplugged design of algorithmic explanations. *J. Res. Sci. Teach.* **56**(7), 983–1007 (2019)
25. Puentedura, R.R.: Learning, technology, and the SAMR model: goals, processes, and practice. <http://www.hippasus.com/rrpweblog/archives/2014/06/29/LearningTechnologySAMRModel.pdf>
26. Tedre, M., et al.: Changing aims of computing education: a historical survey. *Comput. Sci. Educ.* **28**(2), 158–186 (2018)
27. Thijs, A., Fisser, P., Van der Hoeven, M.: 21e-eeuwse vaardigheden in het curriculum van het funderend onderwijs [21st century skills in the curriculum of fundamental education] (2014)
28. Wing, J.M.: Computational thinking. *Commun. ACM.* **49**(3), 33–35 (2006)
29. Yeni, S., Grgurina, N., Hermans, F., Tolboom, J., Barendsen, E.: Exploring teachers' PCK for computational thinking in context. In: The 16th Workshop in Primary and Secondary Computing Education, 18–21 October 2021 (2021, accepted)



Towards Classification of Interactive Non-programming Tasks Promoting Computational Thinking

Tomas Šiaulys^(✉) and Valentina Dagiene

Institute of Data Science and Digital Technologies, Vilnius University, Vilnius, Lithuania
{tomas.siaulys, valentina.dagiene}@mif.vu.lt

Abstract. Tasks from Bebras International Challenge on Informatics and Computational thinking are becoming a widespread educational tool for introducing students to the concepts of computer science not only in a form of a contest, but as well in computer science classrooms, games and unplugged activities. There is an ongoing debate in Bebras community about the need and the role of interactivity in Bebras tasks. Practices in different countries range from pen-and-paper solutions to fully interactive systems with immediate feedback. This work introduces classification of Interactive Bebras tasks based on engagement and interactivity taxonomies aligned with current practices of Bebras task development. Proposed classification is then used to categorize the latest tasks from Lithuanian, French and Italian Bebras Challenges. The results from this work shed more light on the properties, potential benefits and shortcomings of non-programming interactive tasks informing future research on the topic as well as current practices of organizing International Bebras Challenge.

Keywords: Interactive tasks · Bebras · Engagement taxonomy · Computational thinking

1 Introduction

Computational thinking (CT) expands the view of computer science from something only programmers should know to the tool for understanding our technology-infused social world (Denning and Tedre 2019). This view encouraged creation of national K-12 curriculum frameworks for CT as well as plenty of tools for introducing students to computer science. While these numerous tools are implemented in new and shiny environments, vast majority of them are based on the same old ideas of turtle graphics and block-based programming. CT is mostly a domain dependent skill, however, it can be argued that learning a few basic statements in a block-based programming environment might not sufficiently cover the broad area of skills that CT refers to. There are only a few exceptions of tools for introducing students to CS that do not explicitly focus on introductory programming, e.g. CS unplugged (Bell and Vahrenhold 2018) and Bebras (Dagiene 2010).

Bebras International Challenge on Informatics and Computational Thinking is a contest, organized every year by dozens of countries across the world to motivate and introduce students to computer science (www.bebas.org). Due to the popularity of the contest, a large database of good quality tasks (reviewed and revised by international CS educators) and plentiful original ideas, Bebras influenced many different initiatives including games (Aslina et al. 2020), unplugged activities (Dagiene and Stupuriene 2017), curriculum content (Budinská and Mayerová 2019; Lonati 2020), textbook content, teacher training and more (Combéfis and Stupurienė 2020). Bebras tasks are also used as an assessment tool for computational thinking skills (Poulakis and Politis 2021).

Bebras tasks are often presented using visual metaphors for CS concepts. These visualizations often resemble algorithm and data structure visualizations used in higher education, however Bebras task pictures are more concrete, involving characters and objects from the stories of tasks. Bebras visualizations depend strongly on the ideas of the tasks, which sometimes imply interactivity in the task presentation. It can also be argued that in certain cases the dynamic nature of CS concepts might be better explained using interactive visualizations rather than static ones. In their criteria for good Bebras tasks Dagiene and Futschek (2008) promote visualizations and interactivity as well as immediate feedback. On the other hand, Vaníček (2014) argues that interactivity may affect how pupils solve the task and that it is important to consider in what ways tasks change when interactivity is added to avoid worsening the quality of the tasks.

Since interactivity can mean different things in different contexts, studies on interactivity from other fields may not be relevant for studying interactive non-programming tasks in K-12. In the context of educational games and simulations, interactivity and real-time feedback are usually studied more broadly without much focus on details (Vlachopoulos and Makri 2017). The reason for that might be the wide range of interactive activities that games and simulations might contain. In the fields of algorithm and program visualization, interactivity is mostly studied in terms of user engagement in learning with visualizations. Engagement taxonomies have been proposed by Naps et al. (2002) and Sorva et al. (2013), however, broad definitions of engagement levels might not reflect specific details about user engagement and it might be problematic to apply engagement taxonomies when studying specific software visualization types, such as visual programming (Šiaulys 2020) or interactive Bebras tasks. There have also been attempts to introduce general interactivity classifications and models, however, none of them seem to be sufficient to explain interactivity specific to interactive computer science concept-based Bebras tasks.

The goal of this work is to introduce a classification of interactive non-programming tasks on computational thinking and to validate this classification by categorizing recent Bebras tasks used in different countries. The following research questions are addressed:

RQ1: What are the most important properties of interactive non-programming tasks?

RQ2: What are the current practices in Bebras community in terms of interactive task design and what could be improved?

To answer the first research question we discuss engagement taxonomies as well as different classifications of interactivity and visualization relevant for this research (Sect. 2). Based on previous research and current practices, Sect. 3 presents a newly

proposed classification of interactive Bebras tasks. To answer the second research question, the proposed classification is used to categorize the latest tasks used in Lithuanian, French and Italian Bebras Challenges (Sect. 4). The results are discussed in Sect. 5.

2 Related Work

Interactivity is a broad term meaning different things in different contexts. Two broad areas of interactions in computer systems can be distinguished: 1) interactions between users, and 2) interactions between users and the system. In order to analyze the properties of the tasks facilitating solving and learning process, in this work we will focus on user-system interactions. The defining feature of user-system interactivity is system's responsiveness to the learner's actions (Moreno and Mayer 2007).

2.1 Interactivity and Engagement Classification Frameworks

There have been many attempts to categorize interactivity levels or types. One of the first and most cited attempts to do that is a taxonomy by Schwier and Misanchuk (1993) defining three levels of interactivity: reactive, proactive and mutual. Reactive interaction is a response to presented stimuli by the system, while proactive interaction emphasizes learner construction and generative behavior. In mutual interactivity level learner and system are capable of changing in reaction to encounters with each other. Schulmeister (2001) defined a more fine-grained interactivity levels: Level I allows only passive listening, reading or watching, Level II allows users to navigate through the information, Level III allows the learner to manipulate the form of representation, Level IV allows manipulating the content, Level V allows the learner to construct new content and Level VI allows the learners to create new content and receive intelligent system feedback (similar to Schwier & Misanchuk's mutual interactivity). Moreno and Mayer (2007) argue, that interactivity can be seen as continuum and instead distinguish between different interactivity types: dialoguing (learner receives questions and answers or feedback to his/her input), controlling (learner controls pace and/or order), manipulating (learner sets parameters for a simulation or manipulates objects), searching (learner enters queries, selects options) and navigation (learner moves to different content areas).

Similar classifications were proposed in the field of computing education research. Naps et al. (2002) defined six different forms of learner engagement with software visualization technology: no viewing, viewing (watching the visualization, controlling the execution), responding (answering questions concerning the visualization), changing (modifying the visualization, for example, by changing the input), constructing (learners construct their own visualizations) and presenting visualization to the audience for feedback and discussion. The taxonomy was later extended by Myller et al. (2009) introducing ten fine-grained engagement levels. Sorva et al. (2013) criticized previous attempts for the lack of separation of content ownership and engagement with the system and introduced a two-dimensional engagement taxonomy (2DET). 2DET differentiates between direct engagement dimension: no viewing, viewing, controlled viewing, responding, applying (making use of or manipulating visual components), presenting,

creating; and content ownership dimension: given content, own cases, modified content, own content.

It has been hypothesized that increasing the level of engagement would result in better learning outcomes and that the mix of different forms of engagement would be more beneficial than a single engagement form (Naps et al. 2002). A survey partially supporting engagement taxonomy was carried out by Urquiza-Fuentes and Velázquez-Iturbide (2009) regarding program visualization and algorithm animation systems. However, other studies did not find evidence for improved learning (Myller et al. 2007; Osztian et al. 2020). Osztian et al. argued that visualizations can be engaging even without any interactivity and that interactivity might have positive as well as negative effects.

It has been argued that higher levels of interactivity of learning platforms are in line with constructivist learning theories that promote active learner engagement in constructing his/her knowledge. In their meta-analysis Hundhausen et al. (2002) found that studies focusing on constructivist learning theories in algorithm visualization, were much more capable of explaining visualization effectiveness than other learning theories. Authors argue that what learners do and not what they see might influence learning the most. However, what is often missing from the discussion is that constructivism is a theory of how learning is being constructed, which does not necessarily imply that any active or hands-on activity automatically improves learning (Ben-Ari 2001). Moreno and Mayer (2007) note that although interactive environments promote behavioral activity, the focus of learning environments should be promoting appropriate cognitive processing.

Domagk et al. (2010) argued that a focus on interactivity quantification does not give us enough information about its role in the learning process and proposed a new Integrated model of multimedia interactivity. Authors state that the newly proposed model separates six specific components that together comprise interactivity: the learning environment, behavioral activities, cognitive and metacognitive activities, motivation and emotion, learner characteristics, and the learner's mental model (learning outcomes). The interactivity is hence seen as a process and represented by feedback loops that connect these components. While this definition allows a more nuanced view on interactivity, specific details presented in other classifications are left out making this model less suitable for practical applications.

A process-centered approach was also used by Bellettini et al. (2019) in the context of Bebras Challenge by constructing a multidimensional model for describing interactions with the system from a technical point. Authors described three levels of interactivity: Level 0 with no activity (another task is displayed), Level 1 - reading/watching/thinking, when the task is displayed but there is no action and Level 2 - acting, when the student is inserting or changing the answer of the task, or is performing an action that gets a feedback from the system. Authors then used the proposed classification together with other indicators, such as initial reading time, number of sessions, etc., to describe user interaction process with the contest system. While this type of classification is useful for analyzing user behavior, for a full picture of interactivity properties of tasks have to be taken into consideration as well.

3 Classification Framework for Interactive Bebras Tasks

There are different aspects of user-system interactivity that current classifications fail to address. Bebras-like tasks differ from the learning content analyzed in previous categorizations primarily in a way that users are expected to provide solutions to the problems, which implies at least minimal interactivity from the system. However, interactivity in these tasks has different purposes, hence our categorization distinguishes between guidance-oriented interactivity and solution-oriented interactivity dimensions. While these dimensions do partly overlap, we argue that distinguishing between them brings more clarity in analyzing interactivity.

Guidance and feedback are probably the most important factors when it comes to multimedia learning. Taxonomy by Schulmeister (2001) describes the highest level of interactivity (Level VI), where the learners create their own content and receive intelligent feedback. However, it is important to note that feedback (as a form of guidance) itself can be a subject to categorization (de Jong and Lazonder 2014) and thus might be better reflected in a separate dimension rather than a single category. In our classification *feedback/guidance* dimension (Table 1) is partly based on the guidance categories described by de Jong and Lazonder (2014), however the focus is moved towards interactive feedback. While it can be argued that any interaction by definition constitutes feedback from the system, the focus of feedback/guidance dimension is on describing additional interactive features as opposed to the ones that are needed for providing the solution.

Table 1. Feedback/guidance dimension of interactive tasks' classification framework

Type of feedback/guidance	Explanation
Process constraints	Restricting the number of options students need to consider
Numerical feedback	Additional feedback providing users with numerical information on separate objects or the whole system they are interacting with
Visual feedback	Feedback providing users with additional visual information regarding the objects they interact with as a form of guidance. Visual feedback can be static or animated
Descriptive feedback	Mostly text-based feedback as a form of guidance for a particular step or the whole solution. Textual information can be displayed after a certain interaction or before interaction, e.g. suggesting on how to proceed with further interactions

Guidance that is provided in Bebras-like tasks can be assigned to two broad categories: directive guidance that steers the learner in a certain direction and non-directive guidance, for example, restricting the space of possible actions as described by de Jong and Njoo (1992). *Process constraints* category from our classification reflects mostly non-directive guidance and categories of *numerical feedback*, *visual feedback* and *descriptive feedback* reflect more directive guidance. While process constraints do



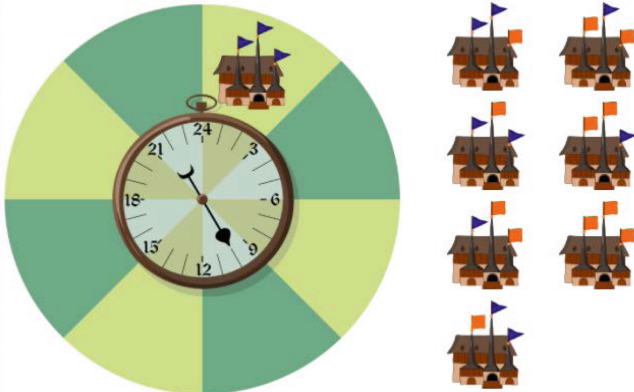
not directly increase interactivity, they direct user attention towards meaningful interactions instead of allowing free exploration, which could be detrimental to the learning process (Patwardhan and Murthy 2015).

Construction of the answer/solution is represented by *solution-oriented interactivity* dimension (Table 2) referring to different ways users can interact with the system to construct their solutions and provide answers. Previous taxonomies by Sorva et al. (2013) and Schulmeister (2001) only partly reflect this aspect – 2DET introduces *applying* category to describe direct manipulation of visualization’s components similarly to Level IV interactivity of Schulmeister’s taxonomy of multimedia components. We argue that in the context of interactive non-programming tasks more attention is required towards user interactions with the system in the process of constructing the solution for the task. Different interactivity types can also be found in a template of Bebras tasks used by international community. While interactivity classification used in this template does help describing how tasks should be implemented, we argue that less technical and more process-oriented classification would be more beneficial from a pedagogical point of view.

The lowest level of solution oriented interactivity, named *basic interactivity* encompasses all the standard test questions, like multiple choice (including options with images), drag-and-drop, input field, etc. separate from the main visualization setting the task. Such visualizations allow students to provide the answer for the task but do not support the solving process. *Integrated interactivity* on the other hand, even though constructed using the same basic interactive objects (e.g., drag-n-drop), allows relating them to the overall system set by the task (like the second example in Table 2). From the learner’s point of view this is interaction with the whole system and thus the solving process is partly supported by interactivity - the learner can solve the task systematically by interacting with objects in a sequential manner, changing previous choices, etc. Integrated interactivity is also supported by split-attention principle stating that learning is hindered, when learners are required to split their attention between several sources of physically or temporally disparate information (Plass et al. 2009). *Solving process interactivity* encompasses interactive objects that are not directly required for providing the solution. These additional interactive objects are intentionally designed to alleviate the solving process and reduce the need for external representation (e.g., pen and paper). However, it is worth noting that the need for external representation is not always easy to judge and representation provided by the task might not correspond with the one, which student would independently choose for the solving process. The last and usually the most complex type of interactivity *interactive simulation* allows additional logic connecting the objects, i.e. when the user interacts with one object, other objects or even the whole system can react to user actions. Interactive simulations usually support the solving process with experimentation and exploration of the system being interacted with. Some interactive simulations also allow situations, where it is self-evident, whether the solution is correct or not thus supporting the process of solving even more. Types of interaction described here do not necessarily describe support for the solving process in a sequential manner, however often that is expected to be the case.

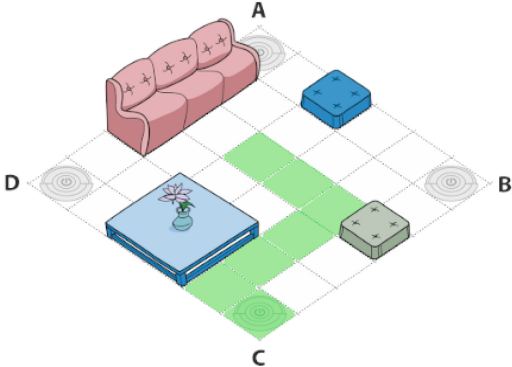
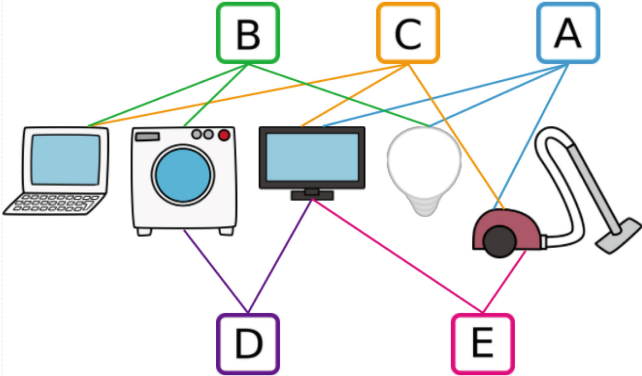
Level of freedom in constructing the content, on the other hand, might not constitute interactivity by itself. Schulmeister (2001) describes the second highest interactivity

Table 2. Solution-oriented interactivity dimension of interactive tasks’ classification framework

Type of interactivity	Task example
<p>Basic interactivity Traditional test response options, such as multiple choice, drag-and-drop, drop-down list, etc. separate from the main visualization.</p>	<p>Bebras task <i>2020-IE-08 Creatures Zoo</i> uses basic multiple-choice template. that does not depend on task contents: Some creatures eat only fish, others eat only plants:</p>  <p>The zookeeper has arranged the creatures as follows. Click on the enclosures where they forgot to put a plant-eating creature:</p> 
<p>Integrated interactivity Multiple choice, drag-and-drop, clickable objects with two or more states integrated into the main visualization.</p>	<p>Bebras task <i>2020-CH-18 Beaver Town Clock</i> differs from the above example in a way that users interact with the visualization that sets the task. Even though the interaction is still basic, some part of the solving process is already supported – users can choose different strategies to construct the solution and change their answer until it fulfils the requirements.</p> <p><...> Provide a perfect flag system, where only one flag changes from each interval to the next. There are many possible perfect systems. Provide any one.</p> 

(continued)

Table 2. (continued)

<p>Solving process interactivity</p> <p>Allows intermediate steps to be made in order to solve/ experiment without the need for external representation.</p>	<p>A variation on Bebras task 2017-LT-03 <i>Vacuum Cleaner Robot</i> allows interacting with the grid for exploring the optimization problem. This type of interaction allows systematic analysis of the path being optimized thus reducing the need to write things down.</p> <p>A robot washes square tiled floor by using the following commands:</p> <ul style="list-style-type: none"> • F – move forward (takes 1 minute) • R – turn 90° right (performed instantly) • W – wash a tile (takes 1 minute) <p>Robot starts and ends on one of the four corner tiles (A, B, C, D) but not necessarily on the same one. You can experiment by clicking the tiles.</p>  <p>How many minutes at least does the robot spend for cleaning the floor? <input type="text"/></p>
<p>Interactive simulation</p> <p>Interaction with one object affects other objects.</p>	<p>Bebras task 2020-JP-01B <i>Household appliances</i> allows users to explore the mechanics behind the task by experimenting with objects and studying how interactions influence other parts of the system.</p> <p><...> Turn on the TV and the bulb just by pressing the buttons! Just press one button at a time and control which devices are working.</p> 

Level V allowing learners to construct new content, however, as noted by Naps et al. (2002), this can be done even with simple art supplies (pen, paper, scissors, etc.). Sorva et al. (2013) separates direct engagement dimension from content ownership dimension in their 2D engagement taxonomy arguing that it is not clear where content ownership lies in the spectrum of engagement. Hence, it can be concluded that without feedback from the system, free construction of content does not imply interactivity. In our classification we do not include the level of freedom in constructing the content, however, interactions that might be associated with freedom in constructing the content are described in both feedback/guidance and solution-oriented interactivity dimensions.

4 Classification of Tasks from National Contests

The proposed classification was used to categorize tasks from the latest national Bebras contests in Lithuania (52 tasks), Italy (28 tasks) and France (12 tasks). Since the tasks from French 2020 Bebras Challenge were not public at the time of preparing this paper, tasks from 2019 have been chosen, which should not influence the interactivity statistics much, since each year only highly interactive tasks are used in French national contest.

As it can be seen in Fig. 1, task distribution according to solution-oriented interactivity in Italy and Lithuania seems similar with the majority supporting basic interactivity as opposed to tasks used in French competition, majority of which involve high levels of interactivity. Solving process interactivity, on the other hand, was not supported by any task across all three countries.

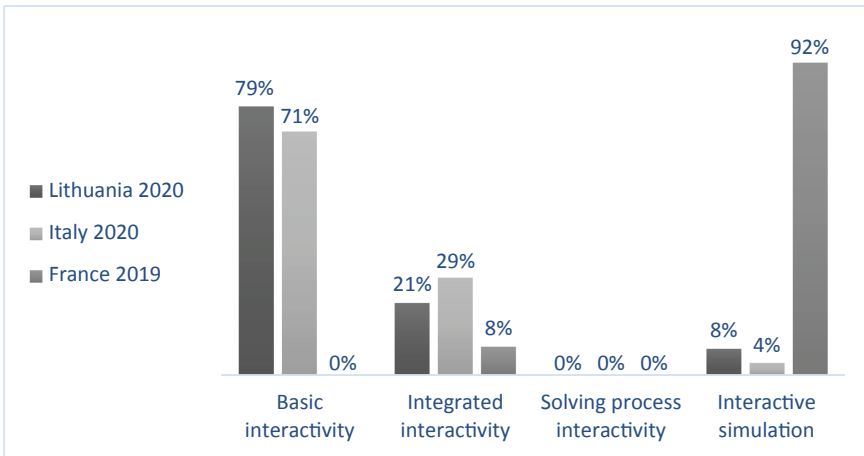


Fig. 1. Percentage of tasks attributed to solution-oriented interactivity categories

In the feedback/guidance dimension (Fig. 2) it appears that all tasks used in France support text-based feedback on user interactions with the system and in majority of the tasks text-based feedback is accompanied by additional visual feedback indicating problematic areas of solutions. Lithuanian and Italian systems rarely provided any additional

feedback to the students. While process constraints could be seen as an integral part of most tasks, statistics included only the tasks that implemented additional constraints in task mechanics. The need for additional constraints depends on the task, however, a few tasks from Lithuanian contest allowed interacting with the system without restrictions described in a task wording, which could arguably cause confusion.

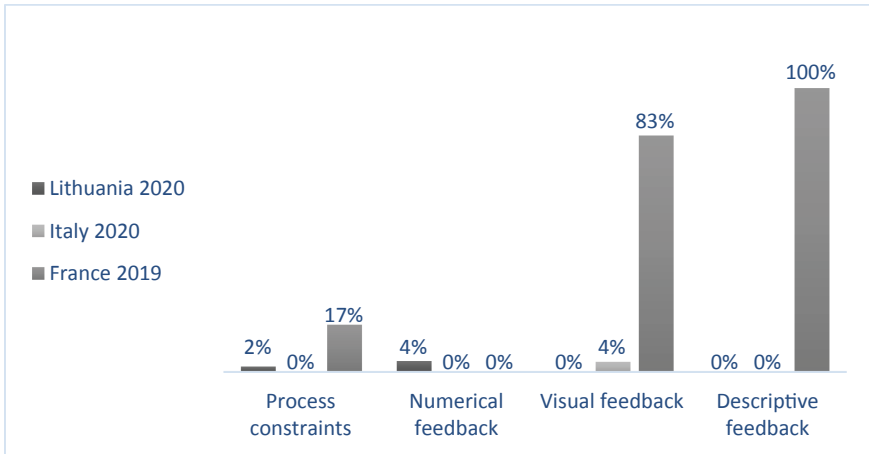


Fig. 2. Percentage of tasks attributed to feedback/guidance categories

5 Discussion

There are a few possible explanations for these results. Firstly, the type of interactivity to a great extent depends on the idea of the task. While different countries do adapt the tasks for national competitions, the main ideas of the tasks remain unchanged and if authors did not have interactivity in mind when creating the task, it might be difficult to change the task in a meaningful way. Even adding simple interactive components to the task might alter the dynamics of the task to such an extent that it becomes a completely new task that requires a different approach to solve. Secondly, task design considerations depend heavily on available resources: on types of interactivity that contest systems support, tools for creating interactive tasks as well as human resources for designing the tasks. While some tools, like Bebras Lodge (Dagienė et al. 2017) and Cuttle.org support designing a few types of higher interactivity tasks without programming, interactive simulations typically require more effort to design.

While interactivity is generally seen as a way to improve learning, it seems to receive rather little attention and there are certainly ways to improve current practices. As interactive features are usually an integral part of the task, task creators should be aware of more possibilities than multiple-choice questions when designing the tasks. For example, one of the explanations for the lack of solving process interactivity in tasks could be that it is not in the field of attention of task designers, even though it is usually not

as complicated to implement and could support learners in a meaningful way. The same could be true for integrated interactivity. Tools for designing interactive tasks should also receive more attention, as interactivity is relevant not only in computer science education, but in other subjects as well. Accepted standards for tools and libraries could also facilitate creation and exchange of such tasks.

It should not be expected for all the tasks promoting computational thinking to be highly interactive. However, computational thinking as a problem solving skill involves activities as well as concepts that are heavily process-oriented and interactivity can be an effective way to introduce them to students. Interactivity can also allow students to demonstrate higher cognitive skills according to Bloom's taxonomy, e.g. at the apply level for "carrying out or using a procedure in a given situation" (Krathwohl 2002). In a context of e-assessment Boyle and Hutchison (2009) discuss sophisticated tasks that are highly interactive and allow measuring different things that otherwise could not be measured using multiple-choice questions. It can be argued, that some learning goals cannot be reached by using interactive tasks, for example, it might be complicated to address students' ability to find a suitable external representation for a given situation. However, interactive tasks provide more possibilities and should be taken into account for the wide array of learning goals to be addressed.

6 Conclusion

In this work we introduced a classification framework for interactive non-programming tasks based on feedback/guidance and solution-oriented interactivity dimensions. The framework was used to categorize recent Bebras tasks from three different countries illustrating differences between approaches towards interactivity.

While there is no easy way to compare interactive and non-interactive tasks and it is clear that a universally optimal interactivity level cannot be established (Osztian et al. 2020), our research suggests that certain interactive qualities of tasks could enrich student experiences in CS and beyond. This is especially true for interactivity types supporting specific procedural activities that would otherwise be more difficult to reach for learners.

Future work could analyze if and how student's mental models and learning depend on different types of interactivity, possibly taking into account content types as well. Learning from interactive content is influenced by many factors and there remain many unanswered questions. This work is a small step towards better understanding of interactivity.

References

- Aslina, Y.R., Mulyanto, A., Niwanputri, G.S.: Designing "Bebras" serious games interaction for Indonesian upper elementary school students. In: 2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications (ICAICTA), pp. 1–6 (2020)
- Bell, T., Vahrenhold, J.: CS unplugged—how is it used, and does it work? In: Böckenhauer, H.-J., Komm, D., Unger, W. (eds.) *Adventures between lower bounds and higher altitudes*. LNCS, vol. 11011, pp. 497–521. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98355-4_29

- Belletтини, C., Lonati, V., Monga, M., Morpurgo, A.: How pupils solve online problems: an analytical view. In: CSEDU, no. 2, pp. 132–139 (2019)
- Ben-Ari, M.: Constructivism in computer science education. *J. Comput. Math. Sci. Teach.* **20**(1), 45–73 (2001)
- Boyle, A., Hutchison, D.: Sophisticated tasks in e-assessment: what are they and what are their benefits? *Assess. Eval. High. Educ.* **34**(3), 305–319 (2009)
- Budinská, L., Mayerová, K.: From bebras tasks to lesson plans – graph data structures. In: Pozdniakov, S., Dagienė, V. (eds.) *Informatics in Schools New Ideas in School Informatics. Lecture Notes in Computer Science*, vol. 11913, pp. 256–267. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33759-9_20
- Combéfis, S., Stupurienė, G.: Bebras based activities for computer science education: review and perspectives. In: Kori, K., Laanpere, M. (eds.) *ISSEP 2020. LNCS*, vol. 12518, pp. 15–29. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63212-0_2
- Dagienė, V.: Sustaining informatics education by contests. In: Hromkovič, J., Kráľovič, R., Vahrenhold, J. (eds.) *ISSEP 2010. LNCS*, vol. 5941, pp. 1–12. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11376-5_1
- 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
- Dagiene, V., Stupurienė, G.: Algorithms unplugged: a card game of the Bebras-like tasks for high schools students. In: *The 10th International Conference on Informatics in Schools (ISSEPS 2017)* (2017)
- Dagienė, V., Stupurienė, G., Vinikienė, L.: Implementation of dynamic tasks on informatics and computational thinking. *Baltic J. Mod. Comput.* **5**(3), 306 (2017)
- de Jong, T., Lazonder, A.W.: The guided discovery learning principle in multimedia learning. In: Mayer, R.E. (ed.) *The Cambridge Handbook of Multimedia Learning*, 2nd ed. pp. 371–390. Cambridge University Press (2014). <https://doi.org/10.1017/CBO9781139547369.019>
- de Jong, T., Njoo, M.: Learning and instruction with computer simulations: learning processes involved. In: De Corte, E., Linn, M.C., Mandl, H., Verschaffel, L. (eds.) *Computer-Based Learning Environments and Problem Solving*, pp. 411–427. Springer, Heidelberg (1992). https://doi.org/10.1007/978-3-642-77228-3_19
- Denning, P.J., Tedre, M.: *Computational thinking*. MIT Press, Cambridge (2019)
- Domagk, S., Schwartz, R., Plass, J.: Interactivity in multimedia learning: an integrated model. *Comput. Human Behav.* **26**(5), 1024–1033 (2010). <https://doi.org/10.1016/j.chb.2010.03.003>
- Hundhausen, C.D., Douglas, S.A., Stasko, J.T.: A meta-study of algorithm visualization effectiveness. *J. Vis. Lang. Comput.* **13**(3), 259–290 (2002). <https://doi.org/10.1006/jvlc.2002.0237>
- Krathwohl, D.R.: A revision of bloom’s taxonomy: an overview. In: *Theory into Practice*, vol. 41, no. 4, pp. 212–218. Ohio State University Press (2002). https://doi.org/10.1207/s15430421tip4104_2
- Lonati, V.: Getting inspired by bebras tasks. How Italian teachers elaborate on computing topics. *Inf. Educ. Int. J.* **19**(4), 669–699 (2020)
- Moreno, R., Mayer, R.: Interactive multimodal learning environments. *Educ. Psychol. Rev.* **19**(3), 309–326 (2007)
- Myller, N., Bednarik, R., Sutinen, E., Ben-Ari, M.: Extending the engagement taxonomy. *ACM Tran. Comput. Educ.* **9**(1), 1–27 (2009). <https://doi.org/10.1145/1513593.1513600>
- Myller, N., Laakso, M., Korhonen, A.: Analyzing engagement taxonomy in collaborative algorithm visualization. In: *ITICSE 2007: 12th Annual Conference on Innovation & Technology In Computer Science Education: Inclusive Education in Computer Science*, pp. 251–255 (2007)

- Naps, T.L., et al.: Exploring the role of visualization and engagement in computer science education. In: Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, pp. 131–152 (2002). <https://doi.org/10.1145/960568.782998>
- Osztian, P.R., Katai, Z., Osztian, E.: Algorithm visualization environments: degree of interactivity as an influence on student-learning. In: Proceedings - Frontiers in Education Conference, FIE, 2020-Octob (2020). <https://doi.org/10.1109/FIE44824.2020.9273892>
- Patwardhan, M., Murthy, S.: When does higher degree of interaction lead to higher learning in visualizations? Exploring the role of “Interactivity Enriching Features.” *Comput. Educ.* **82**, 292–305 (2015). <https://doi.org/10.1016/j.compedu.2014.11.018>
- Plass, J.L., Homer, B.D., Hayward, E.O.: Design factors for educationally effective animations and simulations. *J. Comput. High. Educ.* **21**(1), 31–61 (2009). <https://doi.org/10.1007/s12528-009-9011-x>
- Poulakis, E., Politis, P.: Computational thinking assessment: literature review. In: Research on E-Learning and ICT in Education: Technological, Pedagogical and Instructional Perspectives, pp. 111–128 (2021)
- Schulmeister, P.R.: Taxonomy of multimedia component a contribution to the current metadata debate interactivity. *Learning*, 1–17 (2001)
- Schwier, R., Misanchuk, E.R.: Interactive multimedia instruction. *Educational Technology* (1993)
- Šiaulys, T.: Engagement taxonomy for introductory programming tools: failing to tackle the problems of comprehension. In: Kori, K., Laanpere, M. (eds.) ISSEP 2020. LNCS, vol. 12518, pp. 94–106. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63212-0_8
- Sorva, J., Karavirta, V., Malmi, L.: A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Edu.* **13**(4), 1–64 (2013). <https://doi.org/10.1145/2490822>
- Urquiza-Fuentes, J., Velázquez-Iturbide, J.Á.: A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Trans. Comput. Educ.* **9**(2), 1–21 (2009). <https://doi.org/10.1145/1538234.1538236>
- Vaniček, J.: Bebras informatics contest: criteria for good tasks revised. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 17–28. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_3
- Vlachopoulos, D., Makri, A.: The effect of games and simulations on higher education: a systematic literature review. *Int. J. Educ. Technol. High. Educ.* **14**(1), 1–33 (2017). <https://doi.org/10.1186/s41239-017-0062-1>



Tell, Draw and Code – Teachers’ Intention to a Narrative Introduction of Computational Thinking

Karin Tengler¹ (✉) , Oliver Kastner-Hauler¹ , and Barbara Sabitzer² 

¹ University of Teacher Education Lower Austria, 2500 Baden, Austria
k.tengler@ph-noe.ac.at

² Johannes Kepler University Linz, 4040 Linz, Austria

Abstract. The increasing importance of introducing computer science and computational thinking in primary education highlights the need to prepare teachers adequately. This study reports on professional teacher training in programming. It draws on the Technology Usage Inventory (TUI) model to investigate how an interdisciplinary intervention with programmable robots, combined with the storytelling method, can influence the intention to use them in the classroom. Special focus is given to the Tell, Draw & Code method to offer teachers a didactic concept for implementation. The floor robots used in this study are Bee-bots. At the beginning and end of the teacher training course, the participating teachers completed a questionnaire about their perceptions, attitudes, and intentions about using robots in the classroom. In addition, after designing and implementing activities with robots, the teachers provided qualitative reflections on their experiences. In comparing pre- and post-test and the analyses of the qualitative data, the study shows a significant increase in the positive attitude towards using the robots. These findings highlight the need for teachers to have opportunities to explore, reflect on and experience the potential of new technologies as part of their teacher development to implement innovations sustainably.

Keywords: Computer science education · Educational robotics · Primary school · Professional teacher development

1 Introduction

Emphasis on problem-solving thinking and the introduction of computer science education for young learners have been increasing worldwide. To integrate computer science education in primary schools, in Austria, there is currently only the recommendation of the digikomp4 model [1], which serves as a reference framework and orientation guide for equipping primary school students with appropriate digital competencies up to the 4th grade. However, it is planned to anchor computer science education in the primary school curriculum in autumn 2023 [2]. Educational robots as a didactic tool offer one possibility of getting introduced to the first steps of computer science education and promoting computational thinking skills [3].

However, Austrian primary school teachers' knowledge of programming and programming didactics is limited. Although the topic has gained importance in recent years and research also attests to positive effects, there are still only a few singular initiatives in Austria dealing with programmable robots [4]. This situation has led to the case that the primary school teachers have limited support in didactic knowledge and good examples. But an essential aspect of implementing innovations in the education system is the willingness of teachers [5]. Furthermore, good training is necessary.

In the 2018/19 school year, all state kindergartens and primary schools in the province of Lower Austria - a total of 1,700 locations - were equipped with Bee-bot sets. The set consists of four Bee-bots, two Blue-bots, and two Tac Tile readers. Although at the beginning to offer training to teachers, two years later, there are still many schools where the Bee-bots are unfortunately stored unused in a box. One reason for this is undoubtedly the limited use during the Corona pandemic. Furthermore, skepticism and lacking knowledge about how to integrate robots in teaching could be why they are not used. To solve this situation, a project was developed that deals with programmable robots' interdisciplinary use combining the storytelling method in primary schools. This paper presents a teacher training course that consisted of a workshop followed by an asynchronous online phase. Since the training was attended by teachers who had little or no experience with programmable robots, the aim was to provide them with didactic possibilities and examples of an interdisciplinary approach to familiarize the students with simple concepts of computer science teaching. The study aims to answer the following questions:

RQ1: To what extent do interaction and teaching with programmable robots influence primary school teachers' intention to use robots in their classrooms?

RQ2: What are the teacher's perspectives of the intervention using robotics-based storytelling activities?

2 Background Work

2.1 Educational Robotics in Pedagogical Context

Digitization is postulated in all areas, especially in education. The responsibility of education is to teach students to become problem-solving, creative, and collaborative personalities [6]. Learning and innovation skills teach students the mental processes required to adapt to and improve a modern working environment. A focus on creativity, critical thinking, communication, collaboration, and computational thinking is essential to prepare students for the future [7]. Computational thinking [3] should already be promoted in primary school to develop one's solution strategies reflectively and solve the problem according to the scheme of an algorithm. Computational thinking (CT) can be understood as a problem-solving process [8] and should encourage pupils to apply this competence in everyday life. Programmable robots are increasingly part of everyday school life to acquire competencies in the field of digitization. Most of them are floor robots, which are being used and tested as learning media in more and more schools. Aspects of robotics and general media use can be introduced to the learners in this way and the technical and content-related competencies.

Educational robotics (ER) can be an effective way to introduce computational thinking. Students can systematically complete tasks and develop the sequenced step-by-step coding commands required to program a robot [9]. Benitti [10] investigated the educational potential of robotics in schools, identified how robotics could contribute by integrating it as an educational tool in schools, and examined its effectiveness. The use of programming tools provides young learners important computational approaches to addressing real-world problems [11]. Nouri et al. [12] investigated teachers' experiences with Bee-bots and Scratch. Their study identified the promotion of CT skills, digital literacy, and 21st-century skills, language skills, collaborative skills and attitudes, and creative problem-solving skills. The successful integration of robotics into the school classroom depends on the robot itself, the activities selected, and the material designed [13].

2.2 The Bee-Bot

The robot used in this study is the Bee-bot. This tangible robot allows students to initiate their coding skills [14]. They are especially suitable for young learners by promoting haptics and introducing them to computational thinking [4]. A Bee-bot is a very simple floor robot that represents a bee without wings. Seven buttons on the back program it. The programming is done with the key forward, backward, left turn, right turn, pause and delete. The program sequence is not started until the GO key is pressed. The steps to the destination must be well thought out and planned.

Nevertheless, its easy handling and the fact that it makes programming tangible for children makes the Bee-bot particularly suitable for encouraging problem-solving thinking [15]. Schina et al. [13] showed in their study that Bee-bots are not only helpful in promoting problem-solving thinking but can also be beneficial for children with attention deficits and dyslexia. Furthermore, the same authors reported that teachers mentioned that “the robot toys have various applications in foreign language teaching (in grammar and vocabulary)” [13, p.10].

2.3 Professional Teacher Development

The increasing demand for CT in K-12 education [16] and the introduction of computer science education in primary education have highlighted the need to prepare teachers with the technological, pedagogical and content knowledge (TPACK) necessary for teaching CT [17]. Many efforts have emerged to prepare teachers to teach coding [18]. Primary school teachers face a variety of obstacles when teaching CS. Teachers may meet many barriers, such as a lack of computers or reliable Internet access, and institutional obstacles in the form of unsupportive principals, lack of legislation, and emotional barriers, including beliefs, attitudes, and dispositions that hinder the use of technology [19]. To integrate these innovations into primary education, teachers must have the required knowledge, self-confidence, and a positive attitude [20] towards this concept. Regarding teachers' acceptance of robots in education, Chevalier et al. [21] highlight that intention to use robotics in the classroom increases when teachers are provided with appropriate teacher training, didactic approaches, and pedagogical materials that can be used directly. To teach coding, they must be well-prepared, and the training program must

build self-efficacy and address teacher's beliefs in importance and applicability [19]. Concerning teacher development and its relation to CT Kong [17] considers a lack of high-quality research. Angeli et al. [22] provided a conceptualization of TPACK for the construct of CT. However, this conceptualization focuses on the knowledge necessary to teach courses aligned to a specific didactic design independently. However, for any new educational technology to be successfully implemented and used in K-12 classrooms, teachers must be aware of new educational technology tools available, accept the technology as having practical benefits, feel confident that the technology is easy to use. They should have confidence in their ability to use the technology [5] and have opportunities for experimentation to minimize risks. In innovation research [23], it is assumed that the more the factors mentioned are fulfilled, the more reliably and quickly innovation is adopted and spread. In addition to the factors already mentioned, Kong et al. [16] recommend that effective teacher development take place over an extended period of time. The school context and opportunities for practice and reflection are essential, with the best development occurring when there is an opportunity for sharing. Also, Schina et al. [24], in their literature review on teacher training in the ER context, recommend that teachers should be given the opportunity to put their technical and pedagogical knowledge of robotics and programming into practice in the classroom.

3 Conceptual Framework

3.1 Technology Usage Inventory

In assessing teachers' attitudes towards using programmable robots as an introduction to computer science education, the Technology Usage Inventory (TUI) questionnaire developed by Kothgassner et al. [25] was used. This survey instrument is a further development of the technology acceptance questionnaire by Davis [26] and Venkatesh and Davis [27]. The further development mainly concerns psychological factors which are not sufficiently considered in the instruments used so far. The original procedure contains the following eight scales: Curiosity, Anxiety, Interest, Ease of Use, Immersion, Usefulness, Skepticism, and Accessibility [25]. Their internal consistencies range from $\alpha = .70$ to $\alpha = .92$. An adapted form of the questionnaire was used, as the scale "Immersion" was not relevant for this study. Instead, the scale "Necessity" ($\alpha = .92$) was added.

3.2 Teacher Training Course

At the beginning of the training, the participants were given theoretical knowledge about the programmable robot and informatics concepts that can be implemented using the Bee-bot. After that, they were familiarized with the programming commands and the functioning of the yellow floor robot. One difficulty was to provide practical experience with the Bee-bot, as all classes at the university were only held online due to the Corona pandemic. However, since all schools in Lower Austria were equipped with Bee-bots, some participants have a Bee-bot at home and use it to do the exercises.

The trainer transmitted the Bee-bot's functioning with the document camera so that all could at least see the Bee-bot. The participants were shown videos that were still

filmed in the classroom using the programmable robot. Moreover, the participants had further practice opportunities using the Bee-bot simulator¹ and the Bee-bot App. This setting made it possible to simulate the robot's functioning very well, and there was no disadvantage due to the online situation.

The second part of the training consisted of offering the participants possibilities for the interdisciplinary use of the Bee-bot. Above all, they were introduced to the didactic design of the Tell, Draw & Code method.

3.3 Didactic Design – Tell, Draw and Code

The didactic design chosen was an approach called Tell, Draw & Code by the authors. This method aims to implement computational thinking in connection with creative narrative and writing processes. In introducing simple programming languages, literary texts become a vehicle for coding and decoding language. First, the students' task is to read a text or invent a story, and then they reproduce it graphically. They have to create a map with the story or put the pictures on the map, e.g., they place parts of a picture story so that the Bee-bot will move along the story if it has been programmed correctly. The students can use as many commands as necessary. The Bee-bot can execute a maximum of 40 commands in a row. The text is structured through graphic representation, and through practical action, the storyline is constructed. By retelling the story or tale, the text is decoded again. This transformation requires problem-solving strategies by breaking down the problem into smaller steps or applying known patterns, as described in several computational thinking frameworks [8, 11]. The dialogic negotiation of computer science problems combined with creative representations of the stories in visual form should sustainably promote the children's narrative language. At the same time, the structuring and coding of the text should also give children with reading and spelling difficulties the opportunity to deal with the individual sections of the story in greater detail through programming. Initial successes have already been recorded with the Ozobot [28]. Tell, Draw & Code can be applied to the following settings:

- Retelling stories or fairy tales
- Creating stories
- Taking information out of a text and presenting it
- Getting to know children's literature.

4 The Study

4.1 The Participants

The quasi-experimental pre-post-test study occurred within a teacher university course.

The participants took part in a course consisting of six modules and lasting two semesters. This university course aims to provide teachers with knowledge about digital basic education and concepts for effective implementation in primary education. One

¹ <https://beebot.terrapinlogo.com>.

module called “Developing problem-solving strategies” teaches introduction to computer science education, particularly the development of computational thinking, i.e., the promotion of problem-solving competence, and how it can be implemented, for example, with programmable robots. This training consisted of a live workshop held online due to the Corona pandemic and an asynchronous online phase over seven weeks, which took place via a learning platform. The participants are introduced to the Bee-bot, the Blue-bot, and Ozobot robots and shown how to use programmable robots in the classroom. As part of this module, their task was to plan, implement and reflect on an interdisciplinary teaching unit. Afterward, they had to report their experiences and findings in writing.

The participants ($n = 16$) of the study were primary school teachers; 14 women and two men. Most participants were between 30 and 40 years old. They teach classes from grade 1 to grade 4.

4.2 The Survey

Pre-and post-test were conducted by using an online questionnaire on the LimeSurvey platform. The pre-test questionnaire consists of 31 items, 2 relate to demographic data and 29 to the TUI model. The post-test consists of 31 items, 2 demographic questions, 24 related to the TUI model, and 5 items related to acquired competencies. The four-part Likert scale (4 = strongly agree, 3 = agree, 2 = disagree, 1 = strongly disagree) was chosen as the response format. In addition, the post-test questionnaire contains an open question on the experiences made using the programmable robots. 14 of the 16 participants completed both questionnaires, 12 female and 2 male participants.

The quantitative data were analyzed with descriptive statistics using SPSS 26. The scales INT, SKE, ETU, USE, NEC, ITU were used to examine a change in participants’ attitudes before and after the intervention. The t-test was applied to compare significant mean values. This test is particularly suitable for comparing two dependent samples [29]. The participants’ reflections provided the data for the qualitative results after the implementation of the intervention. The statements were coded and analyzed using a qualitative content analysis [30]. They offered insights into the teacher’s perspectives of the intervention and the students’ competencies promoted through these settings.

5 Results

The presentation of the results is organized as follows. First, two examples of the lessons designed by the participating teachers are presented. Then, the quantitative survey results are presented, which investigated how interaction and teaching with programmable robots influence primary school teachers’ intention to use robots in their classrooms. Finally, the question about the experiences and perspectives is answered.

5.1 Lessons

This chapter provides two examples of the lessons where the participants implemented the Tell, Draw & Code method. For lesson planning, most participants opted for the

Bee-bot. Two teachers used the Blue-bot. The teachers reasoned their choice that these robots were available at their school.

Creating a Story: The task of the Year 2 pupils was to write a picture story. In this case, they were given six pictures, which they had to put in the appropriate order to make a meaningful story. Then the story was programmed with the Bee-bot. At the same time, the students told the plot of the story to the other children. Afterward, the story was written down in the exercise book.

Retelling a Book: The book “The Gruffalo” was read to the students (Fig. 1). When an animal was named, a child programmed the Bee-bot to move on to the respective picture card of the animal. In the end, the story was retold.



Fig. 1. The Gruffalo

5.2 Evaluation Pre-/Post-test

To answer the research question (RQ1), to what extent do interaction and teaching with programmable robots influence primary school teachers’ intention to use robots in their classrooms, a pre-post-test was conducted.

The pre-test was carried out right at the beginning of the workshop. First, the participants were asked which of the three robots presented they were familiar with and whether they had already tried them out. All participants who correctly completed the questionnaire ($n = 14$) knew the Bee-bot, but only six of them had already tried it out. Only six teachers knew about the Blue-bot, and none had had any experience with it. The post-test was conducted after the course and the intervention. Analyzing the descriptive data of both tests it was surprising that the intention to use (ITU) already showed a very high value in the pre-test ($M = 3.50/SD = 0.58$) and that this value increased even more in the post-test ($M = 3.73/SD = 0.27$). Ease to Use (ETU), pre-test ($M = 3.0/SD = 0.72$) and post-test ($M = 3.60/SD = 0.47$) and Necessity (NEC), pre-test ($M = 3.26/SD = 0.63$) post-test ($M = 3.61/SD = 0.57$), also showed high ratings.

The post hoc paired samples t-test (Table 1) was used to compare pre- and post-test changes in teachers’ attitudes after learning about the Tell, Draw & Code method and after the intervention.

Table 1. Post hoc paired sample t-test

Variable	Mean difference	SD	t	df	Sig.	Cohen's d_z
ACC	0.191	0.855	0.834	13	0.419	0.222
INT	0.304	1.020	1.114	13	0.286	0.298
NEC	0.357	1.008	1.325	13	0.208	0.350
SKE	-0.179	0.717	-0.093	13	0.927	0.025
ETU	0.536	0.720	0.192	13	0.015	0.745
USE	0.643	0.891	2.699	13	0.018	0.721
ITU	0.238	0.646	1.379	13	0.191	0.368

Table 1 shows a significant difference between teachers' combined robot-related attitudes and intentions before and after the intervention. Particularly significant differences are found in the variables *Ease to Use* (ETU) ($s = 0,015/d = 0,745$) and *Usefulness* (USE) ($s = 0,018/d = 7,21$). Hence, this test shows that the participants' increase in these variables can be explained reliably by the intervention carried out.

5.3 Qualitative Findings

The second research question (RQ2), what are the teacher's perspectives of the intervention using robotics-based storytelling activities and what students' competencies were identified, was determined qualitatively. 15 of the 16 participants in the course planned a lesson implementing programmable robots, conducted it, and reflected on it in writing.

Table 2 shows the teacher's perspectives of the intervention using robotics-based storytelling activities. Although they were skeptical about the usefulness and necessity of educational robotics in teaching at primary school initially, all participants ($n = 15$) were enthusiastic after the course. There was only positive feedback. Although all the participants have known the Bee-bots, their experience lacked so far. They also reported not having had any appropriate concepts and didactic approaches until now. "Until today, I have not known how to use the robots in a meaningful way" (T_11).

Most of the participants were aware of the Bee-bots because they are available at the school. But the teachers lacked experience so far. They also reported not having had any appropriate concepts and didactic approaches until now. "Until today, I have not known how to use the robots in a meaningful way" (T_11). The participants reported that they were delighted that they finally had to try out the Bee-bots. They also stated that actively interacting with Bee-bots during teacher training's task plays a crucial role in their positive attitude towards them. Ten participants (66.7%) reported that they now have relevant knowledge. "I was able to take away fascinating ideas and didactic approaches from this course, and I am happy that I was able to try out the Bee-bots" (T_5). "I probably would not have used the Bee-bots without this course. Now I am convinced that they can be used successfully in the classroom" (T_10). 73.3% of the participating teachers were particularly enthusiastic about the playful and simple programming and the children's immediate sense of achievement. One teacher reported: "I am delighted

that I was finally able to get to know the Bee-bot better. This made me realize that programming the bee is more difficult for adults than for the children” (T_3). “The children programmed without any problems, and each unit was a great success” (T_5). “The effort is shallow, and the learning success is enormous” (T_8).

Table 2. Teachers’ perspectives of the intervention using robotics-based storytelling activities

Codes	Example of statements	F	%
Causing joyful learning	The children were very enthusiastic and programmed great stories (T_15)	11	73.3
Providing playful and simple programming	The children soon understood how to program the robot (T_5)	10	66.7
Supporting the learning process	The students have never learnt so many words before (T_6)	4	26.7
Fostering problem-solving thinking	I was observing how the students playfully developed problem-solving strategies (T_10)	8	53.3
Promoting teamwork	I was very fascinated how the children worked respectfully in the group and supported each other (T_12)	13	86.6
Promoting reading and narrative skills	By programming the story, special attention was paid to detailed retelling of the story of the Gruffalo (T_8)	7	46.7
Fostering spatial thinking	Spatial orientation was greatly enhanced by programming the paths of the Bee-bot (T_6)	5	33.3
Providing an appropriate didactic approach	I was able to take away fascinating ideas and didactic approaches from this course (T_5)	10	66.7

Seven participants (46.7%) also confirmed that the promoting of reading and narrative skills plays a big part. Since the students spoke a lot during programming and when creating the stories. “The children also spoke aloud during programming” (T_6). The method of storytelling was also very well received by the participants because “the children are even more motivated to tell stories through the Bee-bots” (T_5). “The children made up different stories and were happy when the bee successfully followed the right path. In doing so, they retold the story” (T_15). They described in the reflections the method as very purposeful and motivating. “Narrative skills were fostered playfully” (T_8).

According to the teachers, this interdisciplinary use of robots can promote problem-solving thinking very well. 13 participants (86.6%) mentioned this aspect by using educational robotics combining storytelling activities. “I was able to observe the strategy development of individual pupils, and it also became clear who still had problems with foresight and logical thinking. But by helping each other, everyone was able to develop further” (T_8). Mainly (86.6%), teamwork was highlighted as very positive: “They

helped each other appreciatively in the group so that everyone had a sense of achievement with the Bee-bots in the end.” (T_14).

Four teachers (26.7%) stated that Bee-bots supported the learning process very much in the subjects. In English classes, many children naturally learned and used more vocabulary than before” (T_14).

Four teachers passed on their knowledge to their colleagues in a conference. One teacher reported that “other teachers have also acquired a taste for it” (T_12). The teachers also agreed to continue using the robots. “I am totally enthusiastic about how much fun and motivation the bee has provided and will definitely use this teaching tool more often” (T_4). “I can only recommend working with these programmable robots” (T_10).

5.4 Discussion

The results of the study show that the use of learning robots in primary school is very motivating. Furthermore, storytelling in combination with programmable robots is a promising didactic method to introduce computer science education in primary school in an interdisciplinary approach, extend the language in creative activities, and use it to develop problem-solving strategies. Specifically, this setting supporting computational thinking skills offers a replicable learning design for teachers. The study also demonstrated that it is essential to actively try out new methods, such as losing one’s fear of programming and being better able to judge whether one is convinced of it. Trying out the robots, getting to know didactic applications, and the practical implementation [16] of the method during the teacher training leads to an increase in attitude regarding the intention to use [25]. This study also highlights the importance of equipping teachers with pedagogical, technical and content knowledge [17], as well as self-confidence and a positive attitude [20], to implement computer education and achieve an approach to computational thinking.

The innovative use of programmable robots notably promoted the willingness to tell stories. Before storytelling, the practice phases showed a high potential for intensive independent engagement with the programming language. For the students and teachers involved in the intervention, informatic skills emerged that could be taken up and developed in other narrative and writing projects. The innovative approach of linking narrative language, visual language and programming language through Bee-bots can thus be sustainably introduced into everyday school life. It shows that computer science lessons can be realized as interdisciplinary competence development through the new curriculum.

6 Conclusion and Future Work

This study contributes to understanding teachers’ attitudes towards using robots in primary education and the usefulness of robots in the classroom. This understanding can contribute to gaining knowledge about implementing computer science education and more positive attitudes towards robotics integration. The method Tell, Draw & Code is dedicated to integrating programmable robots for developing traditional narrative and

writing skills. Engaging in creating one's own stories or examining literary texts found in children's books and controlling the robots in a challenging and motivating way can lead to synergies in the learning process. The stories come alive in a particular way by changing the written language to visual language and spoken language.

This study provided empirical evidence that the teacher training program effectively changed participants' skills and attitudes. The evaluation was based on objective tests and self-assessments, overcoming problems related to initial skepticism. By designing and implementing the Tell, Draw & Code method, teachers can experience a pedagogical model that they can implement in their primary school teaching. Based on the available findings, we can validate that ER teacher training courses are effective, if the teachers are offered appropriate didactic concepts [21], if they last longer and the participants then also have the opportunity to exchange ideas [24], if they can gain practical experience in lesson planning and in implementing the robot activities when teaching [13]. By bringing in own ideas, the familiarity with the method is increased even more.

Future work is planned to study teachers' attitudes towards implementing and using the Tell, Draw & Code method specifically with Ozobots only and comparing the results with the research presented here.

Appendix

(See Fig. 2 and Table 3).

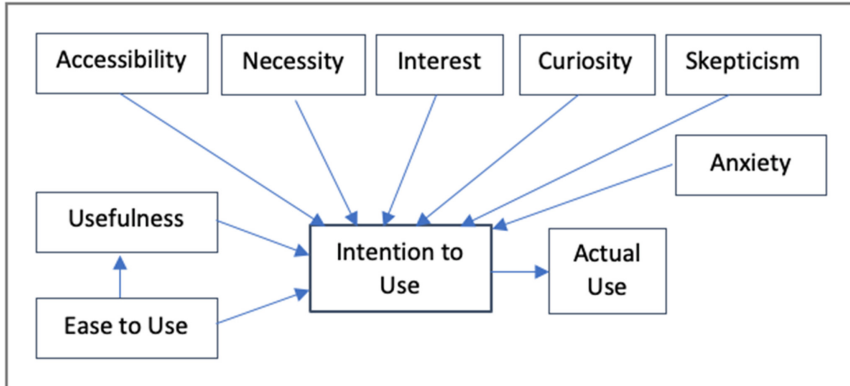


Fig. 2. TUI Acceptance model by Kothgassner et al. [25] (adapted version)

Table 3. Scales and internal consistencies

Scale	Variable	Items	Cronbach's Alpha
Anxiety	ANX	4	0,83
Curiosity	CUR	4	0,87
Interest	INT	4	0,86
Skepticism	SKE	3	0,70
Ease to Use	ETU	3	0,74
Usefulness	USE	2	0,70
Accessibility	ACC	3	0,86
Necessity	NEC	3	0,92
Intention to Use	ITU	3	0,84

Table 4. Descriptive data of pre-and post-test

Variable	Time	Items	M	SD	MIN	MAX	VAR
INT	Pre	4	2,66	0,81	2,36	3,36	0,22
	Post	4	2,96	0,74	1,50	4,00	0,55
SKE	Pre	3	1,74	0,66	1,36	2,00	0,03
	Post	3	2,05	0,52	1,00	3,00	0,27
ETU	Pre	3	3,04	0,72	3,00	3,07	0,00
	Post	3	3,60	0,47	2,67	4,00	0,23
USE	Pre	2	3,29	0,58	3,21	3,36	0,01
	Post	2	3,29	0,37	3,00	4,00	0,14
ACC	Pre	3	2,57	0,77	2,43	2,79	0,04
	Post	3	2,76	0,55	2,00	3,67	0,30
NEC	Pre	3	3,26	0,63	3,22	3,23	0,01
	Post	3	3,61	0,57	2,33	4,00	0,32
ITU	Pre	3	3,50	0,58	3,29	3,64	0,04
	Post	3	3,73	0,27	3,33	4,00	0,04

References

1. BMBWF: Digitale Kompetenzen-Informatische Bildung. digikomp4 (2021)
2. Kern, A.: Weiterentwicklung der Lehrpläne der Primar- und Sekundarstufe in Österreich. Medienimpulse 58 (2020). <https://doi.org/10.21243/mi-01-20-8>
3. Wing, J.M.: Computational thinking. *Commun. ACM.* **49**, 33–35 (2006)
4. Himpsl-Gutermann, K., et al.: Denken lernen–Probleme lösen (DLPL) Primarstufe (2017)

5. Rogers, E.M., Singhal, A., Quinlan, M.M.: Diffusion of innovations 1. In: *An Integrated Approach to Communication Theory and Research*, pp. 415–434. Routledge (2019)
6. Battelle for Kids: Framework for 21st century learning definitions. Battelle for Kids (2019)
7. Fadel, C., Bialik, M., Trilling, B.: *Four-dimensional education* (2015)
8. ISTE & CSTA: *Operational Definition of Computational Thinking* (2011)
9. Chalmers, C.: Robotics and computational thinking in primary school. *Int. J. Child-Comput. Interact.* **17**, 93–100 (2018)
10. Benitti, F.B.V.: Exploring the educational potential of robotics in schools: a systematic review. *Comput. Educ.* **58**, 978–988 (2012)
11. Grover, S., Pea, R.: Computational thinking in K–12: a review of the state of the field. *Educ. Res.* **42**, 38–43 (2013)
12. Nouri, J., Zhang, L., Mannila, L., Norén, E.: Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Educ. Inq.* **11**, 1–17 (2020)
13. Schina, D., Esteve-Gonzalez, V., Usart, M.: Teachers’ perceptions of bee-bot robotic toy and their ability to integrate it in their teaching. In: Lepuschitz, W., Merdan, M., Koppensteiner, G., Balogh, R., Obdržálek, D. (eds.) *RiE 2020. AISC*, vol. 1316, pp. 121–132. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67411-3_12
14. Cervera, N., Diago, P.D., Orcos, L., Yáñez, D.F.: The acquisition of computational thinking through mentoring: an exploratory study. *Educ. Sci.* **10**, 202 (2020)
15. Tengler, K., Sabitzer, B., Rottenhofer, M.: “Fairy tale computer science”—creative approaches for early computer science in primary schools. In: *ICERI Proceedings*, pp. 8968–8974 (2019)
16. Kong, S.C., Abelson, H., Lai, M.: Introduction to computational thinking education. In: Kong, S.C., Abelson, H. (eds.) *Computational Thinking Education*, pp. 1–10. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-6528-7_1
17. Koehler, M.J., Mishra, P., Cain, W.: What is technological pedagogical content knowledge (TPACK)? *J. Educ.* **193**, 13–19 (2013)
18. Bers, M.U., Flannery, L., Kazakoff, E.R., Sullivan, A.: Computational thinking and tinkering: exploration of an early childhood robotics curriculum. *Comput. Educ.* **72**, 145–157 (2014)
19. Mason, S.L., Rich, P.J.: Preparing elementary school teachers to teach computing, coding, and computational thinking. *Contemp. Issues Technol. Teach. Educ.* **19**, 790–824 (2019)
20. Mishra, P., Koehler, M.J., Henriksen, D.: The seven trans-disciplinary habits of mind: extending the TPACK framework towards 21st century learning. *Educ. Technol.* 22–28 (2011)
21. Chevalier, M., Riedo, F., Mondada, F.: How do teachers perceive educational robots in formal education? A study based on the Thymio robot. *IEEE Robot. Autom. Mag.* **23**, 16–23 (2016)
22. Angeli, C., et al.: A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *J. Educ. Technol. Soc.* **19**, 47–57 (2016)
23. Casey, J.E., Pennington, L.K., Mireles, S.V.: Technology acceptance model: assessing pre-service teachers’ acceptance of floor-robots as a useful pedagogical tool. *Technol. Knowl. Learn.* **26**(3), 499–514 (2020). <https://doi.org/10.1007/s10758-020-09452-8>
24. Schina, D., Esteve-González, V., Usart, M.: An overview of teacher training programs in educational robotics: characteristics, best practices and recommendations. *Educ. Inf. Technol.* **26**(3), 2831–2852 (2020). <https://doi.org/10.1007/s10639-020-10377-z>
25. Kothgassner, O.D., Felnhofer, A., Hauk, N., Kastenhofer, E., Gomm, J., Kryspin-Exner, I.: *Technology Usage Inventory (TUI)*. Man, Wien (2013)
26. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.* 319–340 (1989)
27. Venkatesh, V., Davis, F.D.: A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Manag. Sci.* **46**, 186–204 (2000)

28. Tengler, K., Kastner-Hauler, O., Sabitzer, B.: Enhancing computational thinking skills using robots and digital storytelling. *CSEDU* (1), 157–164 (2021)
29. Döring, N., Bortz, J.: *Forschungsmethoden und Evaluation*. Springer-Verlag, Wiesb (2016)
30. Mayring, P.: *Qualitative Inhaltsanalyse. Grundlagen und Techniken*. Beltz, Weinheim **4**, 58 (2015)

Programming Education



First Programming Course in Business Studies: Content, Approach, and Achievement

Djordje M. Kadijevic^(✉)

Institute for Educational Research, Belgrade, Serbia
djkkadijevic@ipi.ac.rs

Abstract. This paper presents the content of, a didactic approach to, and achievement of a first programming course for students engaged in business studies. Visual and textual programming were combined and a didactic approach that mainly focused on programming and fulfilling sub-tasks was used. This educational approach was applied to novice students of average ability and resulted in modest programming achievements. These achievements are first summarized, presenting the solution of a web application whose development was accessible to most students. Then, possible reasons for such achievements are discussed. The paper ends with several suggestions for practice and research, including a proposal for another sequencing of programming topics that would increase students' motivation to learn programming and improve their learning outcomes. Requirements for hybrid environments supporting both visual and textual programming are also considered. Although the applied methodology and resulting achievements reflect the authors experience in teaching programming to second-year undergraduate students, they are relevant to teaching programming in high school education as well.

Keywords: Course achievement · Course content · Didactic approach · First programming course · Teacher professional development · Textual programming · Undergraduate students · Visual programming

1 Introduction

A first programming course is a demanding enterprise for both teachers and learners. It seems that, in general, novice programmers can successfully deal with programs with line and branch structures, but this is not the case for programs with a loop structure that are usually hard for most of them (e.g., [1, 2]). Programming achievements may be low even for students of above average abilities (e.g., high school students oriented towards studying mathematics or science), who may even struggle with introductory tasks involving loop structures, such as “Describe the output for the given program code”, especially “Modify the

given code to attain the specified behavior of the new code” [3]. As a result, various misconceptions have been identified in the literature for basic programming concepts, including variables, conditional statements, and loops (e.g., [4]).

Understanding and using loop structures is a challenging and error prone task because it involves a number of basic concepts, such as variables, data types, variables initialization, assignments, and compare operators, which, when different loop structures are used (e.g., while instead of for), must be packed in different ways [1]. The reader may recall here the SOLO model of learner’s task understanding [5], which progresses from a single aspect (uni-structural response) via several, disjointed, aspects (multi-structural response) to several, integrated aspects (relational response). Undoubtedly, it is much easier to examine program statements (instructions) separately than to examine and use them in an integrated way within a coherent whole such as a loop structure (e.g., [6]).

What should the teacher expect when a first programming course makes use of object-oriented programming (OOP) language, when linear, branch and loop structures have to make use of many built-in (and perhaps some user-defined) classes?

Several recent studies underlined and exemplified the complexity of teaching such a course (e.g., [7]). Nevertheless, it seems widely accepted that visual programming (e.g., in Scratch; <https://scratch.mit.edu/>¹, which is more suitable for beginners (especially at lower educational levels), should be practiced before textual programming [8]. However, learning benefits of visual versus textual programming may be negligible (e.g., [9,10]). Furthermore, in some cases we cannot apply this instructional line: start with visual programming and then switch to text programming. In that case, bearing in mind the importance of web applications (e.g., in business matters), we may base the first object-oriented programming course on developing these applications by combining visual and textual programming. This approach is in accord with a number of recent papers that call for using a hybrid programming environment or a hybrid programming language that may positively influence students’ interest and retention in computer science (e.g., [11]). Moreover, such a course would, as Hubwieser requires [7], promote the learning of issues that are useful in professional life.

Bearing in mind undergraduate students engaged in business studies, what would an appropriate educational context for a first programming course be?

Firstly, computer science is a discipline much broader than programming and this context should not promote a wrong picture about the nature of this discipline [12]. Having in mind e-commerce, databases, and various programs on the Internet, the context should include the creation of simple databases and their use within web applications. Secondly, the teaching of computer science requires an appropriate balance between using computer tools required by the modern information society and programming computer applications that solve certain problems [13]. This opens a space for visual programming as these tools usually have a drag-and-drop interface that is used for various business tasks

¹ Although not an OOP language, it supports OOP programming: see https://en.scratch-wiki.info/wiki/Object-Oriented_Programming.

(e.g., summarizing data using dashboards; see <https://www.idashboards.com/dashboard-examples/>) without typing the commands that lie behind the applied drag-and-drop actions.

The next, second section presents the design of, didactic approach to, and achievement of the developed programming course for second-year undergraduate students engaged in business studies at a private faculty. Several suggestions for practice and research, based upon the experience presented in this section and other relevant research studies, are summarized in the final, closing section.

2 Content, Approach, and Achievement

This section comprises three subsections. The first presents the content of the developed course by referring to the learning material used in it, the second describes the didactic approach applied in covering this content, whereas the third summarizes students' programming achievements in qualitative and quantitative terms.

2.1 Course Content

Inspired by Sharp's book [14], the teacher/author of this paper first taught a one-semester course covering about 40% of this book. The students were required to create simple console and web applications as well as simple classes. The teacher repeated this course two times and realized that students' programming achievements would be better had they only created web applications, where the use of visual ASP.NET language could motivate some of them to learn a bit of traditional, textual programming (which initially started in the context of console applications, typically resulting in programmers frustrations). With his teaching assistant, the teacher then wrote a booklet [15] of which 40 pages are devoted to nine exercises. These exercises deal with:

1. The basics of work in the programming environment chosen (Microsoft Visual Web Developer 2010 Express);
2. Developing first web application;
3. Developing web application with linear structure;
4. Developing web application with data validation;
5. Developing web application with branch structure;
6. Developing web application with loop structure;
7. Developing web application with user-defined class;
8. Creating simple database at Microsoft SQL server and developing web application for displaying its content and updating it;
9. Developing web application for processing data from database.

Each exercise comprises a number of questions and tasks, a reminder of conceptual and procedural issues related to them, and one task for independent work. While exercises 4 and 8 make use of visual programming only, all other

exercises require students to apply both visual and textual programming. Sharp's [14] book focuses on object-oriented features of the C# language (this edition was for Visual C# 2010), whereas the booklet [15] focuses on combining visual and text programming, paying attention to the quality of data. Combining the two kinds of programming is only done for simple, yet interesting and workplace-relevant web applications, whereas concerns about the quality of data (a highly critical business issue) are reinforced in several exercises by using various data validation controls, particularly within data controls that manage the work with databases. Figure 1 presents the web pages of the simplest and the most complex web applications as well as typical data quality checks covered by the booklet.

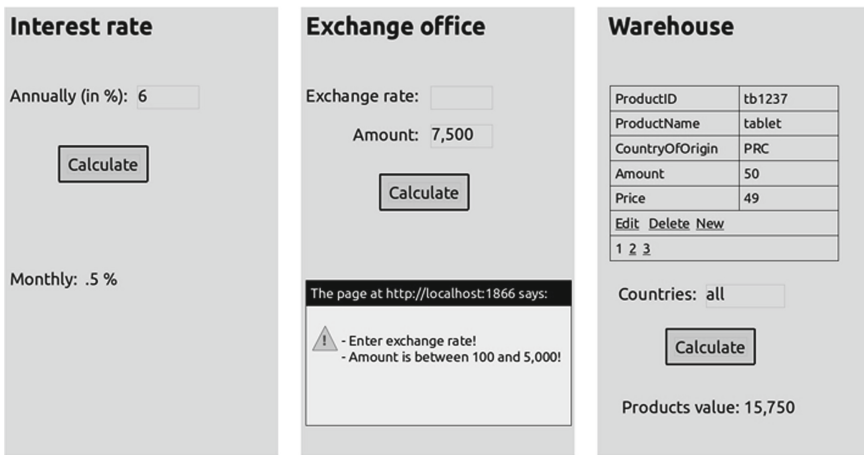


Fig. 1. Screenshots of three web applications: from annual to monthly interest (left), validating data for money exchange (middle), finding the value of all products in warehouse (right)

2.2 Didactic Approach

As mentioned above, learning components of programming tasks should be useful in students professional life [7]. Regarding students involved in a business studies program, such learning components may be:

1. Using drag-and-drop user interface to create objects relevant to task to solve;
2. Managing data quality;
3. Recognizing what data to store and use from database;
4. Recognizing input and output data;
5. Recognizing sub-tasks in task to examine or solve;
6. Recognizing type of structure (linear, branch or loop) in sub-task to examine or solve;
7. Managing the implementation of data and program structures.

Having in mind these components, the teacher and his teaching assistant based their lectures and exercises on the following steps:

1. Recognize input and output data;
2. Select attributes, variables, and web controls;
3. Divide task into sub-tasks;
4. Select type of programming (visual or textual) for each sub-task;
5. Select structure and statements for each programming sub-task;
6. Undertake visual and textual programming;
7. Modify existing web application to solve task or sub-task.

Some of these steps are exemplified in videos mirroring the work in the class. These videos were developed by the teaching assistant to support students' learning outside the class.

The seven steps presented above may be viewed as an elaboration of the didactic approach described by Armoni et al. [16]. Such an elaborated approach may serve as a framework to arrange the practice of programming and analyze its achievement, which was informally applied in this contribution. This elaboration was needed as our course required students to combine visual and textual programming, use various web controls, and work with databases. Step 7 was also added as we applied it in the booklet [15] in several places. We thus advised students to create web applications by modifying existing applications, whenever appropriate.

2.3 Programming Achievement

The programming course described in this contribution was delivered at a small private faculty of business studies in five consecutive academic years. In each year, there were, on average, between 60 and 70 second-year students who had registered for this course (usually with about 10%–15% more females than males); almost all of them were 20/21 years old. Most students had not had any prior experience in programming.

The application of the didactic approach presented in the previous subsection resulted in a modest achievement. On the basis of observations of classroom work and analyses of several exam results – especially in the last four years when the booklet [15] and accompanying videos were available – it was found that most students could only create web applications with line and branch structures. In doing so, they usually skillfully used various standard and validation controls. An example of web application whose development was accessible to 60%–70% of students in these four years is given in Appendix A.

Only about 20% of students could create a one-table database and a web application for displaying its content and updating it, but many of them did not use validation controls within a data control that enables work with a database. Less than 10% of the students could create web applications with a user-defined class (e.g., Exchange office, see Fig. 1) or web applications for processing data from databases (e.g., calculating and displaying the total sum of all donations or just of donations from a given industry).

The modest programming achievement reported above may be found unsatisfactory. However, it is close to the learning outcomes described by Mark Guzdial's 20% rule: "In every introduction to programming course, 20% of the students just get it effortlessly . . . 20% of the class never seems to get it" (source <https://blogs.kcl.ac.uk/proged/2009/09/04/quality-oriented-teaching-of-programming/>).

Why was the achievement so modest?

There were different windows in the programming environment used, different attributes for each web controls, different program structures in each application, as well as different instructions within each structure, and, as already explained by applying the SOLO model [5], difficulties were not generated by parts but primarily by the wholes combining them. In addition, difficulties did not generate from the understanding of instructions, but from recognizing sub-tasks or program structures where they could be applied, which called for a very demanding learning activity of problem structuring and articulation [17]. These sources of difficulties seemed to apply for the seven didactics steps as well, especially for step 7. In order to solve proposed tasks in the classroom and at exams, students could use or modify the code of some existing applications developed in the course (by themselves, the teacher, or the teaching assistant), but this was rarely applied successfully. It must be underlined that several surveys (administered during the course realization and at the final exam in different academic years) evidenced that the overall programming experience of many students was unsatisfactory (less than 10–15 h of individual work) mostly because of the lack of a home computer and/or interest in learning programming.

Were visual and textual programming related?

Analyses of several exam results evidenced the following: skillful visual programming was often coupled with successful textual programming, whereas a failure in the former was frequently coupled with a failure in the latter. This means that despite these kinds of programming not being related to the same programming language (visual programming was in the ASP.NET language, whereas textual programming was in the C# language), they positively influenced each other. This outcome was expected because standard controls used by a web application hold, as their values, data processed by a program implementing this application. In other words, to have these controls live, a program was needed, and vice versa. In addition, the work with validation controls beyond the drag-and-drop actions was similar (albeit small in scope) to the work required by textual programming in general, such as variable declaration, variable assignment, and condition specification. It can thus be said that visual and textual programming were linked by entities and their properties used in both kinds of programming (e.g., input/output entities, entity data type, entity value assignment, entity value constraint(s)).

3 Suggestions for Practice and Research

According to Bennedsen and Caspersen [18], short recordings of good programming practice would have positive learning effects. Because of that, as already

mentioned, the written learning material [15] was coupled with videos mirroring the work in the classroom. These videos were developed in the tradition of minimalism [19] – an approach to document and instructional design. This approach calls for solving tasks (personally meaningful and motivating for students) in direct, solver-tailored ways with the smallest amount of instructional verbiage. As Hubwiese evidences [20], the approach, although not explicitly labeled, is applied in some German programming textbooks. The minimalist approach is particularly relevant for combining visual and textual programming where, due to the sea of concepts and their applications, the programmer can easily lose the intended track. Having in mind Bennedsen and Caspersen [18], its application should include some narrated programming sessions that begin with solutions developed by students (not done in the programming course examined in this contribution).

After realizing that creating only web applications would be more motivating and accessible to students than creating simple console and web applications as well as simple classes (an experience gained in the first year of the course realization in this faculty as well as through a longer teaching of this course in another private faculty of business studies; see Sect. 2.1), the teacher decided to base his teaching on the work with web applications and applied it in four consecutive academic years. To improve students' programming confidence and learning outcome, the teacher and his teaching assistant prepared the aforementioned course booklet and accompanying videos. The programming practice in these four years made use of these resources and resulted in the modest programming achievement reported above, which was similar from year to year. Note that before their work with web applications with branch and loop structures in the last, fifth year, students created simple databases and developed web application that displayed and updated its content, paying attention to data validation. As a result, more students than earlier were interested in this programming practice based upon visual programming, which despite somewhat limited success in applying data validation, make them familiar with important aspects of their future managerial work (using databases, applying data validation). Consequently, more students than earlier were interested in developing web applications using branch and loop structures, although their achievement was similar to that observed in the previous years. To improve their performance, additional support was clearly needed to help them cope with the use of loop structures.

The experience summarized in the previous paragraph may support a proposal that to motivate students to learn programming and improve their learning outcomes, another sequencing of programming topics than that presented in Sect. 2.1 (operationalized in [15]) should be applied. According to this new sequencing reflecting lessons learned, the first exam should deal with developing a web application with a linear structure using data validation controls (combining visual and textual programming). The second exam should assess students' ability to create a simple database and develop a web application that displays and updates its content, paying attention to data validation (only visual programming is required to increase students' motivation for programming). The

third, final exam should focus on web applications with linear, branch, and loop structures applying data validation controls (combining visual and textual programming). Developing a web application with a user-defined class or for processing data from a database should be left for an optional project at the end of the course. A sample of tasks suitable for these exams is given in Appendix B.

The observation reported above that visual and textual programming positively influenced each other, supports the proposal to combine visual and textual programming within a hybrid programming environment. This environment can be implemented in various ways. It may, for example, enable the user to drag-and-drop visual blocks into a text editor with a textual code resulting in an update of that code [21], or use visual blocks that mirror the syntax of a professional programming language such as Java [22].

The author's experience in teaching programming suggests developing an environment that makes use of two connecting views of the solution (application or its component) being developed:

- Design view, where the programmer develops this solution only visually by using a number of visual programming blocks (e.g., implemented in the ASP.NET programming mentioned above, or in the PUCK environment [23]); and
- Code view, where the programmer can not only examine the automatically generated code (that mirrors the solution in the design view; e.g., [23]) but also update that code, having each code change reflected in the visual solution created in the design view (e.g., implemented in the ASP.NET programming; cf. Google Blockly at <https://developers.google.com/blockly>).

Furthermore, the programmer may run solutions in each of these two views (windows), and the results of these runs may be visualized in a third window in a manner applied by a program visualization tool such as Jeliot [24]. Further research may focus on developing such hybrid environment. When available, its use will enable researchers to study ways in which visual and textual programming influence each other. This is an important research question, answers to which would help educators to improve teaching programming.

A final didactic suggestion concerns a learning path that may be followed in developing students' programming knowledge and skills. For Brennan and Resnick [25], an understand–debug–extend learning path is suitable, especially for novice programmers. For Lee et al. [26], a use–modify–create learning path is appropriate. By combining these paths, the following path may be recommended: use web applications (to understand or evaluate them) – modify web applications (to debug or extend them) – create web applications, i.e. develop web applications from scratch.

Acknowledgement. This contribution resulted from the author's research financed by the Ministry of Education, Science and Technological Development of the Republic of Serbia (Contract No. 451-03-9/2021-14/200018). The author dedicates the contribution to his son Aleksandar.

Appendix A – Solution of Exchange office task

- A. Code produced by textual programming, which has to be entered by the programmer, is given in Fig. 2 between {}.

```
protected void Button1_Click(object sender, EventArgs e)
{
    decimal rate = Convert.ToDecimal(TextBox1.Text);
    int amount = Convert.ToInt16(TextBox2.Text);
    decimal value = rate * amount;
    Label14.Text = "Exchange value: " + Convert.ToString(value);
}
```

Fig. 2. Code produced by textual programming

- B. Code generated by visual programming is presented in Fig. 3. Ten controls needed are underlined, whereas the values that have to be assigned by the programmer are shaded.

```
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" Height="9px" Style="z-index: 100; left: 112px;
    position: absolute; top: 59px" Text="Exchange office" Width="120px"></asp:Label>
<asp:Label ID="Label2" runat="server" Style="z-index: 101; left: 50px; position: absolute;
    top: 120px" Text="Exchange rate:"></asp:Label>
<asp:Label ID="Label3" runat="server" Style="z-index: 102; left: 45px; position: absolute;
    top: 173px" Text="Amount:"></asp:Label>
<asp:TextBox ID="TextBox1" runat="server" Style="z-index: 103; left: 109px; position: absolute;
    top: 117px"></asp:TextBox>
<asp:TextBox ID="TextBox2" runat="server" Style="z-index: 104; left: 128px; position: absolute;
    top: 172px"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Style="z-index: 105; left: 108px; position: absolute;
    top: 227px" Text="Calculate" OnClick="Button1_Click" />
<asp:Label ID="Label14" runat="server" Style="z-index: 106; left: 52px; position: absolute;
    top: 286px" Text=""></asp:Label>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
    ControlToValidate="TextBox1" Display="None" ErrorMessage="Enter exchange rate!" Style="z-
    index: 107; left: 356px; position: absolute; top: 42px"></asp:RequiredFieldValidator>
<asp:RangeValidator ID="RangeValidator1" runat="server" ControlToValidate="TextBox2"
    Display="None" ErrorMessage="Amount is between 100 and 5,000!" MaximumValue="5000"
    MinimumValue="100" Style="z-index: 108; left: 370px; position: absolute; top: 97px"
    Type="Integer"></asp:RangeValidator>
<asp:ValidationSummary ID="ValidationSummary1" runat="server" Style="z-index: 109;
    left: 375px; position: absolute; top: 179px" ShowMessageBox="True" ShowSummary="False"
    Style="z-index: 110; left: 383px; position: absolute; top: 168px"></asp:ValidationSummary>
</div>
</form>
```

Fig. 3. Code generated by visual programming

Appendix B – A sample of tasks given at three exams

Exam 1

To participate at a symposium, one needs to pay transportation costs of 500 EUR, accommodation costs of 100 EUR per day, and symposium fees of 300 EUR. The total costs of a 3-day participation at that symposium are thus $500 \text{ EUR} + 3 * 100 \text{ EUR} + 300 \text{ EUR} = 1,100 \text{ EUR}$. Write a web application that calculates and displays the total participation costs for given transportation costs, accommodation costs per day, number of days of stay, and symposium fees. Use three different controls for data validation.

Exam 2

Create a database comprising one table with data about donations, whose attributes are: donation code, donator's name, donator's country of residence (domestic, foreign), and donation amount (in EUR).

- A. Write a web application that displays and updates the content of this database.
- B. Extend the application developed under A so that for the country of residence given, an extended application displays only data of donators having that residence country.

Use three different controls for data validation to ensure that to be entered in the database, the donation code must consist of two letters followed by two digits (e.g., RU07).

Exam 3

Business efficiency is found by dividing expenses by revenue. If expenses equal 100,000 EUR and revenue is equal to 200,000 EUR, this efficiency is $100,000 \text{ EUR} / 200,000 \text{ EUR} = 0.5$, which means that for each EUR earned by a firm, it needs to spend 0.5 EUR.

- A. Develop a web application that calculates and displays the efficiency for given expenses and revenue. Use three different controls for data validation.
- B. If expenses are greater than revenue, the application has to return a message "Expenses greater than revenue!"
- C. For expenses and revenue given, calculate and display efficiency for the given data, as well as for revenues that are 10% and 20% higher than that given. To round the value of the efficiency to two decimal places, use the method `MathRound (name.of.variable, 2)`.

References

1. Hofuku, Y., Cho, S., Nishida, T., Kanemune, S.: Why is programming difficult? Proposal for learning programming in "small steps" and a prototype tool for detecting "gap". In: Diethelm, I., Arndt, J., Dnnebier, M., Syrbe, J. (eds.) Informatics in schools: Local proceedings of the 6th international conference ISSEP 2013 - Selected papers, pp. 13–24. Universitätsverlag Potsdam, Potsdam (2013)

2. Lahtinen, E., Ala-Mutka, K., Jrvinen, H.-M.: A study of the difficulties of novice programmers. In: Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005), pp. 14–18. ACM, New York (2005)
3. Milić, J.: Predictors of success in solving programming tasks. *Teach. Math.* **12**(1), 25–31 (2009)
4. Swidan, A., Hermans, F., Smit, M.: Programming misconceptions for school students. In: ICER 2018: 2018 International Computing Education Research Conference, 13–15 August 2018, Espoo, Finland. ACM, New York (2018)
5. Biggs, J.B., Collins, K.F.: Evaluating the Quality of Learning: The SOLO Taxonomy. Academic Press, New York (1982)
6. Shread, J., Carbone, A., Lister, R., Simon, B., Thompson, E., Whalley, J.L.: Going SOLO to assess novice programmers. In: Proceedings of ITiCSE08, 30 June–2 July 2008, Madrid, Spain, pp. 209–213. ACM, New York (2008)
7. Hubwieser, P.: A smooth way towards object oriented programming in secondary schools. In: Benzie, D., Iding, M. (eds.) Informatics, Mathematics and ICT: A Golden Triangle. IFIP WG 3.1 & 3.5 Working Conference CD Proceedings. IFIP & College of Computer and Information Science, Northeastern University Boston, Boston (2008)
8. Sysło, M.M., Kwiatkowska, A.B.: Introducing a new computer science curriculum for all school levels in Poland. In: Brodник, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 141–154. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_13
9. Xu, Z., Ritzhaupt, A.D., Tian, F., Umapathy, K.: Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. *Comput. Sci. Educ.* **29**(2–3), 177–204 (2019)
10. Weintrop, D., Wilensky, U.: Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Comput. Educ.* **142**, 103646 (2019)
11. Noone, M., Mooney, A.: Visual and textual programming languages: a systematic review of the literature. *J. Comput. Educ.* **5**(2), 149–174 (2018)
12. Tucker, A.B.: K-12 computer science: aspirations, realities, and challenges. In: Hromkovič, J., Královič, R., Vahrenhold, J. (eds.) ISSEP 2010. LNCS, vol. 5941, pp. 22–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11376-5_3
13. Szlávi, P., Zsakó, L.: Programming versus application. In: Mittermeir, R.T. (ed.) ISSEP 2006. LNCS, vol. 4226, pp. 48–58. Springer, Heidelberg (2006). https://doi.org/10.1007/11915355_5
14. Sharp, J.: Microsoft Visual C# 2008 Step by Step. Microsoft Press, Redmond (2008)
15. Kadijevich, D., Stević, M.: Basic of programming - a practicum [Osнови programiranja - Praktikum]. Megatrend University, Belgrade (2010)
16. Armoni, M., Benaya, T., Ginat, D., Zur, E.: Didactics of introduction to computer science in high school. In: Hromkovič, J., Královič, R., Vahrenhold, J. (eds.) ISSEP 2010. LNCS, vol. 5941, pp. 36–48. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11376-5_5
17. Jonassen, D.H.: Toward a design theory of problem solving. *Educ. Tech. Res. Dev.* **48**, 63–85 (2000). <https://doi.org/10.1007/BF02300500>

18. Bennedsen, J., Caspersen, M.E.: Exposing the programming process. In: Bennedsen, J., Caspersen, M.E., Kölling, M. (eds.) *Reflections on the Teaching of Programming*. LNCS, vol. 4821, pp. 6–16. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77934-6_2
19. Carroll, J.M. (ed.): *Minimalism Beyond the Nurnberg Funnel*. The MIT Press, Cambridge (1998). <https://doi.org/10.7551/mitpress/4616.001.0001>
20. 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
21. Weintrop, D., Wilensky, U.: How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *Int. J. Child-Comput. Interact.* **17**, 83–92 (2018). <https://doi.org/10.1016/j.ijcci.2018.04.005>
22. Noone, M., Mooney, A., Nolan, K.: Hybrid Java: the creation of a hybrid programming environment. *Irish J. Technol. Enhanc. Learn.* **5**(1), 3 (2021). <https://doi.org/10.22554/ijtel.v5i1.67>
23. Kohl, L.: Puck - a visual programming system for schools. In: Lister, R., Simon (eds.), *Koli Calling 2007 (Proceedings of the Seventh Baltic Sea Conference on Computing Education Research)*. *Conferences in Research and Practice in Information Technology*, no. 88, pp. 191–194. Australian Computer Society, Darlinghurst (2007)
24. Ben-Ari, M.: Visualisation of programming. In: Kadijevich, D.M., Angeli, C., Schulte, C. (eds.) *Improving Computer Science Education*, pp. 52–65. Routledge, New York (2013)
25. Brennan, K., Resnick, M.: New frameworks for studying and assessing the development of computational thinking. In: *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*. AERA, Vancouver (2012). <http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf>
26. Lee, I., et al.: Computational thinking for youth in practice. *ACM Inroads* **2**(1), 33–37 (2011). <https://doi.org/10.1145/1929887.1929902>



Why Young Programmers Should Make Game Art: A Study from a Game-Making Course

Mária Karpielová and Karolína Miková^(✉)

Comenius University, 842 48, Bratislava, Slovakia
{maria.karpielova, karolina.mikova}@fmph.uniba.sk

Abstract. Creating computer games is a complex task, and when the game is treated not as a program but rather as a project, other aspects may enter the game-making process, such as game design, game art or project management itself. The task's complexity also allows pupils to develop new skills and practices, which can help them motivate themselves, solve problems, or accomplish tasks. And although it is programming that is currently at the centre of many educational research endeavours in primary schools, other aspects of the game-making process could offer valuable insights, such as how pupils approach those other aspects or how this new complexity alters pupils' learning. In this paper, pupils' practices regarding the aspect of game art are explored. Using a mixed-methods design, we describe the experience of teachers at a primary school game-making course. The aim was to explore the role of game art in a programmer's creating process. Building on interviews with experienced teachers, functions that game art tasks fulfil are identified and then measured again to determine how meaningful they are according to teachers. Moreover, the assessment of the method of using pixel art tasks is described and it outlines reasons why it is a suitable option when introducing pupils to creating game art. Together, this paper offers insights into how game art can help regulate learning and promote good practices among primary school pupils.

Keywords: Constructionism · Programming in primary school · Game art · Pixel art · Practices

1 Introduction

1.1 Game Programming and Game Art

Creating computer games is a popular gateway into programming in primary schools. There is evidence of it being effective in introducing pupils to programming concepts [1–3], that it improves design thinking skills [4] and other skills such as switching from designer's perspective to player's, explaining issues, testing hypotheses [5, 6] or iteratively developing and revising solutions [7]. However, there is a lack of research focused on game art within creating games in primary school informatics; though a recent study provides more evidence that this aspect of a game creates space to express pupils'

interest and encourage more creativity within projects [8], which are both important aspects of constructionist learning [9].

Although game art may not serve to integrate programming concepts into pupils' work [8], it does add pupils' personality to games, moreover, it contrasts the difference between a game as a program and a game as a multi-faceted project. It may help introduce pupils to working with digital graphics, which is also specified in Innovated National Slovak Curriculum [10], and in a more authentic context, where there is natural need for this knowledge. Furthermore, the pause or the change in focus that it provides may be beneficial for pupils when solving problems, due to the phenomena of fixation and incubation [11], which explain why focusing on a different task allows for the subconsciousness to find solution to a problem. The final argument is based on the personal experience teaching at the game-making course GameCraft – most pupils simply enjoy creating game art for their games.

1.2 GameCraft

GameCraft¹ is a computer game development course for 8–15 years old pupils. It is not focused only on programming, but also on game design, game art, worldbuilding or storytelling. The teacher's role is to assist pupils as they learn and explain or instruct if it is needed. Pupils learn programming from various instructional tutorials, and this way they learn how to navigate the interface of the game-making software (Scratch, Construct 2 or Unity 3D) and new game mechanics. The goal is for pupils to become autonomous and regulate their learning. By completing tasks of their choosing, they build their own repertoire, and this allows them to eventually become independent and create games on their own.

Depending on what kind of tasks pupils prefer, many of them decide to focus on one aspect of the game, and it is usually either programming or game art. Despite this natural division, many times programmers choose to supply their own game art for their projects. This study explores this issue and offers some of the reasons why.

1.3 Previous Research

The research prior to the study presented in this paper was aimed at better understanding of the learning process of pupils focusing on programming tasks at this game-making course. Interviews with teachers helped describe the course's learning model and how different motivation drives pupils to become independent creators [12]. Moreover, the five-year-long history of the course allowed to also observe behaviours of pupils who had been attending the course for longer periods of time, sometimes years, and this provided information on how they differ from beginners and which skills they developed over time [13]. It also probed the question how programming and game art tasks complement each other, and in this paper, this relationship is further explored.

¹ More information about the course is available at <https://hemisfera.sk/gamecraft-online>.

2 Research Aims

The main objective of the research was to describe primary school programmers' tendencies regarding game art in the process of creating a game. During the interviews in the first phase of the research, teachers mentioned that programmers usually create their own game art using pixel art and identified numerous reasons why pupils enjoy it often despite not being very good at it. It was through a follow-up questionnaire that this issue could be explored further, gaining a deeper understanding of the importance of game art tasks' roles in the learning process.

Although the research describes practices at a game-making course, we believe that it could also offer interesting insights for other educators and practitioners into how to encourage healthy practices among primary school pupils and foster self-regulation in student-centred learning by incorporating game art tasks into programming projects.

3 Methodology

3.1 Sample of Participants

For the purpose of this research, 10 teachers were chosen to participate. All of the teachers had extensive experience with pupils at GameCraft; experience based on the length of their practice and the number of groups they taught.

There were seven men and three women in the study, out of which two were younger than 20 years old, five of them were between 20–26 and three were older than 26 years old.

As for their teaching experience, all of them taught at the course for at least two school years. There was a teacher, who started teaching in 2017/2018, then three teachers who began in 2018/2019 and six who started in 2019/2020. All of the teachers started teaching before the pandemic, and except one teacher, who now works on an educational project for older students, they continued to teach although the courses switched to online form.

Another important factor was the groups those teachers had experience with. Groups are usually quite diverse in age but even more diverse in the skill level of individual pupils. When asked, eight teachers answered that they taught complete beginners, all of them encountered intermediate pupils, who were creating their first games usually with the need of assistance, and nine of them taught proficient programmers who were creating their games independently, with little help. Regarding the programming software, all teachers supervised pupils' projects programmed in Scratch and Construct 2 and four of them also in Unity 3D.

We are aware that the sample size in this research could be considered too small to fully achieve the objectives of this research, especially those determined by the quantitative phase of the research, but we decided to prioritise experience of teachers over the number of participants. All participants in this research taught multiple groups of pupils, and both before and during the pandemic, which we considered invaluable for this study.

3.2 Data Collection and Analysis

The research was carried out in two phases. In the first phase, teachers were interviewed, which helped describe the learning process and various practices that young programmers attending the course had, among which were also tendencies regarding the place of game art in their learning process. Due to the worsening pandemic situation at the beginning of the year, the individual interviews with teachers were in the form of online video calls. All of these interviews were recorded and later transcribed. Then, the transcripts were manually coded using open coding [14] within a software for qualitative analysis of data, Quirkos.

In the second phase, an online questionnaire was created based on the data from the previous data collection. There were three objectives of the questionnaire; to describe how and when pupils create game art, to determine the significance of game art's various roles that teachers mentioned in the interviews and to assess pros and cons of using pixel art which seems to be very popular among programmers. The questionnaire consisted of mainly close-ended items and few open-ended items. Close-ended items aimed at the first two objectives and the open-ended items at the third.

Part of the close-ended items in the questionnaire contained items aimed at the functions of game art tasks. Based on the experience, teachers were asked to describe how meaningful were the individual functions in their groups of programmers, which would help determine the functions which were most common among programmers and which were not. They scored each item on a Likert scale from 1 (strongly disagree) to 5 (strongly agree). Quantitative data from the questionnaire were analysed using descriptive statistics and qualitative data were coded using open coding, then were categorised, and described.

4 Results

4.1 Game Art in the Game-Making Process

To better understand the context, it was important to gather data about programmers' tendencies regarding making game art, e.g. where the game art comes from or at which point of the process programmers create it.

The answer to the question of the game art's source can be seen in Fig. 1. When asked to check all the boxes which applied to pupils in their groups, two options were the most common; that pupils both plan and create the game art themselves (7) and that they find reference on the internet, which can be a character from another game, series or a movie, and make game art based on that reference (7). In comparison to the two leading options, less common were options that pupils would download pictures or game assets from the internet (3) or would ask a friend to make it for them (3). The option with the lowest score was that pupils wouldn't replace placeholders at all, leaving the project with only basic shapes (1). According to these data it can be concluded that most programmers at the course do use game art in their projects, and majority of them make the effort to create it themselves.

The second question aimed to identify when the game art is added to the game. Teachers' responses are displayed in Fig. 2. The highest scored answer was that game

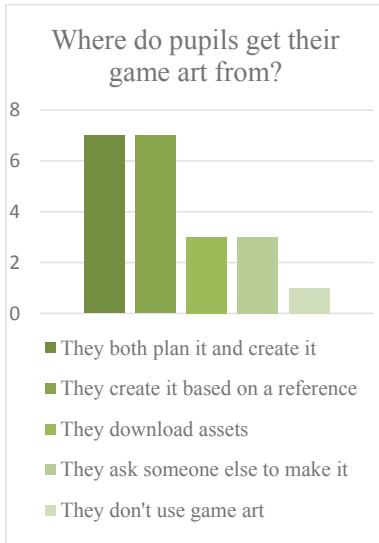


Fig. 1. Source of the game art in pupils' projects

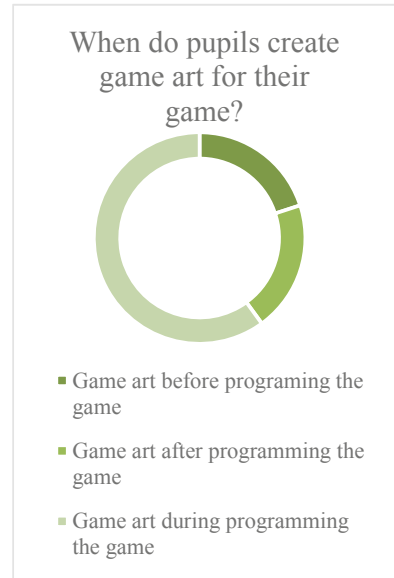


Fig. 2. The phase of the project in which pupils create or add game art in their projects

art is added throughout the game-making process (6), while marginal options, either in the beginning (2) or in the end of the project (2), scored lower.

There are several reasons why pupils would add game art throughout the project. Based on one of author's personal experience from teaching at this course, one of reasons could be that this choice comes naturally to them in this form of learning. While they are designing the game, their initial idea often transforms into something different; they discover a new functionality or think of a different context for the game and the project then must accommodate these changes. Another possible reason could be linked to the results of the previous study. According to teachers, planning is one of the lead skills that pupils acquire at this course over time, and before they acquire it, the beginners often struggle [13], which could manifest in various aesthetical changes as well.

4.2 Description of Functions of Game Art Tasks

The analysis of the data from interviews helped identify and describe eight different functions that game art activities fulfilled in the programmers' learning process. The list of functions is the result of open coding and categorisation of teachers' responses, and this list includes:

- **Game art as rest time:** Programmers create game art when they simply do not have sufficient energy for programming in the beginning stage of the project. Other times it

may happen when they stop programming for the day, but they still want to continue working on the project in some way, other than programming.

- **Game art as a training of fine motor skills:** Teachers mentioned that pixel art helped young pupils familiarise themselves with mouse and keyboard and that it improves their overall skills behind the computer.
- **Game art as a small victory:** The process of creating a complex project such as a computer game is exhausting and can take a lot of time, sometimes even two to three months. Teachers notes that this is one of the reasons why programmers like making pixel art – because they can create something valuable in a shorter period of time and get a sense of accomplishment from it.
- **Game art as a change of activity:** Programmers often turn to game art for a change of pace after they spend longer period of time programming. This allows them to focus on a different goal within the project, for example what kind of world the game is situated in or how will the player or enemies look like.
- **Game art as motivation for programming:** The main difference between creating pixel art and creating pixel art for a game is the purpose. Pixel art becomes more meaningful to pupils when they incorporate it into their projects, which is also a source of motivation for them, that they can use their art in their game.
- **Game art as an improvement of their game:** If we look at various games, more often than not they can be stripped down to the same set of game mechanics; the most noticeable difference is often in the story and its graphic design. Pupils also like creating game art to their games, because it makes the game feel more like “a real game”.
- **Game art as self-realisation:** The reason why many programmers like pixel art in general is that it is a fun activity. They like playing with pixels and shapes and how even a weird outline can become a collectible item or a spaceship. The activity leaves them in a better mood.
- **Game art as a way to regulate perfectionism:** When creating something artistic, it can be difficult to not to feel demotivated by not being able to translate the mental image onto the canvas. According to teachers, pixel art frees pupils from this constraint. They are free to experiment with random shapes and lines, creating an imperfect or an exaggerated representation of an object.

Having identified the functions, the next step was to measure which functions were the most common among programmers or the most meaningful to them. In the next section, we describe the results of individual functions based on Likert scales.

4.3 Functions of Game Art Tasks

Since the functions were identified in the interviews by teachers themselves, it was not surprising to see that almost all of them scored above 3.0. Having considered this as well as the size of the sample, we decided for the benchmark of a meaningful function to be above 4.0.

When analysing teachers’ responses, determiners of score for each function were the function’s mean – the average value, mode – which was important when considering the difference between average value of the function and its most common value, and

Table 1. The scores of each function of game art tasks

Function	Mean	Modus	SD
Rest time	4.2	5	1.0328
Fine motor skills training	2.3	1	1.2517
Change of activity	4.3	4	0.6750
A small victory	4.2	4	0.7889
Motivation for programming	3.1	2	1.3702
Improvement of a game	3.1	4	1.3702
Self-realisation	4.1	5	0.8756
A way to regulate perfectionism	3.5	4	1.4337

standard deviation (SD) – to determine the accuracy or uniformness in teachers’ opinions. When two functions reached the same mean, we first considered SD to determine which function teachers agreed on more uniformly, and when those were the same as well, then the modus determined their order. All values are displayed in Table 1 above (Fig. 3).

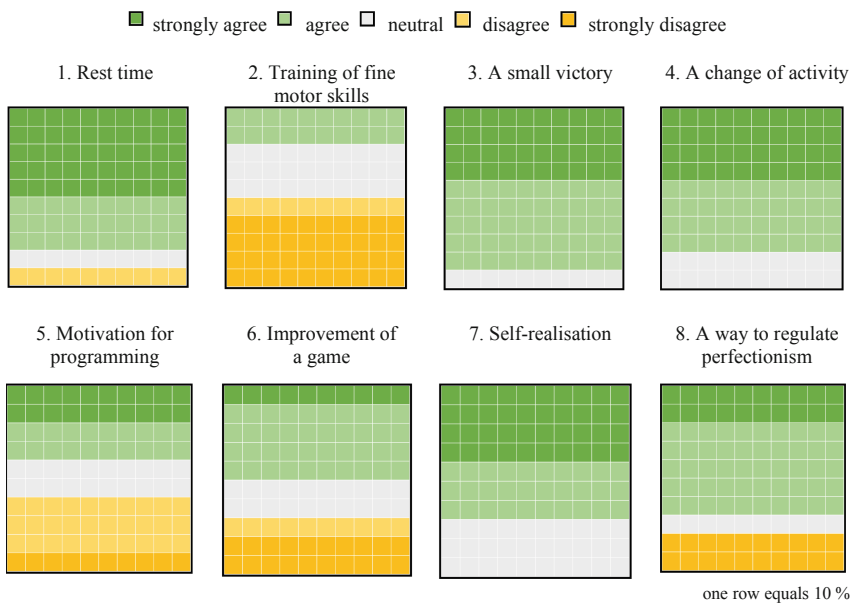


Fig. 3. Likert scales: teachers’ replies to functions of game art tasks for programmers

The analysis determined that based on the mean, four functions out of eight that teachers identified in the interviews were meaningful and important for pupils in their groups. The most important function seemed to be the change of activity (mean 4.3, modus 4, SD 0.6750), followed by game art as small victory (mean 4.2, modus 4, SD 0.7889)

and rest time (mean 4.2, modus 5, SD 1.0328) and finally game art as self-realisation (mean 4.1, modus 5, SD 0.8756).

The other four functions scored under 4.0 and were identified as less common among their groups. Game art's function as a way to regulate perfectionism in pupils was the highest scoring function under 4.0 (mean 3.5, modus 4, SD 1.4337), followed by game art as improvement of pupil's game (mean 3.1, modus 4, SD 1.3702) and motivation for programming (mean 3.1, modus 2, SD 1.3702). The function which scored the lowest was the function of game art for practice of fine motor skills (mean 2.3, modus 1, SD 1.2517).

When analysing SDs of these functions, we can notice differences between the two groups. SDs of meaningful functions were in the first three cases lower than or in one case very close to 1. When compared to the other four functions whose SDs were all higher than 1, the data suggest that teachers were more uniform about the importance of the first four functions than the other four, which again underlines the importance of the first four functions.

Beginners and Proficient Programmers. One of the open-ended questions in the questionnaire was about the differences between beginners and proficient programmers. According to teachers, the functions of game art differ between the two groups.

The data suggests that the feeling of accomplishment is very important to the beginners, as well as changing focus by switching between programming and game art. They are also more motivated to finish the game when they see that their game art can be imported into their project.

On the other hand, teachers noted that proficient programmers seem to appreciate the change of activity the most. Moreover, the awareness of the game becomes more apparent since they perceive game art as a way to improve the quality of the finished game.

These results seem to align with the expected growth in difficulty of programming projects and the growth of skills over time at this course, as suggested by a previous study [13]. Teachers noted that the beginners need a push in the beginning and need to learn to pace themselves throughout a bigger project, while they do not see this need in proficient programmers. Those view pixel tasks more as a change from programming, and one which increases the quality of the finished product, which becomes more important in later phase of the learning process.

After discussing whether they create game art, when they create it and why they do it themselves, the remaining issue is what they create. It is common practice at the course that those who want to learn game art always begin by completing pixel art tasks and eventually move on to finer drawings or vector art. Pixel art is the first option everyone at the course encounters, and it is the option programmers at the course rarely stray away from. In the next section, teachers' observations are described as to why it may be so.

4.4 Why Pupils Like Pixel Art Tasks

Pixel art had always been a very popular choice among pupils at GameCraft, especially among programmers. Teachers attribute this popularity to characteristics such as its simplicity, learning curve and its limitations.

The tool that is commonly used at the course is free online editor Piskel². Despite its minimalistic nature, it offers all necessary tools, e.g. single-pixel brush, fill bucket, colour drop, selection tool or eraser. Although there are also options for more advanced techniques, pupils usually find the basic tools sufficient for their work. Teachers mentioned that pupils appreciate the minimalistic interface and that it is easy to navigate and to learn how it works.

Not only tools are simple but creating and editing sprites is easy too. Teachers commented on the fact that even younger pupils were able to learn pixel art quickly, which is especially appreciated by programmers. Since the canvas is by default 32×32 pixels, the space fills up quickly and they choose how much detail they apply. According to teachers, they also appreciate the simple shapes that are the basis of the designs and that although sprites may lack details, they can still be perceived as good game art.

The final point is the limiting nature of pixel art. Several teachers mentioned the fact that since pupils work within a grid made of squares that they fill in, the activity resembles antistress colouring books rather than drawing. The grid also reminds pupils of games that they are familiar with such as Minecraft or games on the Roblox platform, which it is also where many boys draw their inspiration from (Fig. 4).



Fig. 4. Interface of Piskel online editor

When asked about pixel art tasks' weaknesses, teachers emphasised three aspects: improvement of skills, animations and again its limitations.

The most common comment of teachers was that while it is very easy to start, it gets more difficult when pupils start to incorporate more details into their sprites. To add details, the canvas must be resized, which also means smaller pixels, and it is sometimes difficult to find the balance between the amount of details that programmers want to add and the effort they are willing to invest into a larger sprite. Moreover, pixel art as an art style has its specific techniques, and since pupils often do not want to learn them correctly, they are limited by their own skills.

Another commonly mentioned point was creating animations for games, which is another popular activity among programmers, especially less complex animations, e.g.

² Online editor Piskel is available at <http://piskelapp.com>.

a blinking spaceship, explosions or idle movement of sprites. To create animation in Piskel is not a difficult matter, but as teachers noted, it requires planning ahead, which is difficult for some pupils, and it may result in starting over or redrawing a number of frames. Previous research suggests that the planning skills of programmers improved as they mastered the game-making process at the course [13], but for beginners who still lack the skill or patience it can be discouraging.

Although the simplicity of the art style is one of its great advantages, it can also be a disadvantage. Teachers noted that while some aspects are simplified, others are even more difficult, especially drawing curves and circles on a small canvas. They also mentioned that although the pixel art style is still popular in the game industry, it is different from high resolution 3D graphics that new games offer and that children are getting used to. As a result, some pupils may perceive their pixel art as childish in a way. This does not seem to be common, though, and it is usually compensated for by collaboration with a more skilled game artist at the course.

5 Discussion

The analysis of the data showed that most programmers at GameCraft create game art for their own games and many of them do so throughout the process of creating a game. Looking at the functions which the analysis determined as meaningful, we begin to understand why. Considering that to create a game takes a lot of time, sometimes weeks to months, it seems reasonable that programmers would turn to game art to seek change of activity from programming, to rest, to experience a small accomplishment or simply to enjoy a different activity.

When comparing pupils in different stages of learning programming, there were some similarities as well as differences with the overall results. Firstly, the function which scored as the most meaningful – the change of activity – proved to be important for both beginners and proficient programmers. However, teachers considered two of the functions to be more important in specific stages of learning despite their lower score in the overall analysis. For beginners, it was game art's motivation for programming - according to teachers, pupils get motivated by seeing their art used in their projects. And as for the proficient programmers, teachers highlighted game art's ability to improve the quality of the game. Proficient programmers seem to be more aware of this and place more effort into how their game is visually perceived, while in beginners' projects it is often about the functionality of a game, not its visual aesthetics.

When choosing the method for game art tasks, teachers' experience suggests that pixel art was effective for engaging pupils in game art. Though programmers were sometimes hesitant to create game art, pixel art made it easier to begin and it was quick to learn, which teachers considered very important in the beginnings.

Change of activity was a function which scored the highest in the overall analysis and as it was mentioned, teachers emphasised its importance for both beginners and proficient programmers. These results probe the question why. One of the possible explanations found in literature has to do with the phenomena of fixation and incubation [11]. Perhaps, when pupils switch focus to a different type of task - such as filling in pixels on a small canvas – it creates a needed distraction from programming, one that could unblock the

flow of new ideas when solving problems. Another perspective is offered by Ackermann, who noted that putting distance between pupils and their projects, or stepping-out of experience, which in our case would be achieved by game art tasks, could help pupils gain an outsider's perspective and internalise new knowledge before reengaging with the project [15]. However, another research would be necessary to explore the issue further, as this is beyond the scope of this study.

Looking at the big picture, all of the significant functions of game art in this research – change of activity, a small victory, rest time or self-realisation – point in the same direction. Pupils' needs. And although there might be other benefits to game art, e.g. creativity, planning or organisation of files, we believe that building good practices such as self-regulation in the learning process or awareness of their wellbeing while programming are some of its greatest benefits. The current data suggest that it might be so at this game-making course, however, more research would be necessary to confirm and assess whether these results were bound to the learning ecology of this course or if it would transfer to other ecologies, such as classroom environment.

6 Conclusion

The purpose of this mixed methods research was to explore pupils' relationship to game art and roles that game art tasks might play when programming computer games. The data analysis showed that the four most meaningful functions of game art were game art as a change of activity, as rest time, as a small victory and as self-realisation, and it hinted at the fact that different functions may be relevant for different stage of learning programming. Moreover, the analysis of pixel art tasks noted advantages of implementing this method when introducing pupils to creating game art.

The process of creating a game has many aspects pupils can learn from, and this research described what creating game art can offer. Incorporating these tasks into the creation process could encourage good practices, such as pacing themselves through the learning process, motivating themselves by experiencing small successes or to have agency over their learning.

Acknowledgements. We would like to thank teachers from GameCraft course for their cooperation and project VEGA 1/0602/20 and grant UK/161/2021, thanks to which the results of this research could be published.

References

1. Fields, D., et al.: Working toward equity in a constructionist Scratch camp: lessons learned in applying a studio design model. In: *Constructionism in Action: Conference Proceedings Constructionism*, pp. 290–297 (2016)
2. Czakoová, K., Stoffová, V.: Training teachers of computer science for teaching algorithmization and programming. In: *14th International Multi-Conference on Society, Cybernetics and Informatics, Florida*, pp. 231–235 (2020)
3. Werner, L., Denner, J., Bliesner, M., Rex, P.: Can middle-schoolers use Storytelling Alice to make games?: results of a pilot study. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*, pp. 207–214 (2009)

4. Petri, A., Schindler, C., Slany, W., Spieler, B.: Game design with pocket code: providing a constructionist environment for girls in the school context. arXiv preprint [arXiv:1805.04362](https://arxiv.org/abs/1805.04362) (2018)
5. Salen, K.: Gaming literacies: a game design study in action. *J. Educ. Multimed. Hypermed.* **16**(3), 301–322 (2007)
6. Denner, J., Werner, L., Ortiz, E.: Computer games created by middle school girls: can they be used to measure understanding of computer science concepts? *Comput. Educ.* **58**(1), 240–249 (2012)
7. Weintrop, D., Holbert, N., Horn, M., Wilensky, U.: Computational thinking in constructionist video games. *Int. J. Game-Based Learn.* **6**, 1–17 (2016)
8. Coenraad, M., et al.: The effects of providing starter projects in open-ended scratch activities. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pp. 38–44 (2021)
9. Papert, S., Harel, I.: Situating constructionism. *Constructionism* **36**(2), 1–11 (1991)
10. Innovated National Slovak Curriculum: Informatics (2014). <http://www.statpedu.sk/sites/default/files/dokumenty/inovovany-statny-vzdelavaci-program/informatika-nsv-2014.pdf>
11. Smith, S.M., Blankenship, S.E.: Incubation and the persistence of fixation in problem solving. *1st Am. J. Psychol.* 61–87 (1991)
12. Karpelová, M.: Faktory zvyšujúce žiacky záujem o programovanie na kurze tvorby počítačových hier [Factors increasing pupils' interest in programming at a game-making course]. In: *Proceedings of DidInfo 2021, Univerzita Mateja Bella*, pp. 102–106 (2021)
13. Karpelová, M.: What we have learnt about the effects of making computer games from primary school pupils. In: *Proceedings of the 17th International Scientific Conference eLearning and Software for Education, Bucharest* (2021). [in press]
14. Corbin, J., Strauss, A.: *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications (2014)
15. Ackermann, E.: Perspective-taking and object construction: two keys to learning. In: *Constructionism in Practice*, pp. 39–50. Routledge (2012)



Teaching Recursion in High School

A Constructive Approach

Dennis Komm^{1,2}(✉)

¹ PH Graubünden, Chur, Switzerland
dennis.komm@phgr.ch

² Department of Computer Science, ETH Zürich, Zürich, Switzerland

Abstract. There is little doubt about both the importance and at the same time difficulty of teaching recursion as part of any sophisticated programming curriculum. In this paper, we outline an approach that has its focus on introducing the concept in very small steps, integrating recursion into a K-12 programming class. The main paradigm is to be as constructive as possible in that everything that is introduced is also implemented right away from the first lesson on.

Keywords: Programming education · Recursion · K-12 · Turtle graphics

1 Introduction

Many computational problems have elegant solutions that utilize *recursion*, such as the famous Towers of Hanoi [16]. As a result, recursion should be part of informatics K-12 education as a rather natural application of *problem decomposition* (in particular, *divide-and-conquer*), which in turn is a cornerstone of computational thinking [9, 15]. However, the topic is not only important but also rather challenging for novice programmers [4], and therefore a careful introduction is necessary.

In this paper, we propose a concrete roadmap, assuming a setting where students have prior knowledge about imperative programming.

There is a wealth of literature about examples that describe how to teach recursion – or how to “think recursively” – in a high school context, some of which dates back decades by now; see Rinderknecht [13] for a survey. Many of these approaches focus on the idea of introducing divide-and-conquer as a first motivation, practicing this technique of decomposing a given problem into subproblems “unplugged,” and after that possibly using a computer to put the ideas into code, see, for instance, Hauswirth [5], Anderle et al. [1], or Sysło and Kwiatkowska [17].

Without criticizing these approaches, in this paper we would like to offer an alternative strategy, which – although we are speaking about a top-down concept – somewhat follows a bottom-up approach. In a nutshell, we would like to be as “hands-on” as possible in that we put programming in the center, and only move

to more complex algorithms once terms like *base case* are understood and have been successfully implemented. Thus, we start with very simple programs that very carefully introduce core concepts one after the other, and have the students put everything they learned into code as soon as possible. In particular, the concept of divide-and-conquer is only introduced at the very end, at a point where the students are able to actually implement the ideas described on a high level.

The individual ingredients used in this paper are well known; our main contribution is a detailed recipe that puts them into context. We give step-by-step instructions on how all concepts involved can be taught in a constructive fashion. The result is an exemplary lesson plan with examples and exercises that enable the students to master basic recursion in small steps.

Didactically, our approach is in parts inspired by the PRIMM [14] principle, although we do not always include all of its steps. The students are given code (that can be executed as it is presented), reason about it, modify single parts of it to understand what it does, and only after that write their own (similar) programs.

The following points are crucial to our approach.

- Introduce one concept after the other in small steps.
- Apply new concepts as soon as possible.
- Use complete examples that can be executed as they are presented.
- Consequently build on the students’ preknowledge.
- Build bridges to known concepts.

2 Road Map

In the main part of the paper, we describe a road map that consists of seven steps. Within these steps, the single aspects that make up recursive programming are described. Whenever a new aspect is introduced, it is immediately applied by the students by writing (Python) code, which is why we call our approach “constructive.” We indeed follow many of Papert’s *constructionist* ideas including programming the Turtle in the first steps of our lesson plan [11, 12].

The lesson plan can be subdivided into seven steps, with a zeroth that is prepended in the first 15 min. A single lesson takes 45 min.

0. Give a quick outlook on what is going to happen (Lesson 1).
1. Introduce recursion using an example (Lesson 1).
2. Link recursion to something known (Lesson 2).
3. Point out the importance of a base case (Lesson 3).
4. Combine recursion with return values (Lesson 4).
5. Allow multiple calls per function body (Lesson 5–6).
6. Implement known algorithms recursively (Lesson 7).
7. Introduce divide-and-conquer (Lessons 8 and on).

Two non-trivial aspects of recursive programming are (1) having a function return a value and processing it recursively, and (2) having more than one recursive call in a single function body. We introduce both of them carefully, and even avoid them completely in the first steps. Only in step 4 values are computed and returned to the caller, and only in step 5 the programs actually branch. In step 7, both concepts are combined for the first time.

In the following subsections, we give a quick overview over the students' programming experience that is assumed, and then describe the seven steps in more detail.

2.1 Preknowledge

Our lesson plan is designed for grades 11 or 12 on the premise that the students already have some experience with programming in Python. In particular, they can define simple functions that include conditionals and loops, use variables, and implement some standard algorithms such as Binary Search and Bubblesort. We have been paying special attention to the introduction of *variables*. First, we have introduced parameters as a kind of “read-only”-variables and only after that considered variables changing their value during execution. The reason is that many misconceptions arise in particular when transferring the concept of variables from mathematics to informatics [7].

Additionally, the students are familiar with Turtle graphics, in particular the `gturtle` library, which is part of the TigerJython environment [18, 19], which the students have used for two to three years by the time this lesson plan should be applied.

We would like to remark that functional programming is not part of this curriculum, and that recursion is therefore introduced after the students gained experience with imperative programming. Dynamic data structures, which may have a recursive nature, have also not been discussed. The students use Python lists utilizing both random access and dynamic aspects, but without having discussed any details of the implementation.

The idea of computational complexity was covered informally by, for instance, reasoning about the number of elements that Binary Search inspects given a sorted Python list of n elements. Big-O notation and other formal tools have not been introduced.

2.2 Give a Quick Outlook on What Is Going to Happen

Before any concepts are introduced formally, we excite the students' curiosity by giving them an idea of what is covered within the following weeks. However, it is not our goal to dive into recursive algorithms right away, but only to build up some tension about an interesting and powerful tool that will be introduced and used. We therefore show them pictures of fractals and point out their self-similar structure. It is obvious that these pictures are hard to design with the programming concepts the students know at this point. But lo and behold! Not

Listing 1: A first recursive program

```

1 def f():
2     print("Hello world!")
3     f()

```

Listing 2: Using a parameter

```

1 def f(k):
2     print(k)
3     if k > 0:
4         f(k-1)

```

Fig. 1. Introducing recursion using two simple programs

only will the new tool allow for drawing such pictures with rather little code, but also for solving many interesting problems in a clever and elegant way.

However, before the students can use this exciting tool, they have to understand the way it works, which is what the following seven steps (subsections) are devoted to. For now, there will not be more motivation, but we will soon get back to the things recursion brings to the table.

2.3 Introduce Recursion Using an Example

We start by asking the students what they think happens if a function “calls itself” by introducing a very short toy example of a recursive function without parameters as shown in Listing 1 (Fig. 1). Note that it may be necessary to adjust some of the metaphors we have used so far. Defining a function may have been thought of as “teaching the computer a new word,” but this analogy makes somewhat less sense if the definition of a word again contains this word.

Studying Listing 1, we discuss with the students what will happen when executing such a program by expanding it using the blackboard, and then running it on a computer. There are two insights: (1) the program will theoretically run forever, but also (2) this does not seem to be prohibited; so far, there is no obvious reason to write code this way, but also nothing seems to prevent us. The students do, however, know that algorithms are supposed to work in finite time, and thus such behavior is at least somewhat undesired. (Conversely, they are not familiar with the concept of a call stack at this point, and we also do not address the finiteness of the computer’s memory.)

So we ask the students what to do about having such a program “run forever,” and they already know the answer, namely to use a parameter with a value that is changed with every call. We show them Listing 2 (Fig. 1), asking them to study the code and reason what it does, then to manually execute the program by hand, and finally using the computer in order to see whether they were right.

2.4 Link Recursion to Something Known

The second step aims at demystifying recursion by building a bridge to known concepts. First, we use the Turtle in order to draw simple shapes. The reason is that the Turtle gives us a tangible notional machine [3, 6, 8] that can be observed while executing the code. Ideally, the students are familiar with the Turtle since primary school where another programming language, for instance, Logo, was

Listing 3: Infinite spiral

```

1 def line(d) :
2   forward(d)
3   right(90)
4   line(d+3)

```

Listing 4: Finite spiral

```

1 def line(d) :
2   if d <= 300:
3     forward(d)
4     right(90)
5     line(d+3)

```

Listing 5: Finite spiral

```

1 def line(d) :
2   while d <= 300:
3     forward(d)
4     right(90)
5     d += 3

```

Fig. 2. Drawing a spiral recursively and with a loop

used instead of Python. The Turtle is navigated using the self-explanatory commands `forward()`, `backward()`, `left()`, and `right()`. Using the `gturtle` library, no object-orientation is needed; the command `makeTurtle()` creates a canvas with the Turtle in its center, which then executes the commands step-by-step.

The students are now presented Listing 3 (Fig. 2) while being asked to figure out what it draws. The task is again to execute the algorithm by hand and then use the computer afterwards. Executing the code leads to the Turtle drawing an infinite spiral. Applying what was learned in Sect. 2.3, the students are now asked to set the maximum side-length of the spiral to 300 pixels, which can be easily done by inserting an `if`-statement in line 2 of Listing 3, resulting in Listing 4 (Fig. 2). This is the same strategy as used in Listing 2, which avoids explicitly defining a base case (although it would not hurt).

Second, we compare Listing 4 to Listing 5 (Fig. 2), which draws the same spiral, but uses no recursion but a loop, pointing out that the main difference is that the variable `d` is now increased in line 5 instead of passing its increased value to an according recursive function call. Further exercises practice converting simple recursive code to non-recursive code and vice versa; an example is drawing a number of steps as shown in Listings 6 and 7 (Fig. 3).

Of course, we have to be careful at this point, as the equivalency to loops is not that easily established in general, but only works smoothly for such simple *tail recursions*.

2.5 Point Out the Importance of a Base Case

So far, we have not used the term *base case* and at the same time avoided Python's `return`-statement – although the latter is known to the students. However, novices struggle a lot with the difference between `return` and `print()`. Before actually returning values in the context of recursion, we therefore make an intermediate step and use `return` in order to end function calls. The reason is to very explicitly study *what* is actually ended, namely only the current function call, not all calls to this function, and not the whole program.

As an example, consider Listing 7, which was created in the preceding step. We show the students that this code can be rewritten in a way that makes the non-recursive case, the base case, more explicit; see Listing 8 (Fig. 3). Here we use

Listing 6: Iterative stairs	Listing 7: Recursive stairs	Listing 8: Base case
1 def step(d):	1 def step(d):	1 def step(d):
2 while d >= 1:	2 if d >= 1:	2 if d == 0:
3 forward(50)	3 forward(50)	3 return
4 right(90)	4 right(90)	4 forward(50)
5 forward(50)	5 forward(50)	5 right(90)
6 left(90)	6 left(90)	6 forward(50)
7 d -= 1	7 step(d-1)	7 left(90)
		8 step(d-1)

Fig. 3. Drawing stairs with a loop and recursively

return to simply end the function call, which again can be linked to something known, namely using Python’s **break** command within a “**while True**”-loop.

Also here, tasks are given to the students afterwards to consolidate what they have just learned. However, the students also already know that **return** does not simply end a function call but that there is usually an expression appended, which is first evaluated and then returned. (As a matter of fact, **return** returns `None` in Python, which is not discussed in class at this point.)

2.6 Combine Recursion with Return Values

Now the students understand that functions can call themselves, and by making sure that there is a non-recursive base case that is guaranteed to be called at some point, an infinite execution can be prevented. For the next step, we leave the Turtle for a short time and present the code displayed in Listing 9 (Fig. 4), which is the standard example of recursively computing the factorial of a natural number n . (Note that we have deliberately not defined the factorial of zero, and the **else**-statement in line 4 is not necessary.)

However, instead of starting with this function, we only present it after having discussed thoroughly what happens when functions call themselves. In this step, the first task for the students is to again reason about what the program does without actually executing it. The results are discussed in class, and the algorithm is executed by hand for small values of n . Together with the students, we sketch how the functions are called, and how the computed values are passed

Listing 9: Computing the factorial recursively	Listing 10: Computing the sum recursively
1 def fact(n):	1 def sum(n):
2 if n == 1:	2 if n == 1:
3 return 1	3 return 1
4 else:	4 else:
5 return n * fact(n-1)	5 return n + sum(n-1)

Fig. 4. Recursive implementations of the factorial and sum

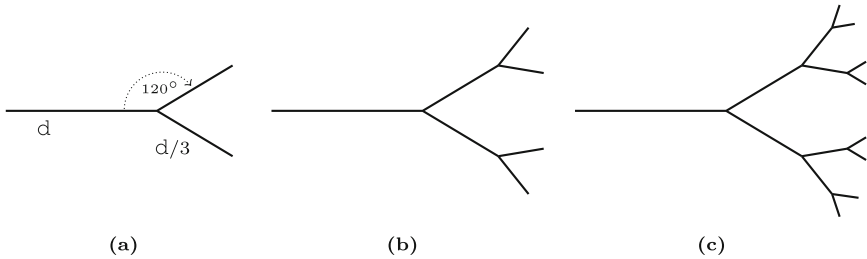


Fig. 5. Three binary trees with increasing depths

to the caller; we do still not formally speak about the concept of a call stack here. After that, we let the students investigate single components of Listing 9, asking them, for instance, what they think happens if we change “**return 1**” to “**return 2**” in line 3, etc.

Only after that, the students write their own code, which involves returning values, but sticks with one recursive call per function body. A sample task is to compute the sum of the first n natural numbers recursively, which is of course quite similar to computing the factorial of n ; see Listings 9 and 10 (Fig. 4).

2.7 Allow Multiple Calls per Function Body

In this step, we return to the Turtle and use it to draw fractals, redeeming our promise from Sect. 2.2. In our approach, the students do not have to learn any new tool or advanced programming techniques besides recursion itself; everything can be done using the very basic movement commands of the Turtle introduced earlier.

Drawing fractals now for the first time adds concrete value to the new technique. Indeed, it is quite obvious that the factorial of some number can also be computed with a loop instead – the similarities between both concepts were even discussed explicitly in step 2 (for such simple cases). As for computing the sum, there is even a closed formula, which may be known by some of the students. However, drawing self-similar shapes using loops is not straightforward at all, but quite elegant using recursion.

We again proceed in small “sub-steps.” The first task is to draw the simple structure in Fig. 5a using a simple function with a single parameter d , making sure the Turtle starts and ends at the left. Then, using this function, the second task is to draw the shape in Fig. 5b, and the third task to use this function to draw the shape in Fig. 5c.

Of course, no recursion is needed to draw any of these shapes, but our goal is to have the students identify a pattern. The next task therefore asks them to compare the three solutions in Listings 11 to 13 (Fig. 6), study where they differ and what they have in common, before finally trying to use this insight to define a recursive function `tree()` with two parameters d and n , the first of which denoting the length of

Listing 11: Binary tree 1	Listing 12: Binary tree 2	Listing 13: Binary tree 3
1 def tree1(d):	1 def tree2(d):	1 def tree3(d):
2 forward(d)	2 forward(d)	2 forward(d)
3 left(30)	3 left(30)	3 left(30)
4 forward(2*d/3)	4 tree1(2*d/3)	4 tree2(2*d/3)
5 back(2*d/3)		
6 right(60)	5 right(60)	5 right(60)
7 forward(2*d/3)	6 tree1(2*d/3)	6 tree2(2*d/3)
8 back(2*d/3)		
9 left(30)	7 left(30)	7 left(30)
10 back(d)	8 back(d)	8 back(d)

Fig. 6. Drawing the three trees from Figs. 5a to 5c, respectively

the first line (the “trunk” of the tree), and the second one the *depth* of recursion; see Listing 14 (Fig. 7).

This is the first time that the students are confronted with the concept of *branching*, that is, they now design functions with more than one recursive function call that is executed within a function body. Understanding what is happening here is crucial to master recursion. Therefore, the subsequent task asks the students to trace the calls when executing a concrete call to their new function, for instance, `tree(135, 2)`. The insight is that this also results in a binary tree (see Fig. 7a), whereas computing the factorial leads to a sequence of calls that could be arranged in a linear structure.

The floor is now open to draw further fractals such as, for instance, the Koch curve [10]. Note that we did not start with this example as the first line is only drawn in this example after a sequence of recursive calls, while Listing 14 draws

Listing 14: Recursive tree

```

1 def tree(d, n):
2   if n == 0:
3     forward(d)
4     backward(d)
5     return
6   forward(d)
7   left(30)
8   tree(2*d/3, n-1)
9   right(60)
10  tree(2*d/3, n-1)
11  left(30)
12  backward(d)

```

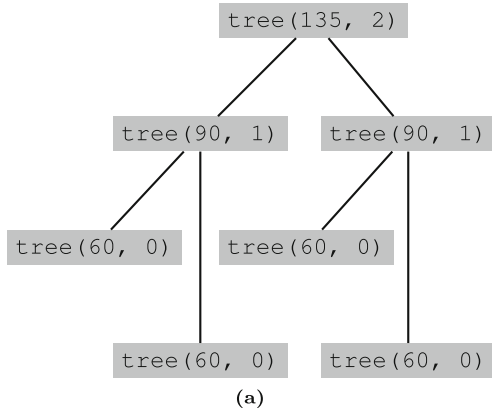


Fig. 7. Drawing binary trees recursively

Listing 15: Euclid's algorithm

```

1 def euclid(a, b):
2     while b != 0:
3         if a > b:
4             a = a - b
5         else:
6             b = b - a
7     return a

```

Listing 16: Euclid's algorithm recursively

```

1 def euclid(a, b):
2     if b == 0:
3         return a
4     else:
5         if a > b:
6             return euclid(a - b, b)
7         else:
8             return euclid(a, b - a)

```

Listing 17: Binary search

```

1 def bsearch(data, x):
2     l = 0
3     r = len(data) - 1
4     while l <= r:
5         c = (l + r) // 2
6         if data[c] == x:
7             return c
8         elif data[c] > x:
9             r = c-1
10        else:
11            l = c+1
12
13    return -1

```

Listing 18: Binary search recursively

```

1 def bsearch(data, l, r, x):
2     if l <= r:
3         c = (l + r) // 2
4         if data[c] == x:
5             return c
6         elif data[c] > x:
7             return bsearch(data, l, c-1, x)
8         else:
9             return bsearch(data, c+1, r, x)
10    else:
11        return -1

```

Fig. 8. Known algorithms and their recursive counterparts

something in any function call. This makes it a lot easier for the students to watch the Turtle execute their code. There is rich literature about other fractals and how to design creative pictures using recursion [10,16].

2.8 Implement Known Algorithms Recursively

The students have now gained some experience with recursive programming, and with fractals also encountered an example where this technique offers an obvious advantage over programming using iteration. For the last two steps, we move to algorithm design with the goal to use recursion to implement divide-and-conquer strategies in step 7 (Sect. 2.9).

Before we get there, however, we again build a bridge to something known in this step. As mentioned earlier, the students know about basic algorithms that have been implemented using loops so far. As an example, consider Euclid's algorithm to compute the greatest common divisor of two numbers a and b . We studied a very simple version without the modulo operator, as shown in Listing 15 (Fig. 8). We compare this known variant of the algorithm to the recursive implementation in Listing 16 (Fig. 8). Together with the students, we identify

Listing 19: Merging two sorted lists

```
1 def merge(leftlist, rightlist):
2     result = []
3     i_left = 0
4     i_right = 0
5     while i_left < len(leftlist) and i_right < len(rightlist):
6         if leftlist[i_left] < rightlist[i_right]:
7             result.append(leftlist[i_left])
8             i_left += 1
9         else:
10            result.append(rightlist[i_right])
11            i_right += 1
12    return result + leftlist[i_left:] + rightlist[i_right:]
```

a pattern that is similar to what we discovered when studying Listings 4 and 5 namely that the functions have in essence a very similar structure, but instead of explicitly decreasing the values of the variables `a` and `b` in lines 4 and 6 (Listing 15), respectively, these exact values are passed when calling the function recursively in lines 6 and 8 (Listing 16), respectively.

The next task revisits the Binary Search algorithm as presented in Listing 17 (Fig. 8). As this algorithm is, although somewhat easy to describe on a high level, very hard to implement [2], the single steps are recalled before asking the students to implement a recursive version as shown in Listing 18 (Fig. 8). The solution again follows the same structure as the original implementation, but instead of decreasing `r` in line 9 or `l` in line 11 (Listing 17), the values are accordingly passed in lines 7 and 9 (Listing 18), respectively.

2.9 Introduce Divide-and-Conquer

Only now we consider the students ready to have a close look at the divide-and-conquer strategy – this is our last step, not the first. In order to demonstrate the power of this technique, we start with the well known Mergesort algorithm. As mentioned in Sect. 2.1, the students already have some basic knowledge about sorting a Python list of n numbers in increasing order using, for instance, the Bubblesort algorithm. Depending on the level of the class, some simple observations about the number of swapping operations may have been made, exhibiting a quadratic complexity.

So we now return to the topic of sorting n numbers and present the observation to the students that “merging” two sorted lists to a single sorted list can be implemented in a straightforward fashion with two pointers, comparing their smallest elements, and inserting the smaller of the two at the end of an initially empty list. After thoroughly discussing this strategy, we ask the students to implement it in Python, resulting in a function as shown in Listing 19.

Listing 20: Mergesort recursively

```

1 def mergesort(data)
2     if len(data) <= 1:
3         return data
4     mid = len(data) // 2
5     leftlist = mergesort(data[:mid])
6     rightlist = mergesort(data[mid:])
7     result = []
8     i_left = 0
9     i_right = 0
10    while i_left < len(leftlist) and i_right < len(rightlist):
11        if leftlist[i_left] < rightlist[i_right]:
12            result.append(leftlist[i_left])
13            i_left += 1
14        else:
15            result.append(rightlist[i_right])
16            i_right += 1
17    return result + leftlist[i_left:] + rightlist[i_right:]

```

With this, the center piece of Mergesort is built, and we discuss in class how the merging can be applied recursively in order to sort a given list. This is done on the blackboard in an unplugged fashion, but as soon as it is understood, the students can use their new tool to transfer the ideas into code. The result is shown in Listing 20. Essentially, all that needs to be done is to rename the function from Listing 19, slightly alter it to take only a single list as parameter (line 1), take care of the base case (empty lists and lists with one element can be considered sorted, lines 2 and 3), then split the given list in half using Python’s known *slicing* notation, and apply the algorithm to the two resulting lists recursively (lines 4 to 6).

The complexity of this algorithm can be discussed on a high level, without introducing mathematical recursive functions, but arguing about the size of the tree representing the calls. Note that Listing 20 can be implemented in an even less cumbersome manner using Python’s `pop()` function to take the smaller of the two elements out of the respective list instead of using the two pointers `i_left` and `i_right`. However, since `pop(0)` has linear time complexity, this would reduce studying Mergesort to absurdity.

At this point, all concepts have been introduced to write recursive code, and in this step, all have been combined in order to implement a fast sorting algorithm. Now the students are ready to investigate other examples of recursive implementations of the divide-and-conquer strategy or study other problems with natural recursive solutions such as listing all words of a fixed length over some fixed alphabet or the initially mentioned Towers of Hanoi.

Acknowledgement. The author thanks Tobias Kohn for many interesting discussions that improved the paper.

References

1. Anderle, M., Forišek, M., Steinová, M.: Teaching recursion and dynamic programming before college. *Bull. EATCS* 129 (2017)
2. Bentley, J.: *Programming Pearls*, 2nd edn. O'Reilly Media, Newton (1999)
3. du Boulay, B., O'Shea, T., Monk, J.: The black box inside the glass box: presenting computing concepts to novices. *Int. J. Man-Mach. Stud.* **14**, 237–249 (1981)
4. McCauley, R., Grissom, S., Fitzgerald, S., Murphy, L.: Teaching and learning recursive programming: a review of the research literature. *Comput. Sci. Educ.* **25**(1), 37–66 (2015)
5. Hauswirth, M.: If you have parents, you can learn recursion. *Bull. EATCS* 123 (2017)
6. 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: Brodник, 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
7. Kohn, T.: Variable evaluation: an exploration of novice programmers' understanding and common misconceptions. In: *Proceedings of SIGCSE 2017*, pp. 345–350 (2017)
8. Kohn, T., Komm, D.: Teaching Programming and Algorithmic Complexity with Tangible Machines. In: Pozdniakov, S.N., Dagienė, V. (eds.) *ISSEP 2018*. LNCS, vol. 11169, pp. 68–83. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02750-6_6
9. Komm, D., Hauser, U., Matter, B., Staub, J., Trachsler, N.: Computational Thinking in Small Packages. In: Kori, K., Laanpere, M. (eds.) *ISSEP 2020*. LNCS, vol. 12518, pp. 170–181. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63212-0_14
10. Liss, I.B., McMillan, T.C.: Fractals with turtle graphics: a CS2 programming exercise for introducing recursion. In: *Proceedings of SIGCSE 1987*, pp. 141–147 (1987)
11. Papert, S.A.: *Mindstorms: Children, Computers, And Powerful Ideas*, 2nd edn. Basic Books, New York (1993)
12. Papert, S.A.: Introduction: what is logo? And who needs it? In: *Logo Philosophy and Implementation*, pp. IV–XVI. *Logo Computer Systems* (1999)
13. Rinderknecht, C.: A survey on the teaching and learning of recursive programming. *Inform. Educ.* **13**, 87–119 (2014)
14. Sentance, S., Waite, J.: PRIMM: exploring pedagogical approaches for teaching text-based programming in school. In: *Proceedings of WiPSCE 2017*, pp. 113–114. ACM (2017)
15. Wing, J.M.: Computational thinking. *Commun. ACM* **49**(3), 33 (2006)
16. Stephens, R.: *Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C#*. Wiley, New York (2019)
17. Syslo, M.M., Kwiatkowska, A.B.: Introducing Students to Recursion: A Multi-facet and Multi-tool Approach. In: Gülbahar, Y., Karataş, E. (eds.) *ISSEP 2014*. LNCS, vol. 8730, pp. 124–137. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_12
18. TigerJython. <https://tigerjython.ch>. Accessed 5 July 2021
19. WebTigerJython. <https://webtigerjython.ethz.ch>. Accessed 5 July 2021

Advancing Computing Education



A Multi-dimensional Approach to Categorize Bebras Tasks

Christian Datzko^(✉) 

School of Education, University of Applied Sciences and Arts Northwestern
Switzerland, Brugg-Windisch, Switzerland
christian@datzko.ch

Abstract. The Bebras International Challenge of Informatics and Computational Thinking has collected an increasingly large number of tasks. Every year more than 200 new task proposals are submitted of which many are accepted into the task pool. Properly categorizing these tasks to facilitate the use in the national challenges and to reuse them in the future is a challenge in itself: the categorizations in the past were biased and only covered certain aspects. Because the tasks are meant to be used everywhere in the world they are supposed to be solvable by students without any particular knowledge in computer science. General knowledge or skills may be assumed, though.

In this paper a different approach is presented that does not focus on creating perfect categories but rather on different dimensions within which categories must be defined. Such a multi-dimensional approach should offer the ability to find tasks more easily and will fix some of the common shortcomings of the previously used categorizations. The three dimensions covered are: (computer science) topic, Computational Thinking skills, and range of requirements.

Keywords: Bebras International Challenge of Informatics and Computational Thinking · Task preparation · Quality management · Computer science topics · Computational Thinking

1 Introduction

The Bebras International Challenge of Informatics and Computational Thinking (short: Bebras) is held in more than 68 countries around the world reaching about 3 million students annually. Its instances consist of 9 through 15 short tasks that are supposed to be solved within 40 min by the students. So a student has roughly 3 to 5 min to read, understand, and solve a task.

For the Bebras International Challenge of Informatics and Computational Thinking every year hundreds of task proposals are created, submitted to the International Bebras Community, improved, selected for national challenges, and translated and adapted to the local needs. Datzko [9] describes this process in detail. The number of task proposals and the number of tasks deemed fit for the Bebras challenge after the International Bebras Workshop are shown in Table 1.

Table 1. Number of task proposals and tasks in the task pool for Bebras from 2012 to 2021.

Year	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Task proposals	194	238	176	163	209	251	225	233	274	270
Task pool size	133	151	126	125	192	179	126	113	147	205

These tasks, however, are not just used for a single challenge. Since they are made available to the general public after the challenge they are being reused for specially compiled challenges [18], regular teaching (in Switzerland for instance [8, 10], and [12]), and even research (for instance [14, 21, 23], and [20]). Although traditionally published in the form of brochures (some are linked from [2]) some countries already publish them interactively in their challenge system (see for instance [24]) even allowing teachers to generate their own challenges. Discussions about creating databases with all tasks included are going on within the community.

This categorization proposal does not attempt to offer a complete curriculum for computer science education like the K–12 Computer Science Framework [1]. Besides the fact that some aspects of computer science are central to computer science education but not testable in a challenge (see for instance Sect. 2.5), a curriculum would have to offer much more detail than a categorization for the purposes mentioned above.

1.1 Open Issues with Categorization

One of the biggest challenges is to select the right tasks for a task set or one of the other uses mentioned above from the task pool [11]. Over the years different categorizations were used to help the National Bebras Organizations select the tasks. Dagiè, Hromkovič and Lacher [6] reiterate the history of the different categorizations. These categorizations have two things in common: they all mostly focus on the prominent computer science topic of the task and they all are somewhat skewed.

While the first observation is obvious, for the second observation the Table 2 shows the distribution of the tasks in their categories from 2020¹. One must keep in mind that this is merely an assertion by the original task authors or the editors of the International Bebras Community who worked on these tasks. It doesn't necessarily represent an absolute truth, but given that mostly computer scientists and computer science educators work jointly on these tasks at least it can be seen as a realistic trend.

Within the community some discussions went on about the categorizations and several people voiced their discontent with the fact that most tasks were categorized either as “Algorithms and Programming” or “Data, Data Structures,

¹ The total doesn't match the totals in Table 1 since for some tasks more than one category may have been chosen and for some tasks no category may have been chosen at all.

Table 2. Topics of the 2020 task proposals and task pool; ALP is short for “Algorithms and Programming”, DSR is short for “Data, Data Structures, and Representations”, CPH is short for “Computer Processes and Hardware”, COM is short for “Communication and Networking”, and ISS is short for “Interactions, Systems, and Society”.

Topic	ALP	DSR	CPH	COM	ISS
Task proposals	170	138	12	16	18
Task pool	91	77	12	12	8

and Representations”. Some even mentioned that almost every task of a sufficient complexity could be categorized in almost any of the categories and that thus the categorizations were somewhat arbitrarily.

1.2 Other Approaches

In 2017 Dagienė, Sentance and Stupurienė [7] proposed a two-dimensional approach for categorization. Their two dimensions are “Informatics Concepts: Knowledge Level” as well as “Computational Thinking: Skills Level”. These are explicated with the following categories on the knowledge level as well as on the skills level:

1. *“Algorithms and programming, including logical reasoning;*
 2. *Data, data structures and representations (including graphs, automaton, data mining);*
 3. *Computer processes and computer hardware (includes anything to do with how the computer works – scheduling, parallel processing);*
 4. *Communications and networking (includes cryptography, cloud computing);*
 5. *Interaction (Human–Computer Interaction, HCI), systems and society (all other topics!)” [7, p. 35–36].*
1. *“Abstraction;*
 2. *Algorithmic thinking;*
 3. *Decomposition;*
 4. *Evaluation;*
 5. *Generalization” [7, p. 37].*

They expect the editors of a task proposal to choose: *“In practical terms, a task should be allocated to one informatics content area only but may have up to three computational thinking skills identified” [7, p. 37].* So at most three checkmarks in their 5×5 table are placed.

Dagienė, Hromkovič and Lacher [6, p. 45] refer to [16] when writing: *“In Germany, a three-dimensional competence model was designed a decade ago for use in K-12 informatics education. It is argued that learners develop competency by engaging with content in different processes. Content and skills are seen as interwoven, the content can be identified from analyzing a skill and vice-versa. The third dimension describes a quality of engaging with content and skill, how*

much the learner makes the content and skills his or her own, which relates to dispositions. Hereby, the informatics related topics and methods of the competencies cover a broad range of subject areas of informatics". This model is a further developed version presented in [15]. The general approach is similar to the two-dimensional approach elaborated by Dagienė, Sentance and Stupurienė [7]. As a third dimension instead of checkmarks it offers "Anforderungsbereiche". The term "Anforderungsbereich" in German is widely used when preparing finals for high schools. They are related to Bloom's standard model for different levels of objectives and are described as (roughly translated from [16, pp. 3–4]):

1. Reproduction;
2. Reorganizing and transfer;
3. Deliberation and problem solving.

However, Dagienė, Hromkovič and Lacher stick to two dimensions: one dimension is "Competency of informatics" and the other is "Bloom's taxonomy". So in essence they have a list of competencies ordered in three subfields of informatics ("Data", "Algorithms", as well as "Technology: programming, robotics, networks"). For each of these competencies one or more of the six levels of objectives are marked. In other words to achieve this two-dimensionality by assigning competencies to topics which makes their categorization somewhat 2.5-dimensional with the topic and competency dimension merged into one. As a result they have huge list of competencies in subfields of informatics (around 100 different competencies with the possibility of around 3–4 different possible levels of objectives for each competency). In their discussion they ask for a further elaboration: this paper is such an approach.

2 Proposed Dimensions

2.1 1st Dimension: Topic

Dagienė, Hromkovič and Lacher [6] describe the history of the topic categories within the Bebras community. They all share the same problem: they're skewed as described above. Although Dagienė, Hromkovič and Lacher correctly ascribe a normative aspect to the categories, some topics are mostly put aside while for other topics year after year very similar looking tasks are proposed. This is understandable because for some topics it is quite simple to find student-level questions while for other topics it's inherently difficult. For instance for the topic of finding a shortest path in some graph is much more simple to find a challenging but solvable instance while for Turing machines it's much more difficult.

Dagienė, Hromkovič and Lacher also describe a two-dimensional approach by combining topics with Bloom's taxonomy. Their long list of detailed topics shows nicely that it is possible to formulate competencies on different topics with different levels needed. However, for a categorization that can be filled out within reasonable time too much detail may cost too much time. Therefore for

this approach only the general topics are chosen instead of a long list or a further split into more detailed categories.

The three main topics proposed are:

TOP 1 Data;

TOP 2 Algorithms;

TOP 3 Technology: Programming, Robotics, Communication.

2.2 2nd Dimension: Computational Thinking Skills

Computational Thinking has become a hot topic for computer science educators ever since Wing [25] proposed it in 2006. Unfortunately as she noted herself [26] the original definition wasn't very descriptive. So now different definitions from different viewpoints exist. The International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) [17] further created an operational definition of Computational Thinking. Calcagni et al. [5] also noted that this operational definition isn't suited best for short tasks like Bebras tasks because some of the skills are not really present. Also this operational definition somehow collides with topics.

According to Denning and Tedre Computational Thinking is *“the mental skills and practices for designing computations that get computer to do jobs for us, and explaining and interpreting the world as a complex of information processes”* [13, p. 4, highlighting by original authors]. It is therefore orthogonal to but of course interacting with the topic.

So I propose a further development to the thought processes involved:

CT 1 Thinking in terms of algorithms;

CT 2 Representing and structuring information;

CT 3 Deconstructing, generalizing, abstracting, or transferring a problem solving process;

CT 4 Optimize solutions or solution processes.

2.3 3rd Dimension: Range of Requirements

The requirements a student must fulfill to be able to solve a task varies for different competences. Bloom [3] created a standard model for different levels of objectives. While these still serve a very valid tool for analyzing the difficulties of tasks, the requirements of such an analysis in the contest of categorizing tasks are not so detailed. Approaches like from the German Kultusministerkonferenz [19] with a total of three different levels seems more fit. In order to make it even more simple to assess tasks a new set of levels of objectives is proposed. These “ranges of requirements” (RoR) are supposed to be quickly estimated and a person assessing a task proposal who knows the task proposal in detail should be able to select the right within seconds for each combination of topic and CT skill.

The proposed RoRs are:

- RoR 0 This task requires nothing or simple knowledge of given information or assumed common knowledge of this classification category to solve;
- RoR 1 This task requires comprehension or application of given information or assumed common knowledge of this classification category to solve;
- RoR 2 This task requires analysis or synthesis of given information or assumed common knowledge of this classification category to solve;
- RoR 3 This task requires evaluation of given information or assumed common knowledge of this classification category to solve.

RoR 0 corresponds to the first level of objectives of Bloom’s taxonomy or no objectives at all. RoR 1 corresponds to the second and third level of objectives of Bloom’s taxonomy. RoR 2 corresponds to the fourth and fifth level of objectives of Bloom’s taxonomy. RoR 3 corresponds to the sixth level of objectives of Bloom’s taxonomy. Like in Bloom’s taxonomy a RoR includes all lesser RoRs.

Unlike Bloom’s taxonomy which is based on already acquired knowledge, Bebras tasks are based on the principle that no special pre-knowledge is required to solve it. Of course, in some cases pre-knowledge might help and it’s not prohibited that students use it for a better performance but a task should be solvable within reasonable time without that pre-knowledge [4, p. 7]. Also common knowledge that can be assumed from every student at target age of a task is accepted. More precisely common knowledge is what at least 80% of the students at target age have acquired. So the formulation of the RoRs can be seen as Bloom-compatible once the student has thoroughly read the task.

Of course, RoR ratings are always open to interpretation and require a lot of assumptions. However, this concept of RoR ratings in the “Anforderungsbereich” variety (see Sect. 1.2) has been successfully used for many years in Germany for generating literally thousands of peer-reviewed and expert-reviewed written high school finals for many different subjects.

2.4 Another Dimension: Target Age?

Because in some cases common knowledge for the target age is assumed, assigning a range of requirements for a certain classification category to a task depends on which target age is considered. For instance a task which involves evaluating an expression with variables will likely be RoR 2 for students of age 8–10 while it might be RoR 1 for students of age 14–16. However, most task proposals are at most suggested for a span of 6 years of students’ ages. While it is common that the probability for any student to solve a task increases with age [11], the RoR ratings usually should differ by at most 1 between two extremes. In this case a typical age for a targeted student is supposed to be assumed and if the task is posed for younger or older students its ratings might be changed slightly. For a more thorough analysis it would make sense to write this down explicitly but for a categorization purpose it’s sufficient to ignore this dimension.

2.5 Another Dimension: Stances?

The Swiss curriculum for Computer Science for high schools [22] distinguishes between three types of main goals: “Grundkenntnisse” (basic knowledge), “Grundfertigkeiten” (basic abilities) as well as “Grundhaltungen” (basic stances). While the first two can roughly be associated with topics and computational thinking skills for this purpose the third type of main goal, the basic stances, are not covered. They are also not covered by the third dimension of the range of requirements. These basic stances contain main goals like (roughly translated) [the students are] “willing to understand means of informatics rather than simply being able to use them” [22, p. 5] or “are ready for teamwork and project work as well as interdisciplinary exchange” [22, p. 5].

Of course these are valid main goals for good computer science education. However, they are developed on long term and while some tasks might contribute to these main goals, it would be out of proportion to say that a single task makes a significant difference in any of these stances. Even a single Bebras challenge with 9 to 15 different tasks will likely not make much of a difference at all.

So for the purpose of categorizing task proposals it is safe to ignore this dimension as well.

3 Proposal

Combining the three dimensions into a single table results in a structure is presented in Table 3.

Table 3. A proposal how to easily implement the three-dimensional categorization into a single table.

	Data	Algorithms	Technology
Thinking in terms of algorithms	RoR 0-3	RoR 0-3	RoR 0-3
Representing and structuring information	RoR 0-3	RoR 0-3	RoR 0-3
Generalizing, abstracting, or transferring a problem solving process	RoR 0-3	RoR 0-3	RoR 0-3
Optimize solutions or solution processes	RoR 0-3	RoR 0-3	RoR 0-3

The proposal can easily be integrated into the current template used for task creation. Figure 1 shows the header of the current version of the template while Fig. 2 shows the header of a new version of the template implementing the proposed table.

In the LibreOffice version of the template (shown here) the 12 fields can be made clickable and the editor can easily select one of the four RoR ratings for each of the 12 combinations of topic and Computational Thinking skill.

2021-XY-01-eng Task Title, Doubleclick To Change!

6yo–8yo: _	8yo–10yo: _	10yo–12yo: _	12yo–14yo: _	14yo–16yo: _	16yo–19yo: _
Answer Type: Click To Choose (<input type="checkbox"/> keep order of multiple-choice/-select)					
Categories (click to choose):			<input type="checkbox"/> computer processes and hardware <input type="checkbox"/> communication and networking <input type="checkbox"/> interactions, systems and society		
<input type="checkbox"/> algorithms and programming <input type="checkbox"/> data structures and representations					

Fig. 1. The header of the current template used for task creation.

2021-XY-01-eng (mod) Task Title, Doubleclick To Change!

6yo–8yo: _	8yo–10yo: _	10yo–12yo: _	12yo–14yo: _	14yo–16yo: _	16yo–19yo: _
Answer Type: Click To Choose (<input type="checkbox"/> keep order of multiple-choice/-select)					
		<i>Data</i>	<i>Algorithms</i>	<i>Technology</i>	
<i>Thinking in terms of algorithms</i>		RoR 0	RoR 0	RoR 0	
<i>Representing and structuring information</i>		RoR 0	RoR 0	RoR 0	
<i>Generalizing, abstracting, or transferring a problem solving process</i>		RoR 0	RoR 0	RoR 0	
<i>Optimize solutions or solution process</i>		RoR 0	RoR 0	RoR 0	

Fig. 2. The header of a new version of the template implementing the proposed table.

4 Evaluation

This categorization proposal was sent to 50 long-term members of the International Bebras community with a request for comments. 40% responded.

The general feedback was mixed. About half of the feedback was very positive while the other half didn’t see an improvement above the system used right now. A closer look at the detailed reasons showed that some people fear that this (undoubtedly larger) proposal will take considerably more time and might be too complex for the community. This, however, was also the reason why others liked it as one person wrote: *“This framework is quite important for Bebras community since this will bring quality to the questions and the overall challenge. People will realize the importance of details and be careful while they are thinking about and creating the tasks.”* I interpret this as a conflict between the normative and the formative function of such an categorization model. While it indeed would take more time and might need an extra effort in order to get the community to use any such model more or less consistently, such an effort might have the positive side effect of raising awareness for more aspects (namely Computational Thinking as well as different levels of objectives) within the community thus fostering higher quality.

Despite the size of this proposal quite a lot of people thought that there were not enough topics. One even suggested a really long exhaustive list of

topics of which the task author could select. The one topic that was missed most was around social aspects. This is interesting to note because the vast majority of task proposals in the Bebras community does not address this topic (see Table 2). However, as one person wrote: *“If we do not teach it who will?”* Although, of course, there are a lot of opportunities for students around the world to get in touch with these aspects of computer science, it’s a worthwhile discussion which role the Bebras community and the Bebras challenge should play in teaching students. On the other hand (the same) person wrote: *“Any topic can fit into one of the topics with a bit of force.”* So the question whether the topics should be split up and complemented with further topics must remain open for now.

The Computational Thinking skills were largely accepted as well-selected. Some suggestions for improvement named concepts that are actually part of the range of requirements.

The concept of the range of requirements was viewed very positive.

I accept the notion that a more complex model for categorization takes an extra effort to implement and to maintain. Judging from the feedback it is worth proposing this to the community. It seems as if those who use Bebras tasks not only for the Bebras challenge but also for research seemed to like it even more as one person wrote: *“Moreover, it will be good to make research and test this categorization proposal whether it makes sense to Bebras stakeholders in terms of CT skills, requirements and topics.”* In order to implement it might help like another person wrote: *“It would be useful to have list of subtopics for all topics”*.²

5 Conclusion and Discussion

Making the categorization three-dimensional instead one-dimensional enables one to simplify the categories largely. Therefore simply three topics and four Computational Thinking skills are sufficient to categorize a task. The ranges of requirement make it possible that one does not have to choose a single category (or combination of a topic and a Computational Thinking skill) but rather to decide for each combination how much Computational Thinking is needed for that topic. This is also more realistic: most tasks require different Computational Thinking skills at different levels and quite a few tasks cover several topics.

Although Dagienė, Hromkovič and Lacher already proposed a multi-dimensional approach, their approach doesn’t make a difference between the topics and Computational Thinking skills involved. This makes it potentially difficult for a task to be classified since Computational Thinking skills are orthogonal to the topics. Also they restrict Bloom’s levels of objectives with their checkmarks. Leaving this interpretation up to the person classifying a task makes the model proposed in this paper more flexible.

The RoR rating of a task does not directly represent the difficulty of a task. Besides the fact that likely every task has at least some RoR ratings ≥ 1 the

² For this a combination with the proposal from Dagienė, Hromkovič and Lacher [6] might be worth a consideration.

difficulty of a task also lies in the complexity or size of actions required to solve the task. Also the time is limited so the task length contributes to the difficulty of a task [11].

In comparison to earlier categorization approaches this categorization requires more time to fill out. However, with literally thousands of task proposals in the Bebras community including the early task proposals this more granular approach promises to make searches for tasks in a potential database with all tasks more efficient. So this extension of the categorization can be seen as a good compromise between the time and effort needed to categorize a task and the information which helps searching for tasks.

Beyond the focus on categorizing tasks to construct a balanced task set for the national Bebras challenges, a more detailed categorization like the one proposed offer great benefits for the reuse of the tasks after the challenges. With such a categorization searchable databases with tasks can be used more easily.

This proposal probably isn't the last categorization proposal within the Bebras community. Besides the fact that computer science itself changes, also what is being taught in schools and which hot topics are pushed within the computer science education community shifts. So we might see changes in the topics and in the computational thinking skills dimensions. However, the three-dimensionality of the proposal makes it easy to adapt to these changes. If, for instance, artificial intelligence, machine learning, or other related topics become so vital that they deserve their own topics category it can be added easily and former tasks could be re-assessed.

Acknowledgments. The author would like to thank the Bebras community for continuously striving for higher quality and for better use of the resources. Without the many discussions and presentations this paper would not have been possible. This is especially true for the inventor of the Bebras challenge, Valentina Dagiène, who continually combines practical aspects with general research; she was also so kind to point out current papers on Bebras. The author would also like to thank the members of the Bebras community who participated in the short evaluation and feedback round. Finally the author would like to thank the anonymous reviewers for their constructive feedback.

References





1. K-12 computer science framework (2016). <https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>
2. Bebras International Challenge of Informatics and Computational Thinking: Documents (2021). <https://www.bebas.org/documents.html>
3. Bloom, B.S.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain. David McKay Company, New York (1956)
4. Blöchliger, I., et al.: Pre-workshop review process (2014, unpublished)
5. Calcagni, A., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A.: Promoting computational thinking skills: would you use this Bebras task? In: Dagiene, V., Hellas, A. (eds.) ISSEP 2017. LNCS, vol. 10696, pp. 102–113. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71483-7_9

6. Dagiene, V., Hromkovic, J., Lacher, R.: A two-dimensional classification model for the Bebras tasks on informatics based simultaneously on subfields and competencies. In: Kori, K., Laanpere, M. (eds.) ISSEP 2020. LNCS, vol. 12518, pp. 42–54. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63212-0_4
7. Dagiene, V., Sentance, S., Stupurienė, G.: Developing a two-dimensional categorization system for educational tasks in informatics. *Informatica* **28**(1), 23–44 (2017)
8. Datzko, C.: Informatik-Biber...und dann? (2016). <https://www.abz.inf.ethz.ch/wp-content/uploads/2016/09/STIU2016.WS.5.Informatik-Biber...-und-dann.pdf>
9. Datzko, C.: The genesis of a Bebras task. In: Pozdniakov, S.N., Dagiene, V. (eds.) ISSEP 2019. LNCS, vol. 11913, pp. 240–255. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33759-9_19
10. Datzko, C.: Kreativer Informatik-Unterricht auf der Basis der Informatik-Biber-Aufgaben (2020). <https://www.abz.inf.ethz.ch/wp-content/uploads/2020/02/Kreativer-Informatikunterricht-auf-Basis-der-Informatik-Biber-Aufgaben.pdf>
11. Datzko, C., Datzko, S.: Aspects of designing a successful bebras challenge (2021, to appear)
12. Datzko, S., Guggisberg, M.: Informatik-Biber-Aufgaben als Anregung für kreativen Informatik-Unterricht in der Unterstufe (2020). https://www.abz.inf.ethz.ch/wp-content/uploads/2020/03/Workshop-10_ueberarbeitet_pdf-Version.pdf
13. Denning, P.J., Tedre, M.: Computational Thinking. The MIT Press, Cambridge (2019)
14. Djambong, T., Freiman, V., Gauvin, S., Paquet, M., Chiasson, M.: Measurement of computational thinking in K-12 education: the need for innovative practices. In: Sampson, D., Ifenthaler, D., Spector, J.M., Isaías, P. (eds.) Digital Technologies: Sustainable Innovations for Improving Teaching and Learning, pp. 193–222. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73417-0_12
15. Gesellschaft für Informatik (GI): Grundsätze und Standards für die Informatik in der Schule - Bildungsstandards Informatik für die Sekundarstufe I. LOG IN 150/151 (2008)
16. Gesellschaft für Informatik (GI): Bildungsstandards Informatik für die Sekundarstufe II. LOG IN 183/184 (2016)
17. International Society for Technology in Education (ISTE), Computer Science Teachers Association (CSTA): Operational definition of Computational Thinking for K-12 education (2011). <http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf>
18. IT-Feuer: The mystery of Bebra (2021). <https://it-feuer.ch/event/the-myteries-of-bebra/>
19. Kultusministerkonferenz: Einheitliche Prüfungsanforderungen Informatik (2004). https://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1989/1989_12_01_EPA_Informatik.pdf
20. Palts, T.: A model for assessing computational thinking skills. Ph.D. thesis, University of Tartu (2021). <https://dspace.ut.ee/bitstream/handle/10062/71913/palts.tauno.pdf?sequence=1&isAllowed=y>
21. Román-González, M., Pérez-González, J.C., Jiménez-Fernández, C.: Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Comput. Hum. Behav.* **72**, 678–691 (2017)
22. Schweizerische Konferenz der kantonalen Erziehungsdirektoren (EDK): Rahmenlehrplan für die Maturitätsschulen: Informatik (2017). https://edudoc.ch/record/131917/files/rfp_inf_2017_d.pdf

23. Tang, X., Yin, Y., Lin, Q., Hadad, R., Zhai, X.: Assessing computational thinking: a systematic review of empirical studies. *Comput. Educ.* **148**, 103798 (2020)
24. UK Bebras: 2019 Bebras challenge (2021). https://www.bebas.uk/index.php?action=user_competitions
25. Wing, J.M.: Computational Thinking. *Commun. ACM* **49**(3), 33–35 (2006)
26. Wing, J.M.: Computational Thinking's influence on research and education for all. *Ital. J. Educ. Technol.* **25**(2), 7–14 (2017)



Girls' Summer School for Physical Computing: Methodology and Acceptance Issues

Gabrielė Stupurienė¹✉ , Anita Juškevičienė² , Tatjana Jevsikova² ,
Valentina Dagiene^{1,2} , and Asta Meškauskienė¹

¹ Institute of Educational Sciences, Vilnius University,
Universiteto 9, 01513 Vilnius, Lithuania
{gabriele.stupuriene, valentina.dagiene}@mif.vu.lt
asta.meskauskiene@fsf.vu.lt

² Institute of Data Science and Digital Technologies, Vilnius University,
Vilnius, Lithuania
{anita.juskeviciene, tatjana.jevsikova}@mif.vu.lt

Abstract. Physical computing, making, tinkering and Computational Thinking (CT) are frequently applied to promote Computer Science (CS) and attract the attention of young people. In recent years, STEM (Science, Technology, Engineering, and Mathematics) education has received considerable support from schools, parents, and communities. The combination of physical computing with STEM education can improve CT as part of CS or informatics/computing education and can be considered as an essential skill for the workforce of the 21st century. Physical computing helps children build connections between the real world and programming, while giving them something exciting to focus on. The goal of this study is to demonstrate how a summer school on physical computing and STEM education including CT contributes to shaping girls' attitudes and their acceptance of technology. To examine how the technology acceptance factors are related to the girls' behavioural intention and attitude change towards the use of microcontrollers, an adaptation of the Unified Theory of Acceptance and Use of Technology (UTAUT) was applied. The survey was conducted with 21 girls, 11 to 15 years old, both before and after summer school. The results showed that carefully planned activities on mini-projects with purposeful support of lecturers reduce the technological anxiety and contributes to the girls' intention to use technology in the future. The implications of these results can be used to develop various supporting physical computing activities in the school community. The methodology, examples and real stories of success of girls in computing help teachers to promote CS education in schools.

Keywords: Arduino microcontrollers · Computational thinking · Computer science education · Gender issues · GEM/GEMS · Hands-on activities · Physical computing · STEM · Technology acceptance

1 Introduction

Educators have been worried about the growing gender gap in Computer Science (CS)/informatics education in most countries [21,33]. J. Guggemos (2021) [6] showed that self-concentrated and self-determined motivation plays an important role in explaining gender differences. There is a broad consensus among researchers and educators that STEM provides a variety of benefits for education. STEM plays a vital role in many modern professions that need competitive and innovative professionals to meet the needs of the digital age. Targeted links between different disciplines are needed, where each discipline is taught comprehensively. The goal of STEM is not so much to teach science or mathematics, for example, but to apply the content of different disciplines in real, engaging situations, as applied knowledge leads to deeper learning [12].

S. Papert coined the term “computational thinking” and discussed that computers might enhance thinking and change patterns of knowledge accessibility [24]. Few decades later J. Wing (2006) [35] reformulated CT as thought processes and promoted these ideas in the design and analysis of problem-solving. Recently CT is promoted as one the most desirable skills [7] that should be taught in all schools, employing computational ideas integrated with other disciplines, especially in all STEM subjects. To invent new technologies in the future, learners need to understand fundamental principles beyond and practice using various tools. STEM and CT focus on breaking down complex problems and finding solutions, which can be applied to multiple areas of life, not just to school [14,16,23].

Educators and researchers widely agree that project implementation (hands-on activities) motivates and attracts learners [2,19]. Such projects and practical applications are often referred to the physical computing, making and tinkering paradigm, which brings CT concepts off the screen and into the real world so that the student can interact with them [18,25].

Making is a way of bringing engineering to young learners. Tinkering is a powerful form of learning by doing. Making and tinkering require (computational) thinking and include the design and realization of interactive objects and installations that encourage learners to use their imagination for tangible real-world development products. If learners failed to solve a problem on an abstract level, they often managed to use tangible objects because multiple representations of the same knowledge were used. Learners are motivated by their own creativity, which is achieved through the use of modern prototype boards that allow for the creation of meaningful projects [10].

Nite et al. (2020) [30] stated that the summer camp/school activities on physical computing (i.e., coding Arduino microcontrollers) provide an opportunity for school children to engage in hands-on learning in STEM in an informal classroom setting. Learners are engaged with their peers as they learn to create the circuits and then develop code to perform tasks, such as flashing light, buzzers, and digital musical pitches, that are used in many technological applications in their daily lives. Additionally, physical computing offers diverse connections to other STEM subjects, such as the simulation of behaviour (biology), the collection and analysis of measurements (physics), and logical operations (informatics, mathematics).

1.1 Gender Issues

Computing and technology are seen as a boys' subject, this is mentioned by girls to explain their lack of interest in information technology or IT [36]. These socially constructed gender profiles are presented to children at a very young age and is perpetuated through societal norms. It is difficult for a girl to identify and sustain a passion in technology if there are not many female role models in CS or STEM. This is why girls may have more potential for the cultivation of CT in STEM education. An empirical study by Sun et al. (2021) [29] stated that results showed that “learning attitude of girls from primary school towards STEM was generally more positive than that of boys in the same period, while for CT skills, although the gender difference was not significant, the score of girls was slightly higher than that of boys” (p. 355). In [27] was found out that the progressive abandoning of some STEM fields by girls begins after the age of 12, fuelled by the predisposition of girls to underestimate their ability to be successful in STEM fields. The findings of [5] suggest that one way to boost girls' interest and confidence in CT is to provide engaging computing experience via summer schools. GEMS is an acronym that is usually used for after school groups that support girls and their STEM-interests (GEMScub). While the G, M, and S refer to girls, math, and science, respectively, the E can refer to engaged, excelling, or engineering [17].

In 2020, Vilnius university (VU) joined the international project “GEM – Empower Girls to Embrace their Digital and Entrepreneurial Potential” [4], the aim of which is to provide opportunities to unleash the potential of girls' digital literacy and entrepreneurial skills in STEM activities. One of the activities was to organize training courses for teenage girls using physical computing during summer holidays. The first summer school was organized by VU on August 3-6, 2020. The main purpose was to empower 11–15 years old girls to embrace their knowledge and skills in physical computing and engineering, including digital and entrepreneurial potential, and pursue related studies and careers.

The summer school activities were based on the working principles: (1) to keep the theoretical prerequisites and inputs in the beginning as minimal as possible; (2) to empower girls to get the chance to learn the fundamentals of physical computing in combination with various STEM subjects and solve problems on their level while self-reliant, but mentored, research projects will be provided for more advanced learners; (3) partial failure is an important part of scientific work, understanding mistakes and taking advance of them, gaining a tolerance of frustration are necessary skills to successfully work in STEM and IT jobs, learning these requires care and professional experience by the educators; (4) discussions and reflection take an important part in the work process.

1.2 Technology Acceptance Issues

How do female teenagers accept technology, for example the Arduino microcontrollers, that might seem for them at first glance too technical and sophisticated to use? In order to predict technology acceptance by users, several models and

their modifications have been developed, among which most used are UTAUT (Unified Theory of Acceptance and Use of Technology) and TAM (Technology Acceptance Model) [11, 15, 34]. TAM version 2 (TAM2) was used in research [3] and questionnaire for 40 university students where authors evaluated the Arduino Uno board used in Human-Computer Interaction (HCI) classes. Authors strongly suggest that every HCI course should include practical activities related to tinkering with technology such as applying microcontroller boards, where students actively and constructively participate in teams for achieving learning objectives. However, we could not find any research studying girls' acceptance of Arduino microcontrollers.

For this reason, we decided to use the main constructs of the UTAUT model with additional personal factors related to self-efficacy and anxiety:

- Behavioral intention – Individual's tendency to perform some behaviour [34].
- Social influence – The degree to which an individual perceives that it is important how others believe he or she should use the new system [34].
- Effort expectancy – The degree of ease associated with the use of the system [34].
- Technological anxiety – Negative emotional response, describing an individual's perceived apprehension or discomfort related to using a technology [22].
- Technological self-efficacy – the belief in one's ability to successfully perform a technologically sophisticated new task [20]. Technological self-efficacy is reported to be the dominant determinant of the intention of using the technology [31, 32]. In our study, the technologically sophisticated task is learning while creating new physical computing (Arduino) projects to achieve the intended learning outcome.

1.3 The Aim of the Study

The aim of this research is to determine the effect of the use of Arduino microcontrollers during the girls' summer school. For this purpose, the research questions being considered here are:

- RQ1. Do physical computing activities with Arduino sets in mini-projects promote the development of STEM competences of 11–15 years old girls including CT?
- RQ2. How are technology acceptance factors (effort expectancy, social influence, technological self-efficacy, technological anxiety) related to the girls' behavioural intention to use technology (microcontrollers) and attitude to change towards physical computing?

The paper is structured as follows: Sect. 2 identifies the methodology of the proposed physical computing activities in the summer school, situating our own approach. Section 3 presents our research methodology and general contribution. The results of collected data analysis are presented in Sect. 4. Finally, we discuss the implication of these findings for CS educators, concluding with identified directions and recommendations.

2 Methodology for Physical Computing Education Activities

During four days at the GEM summer school in August 2020, sixteen mini projects were implemented. The main Arduino-set and sensor-based activities were aimed at exploring ecology and automation solutions for safety and comfortable life. The idea of the mini projects was implemented by pairs of girls. Completed mini projects were presented to all participants at the end of the school. Smart city and smart greenhouse proposal topics were presented as complex real life problems that cannot be solved by applying a simple algorithm (Appendix A). Additionally, a single correct answer usually does not exist, and that requires a learner to consider alternative ways of thinking to provide a reasoned argument for supporting the generated solution. Thus, it results in more motivation and effort for the learners in the development of the solution.

Sixteen different projects were implemented covering all proposed topics. During the summer school, some were slightly modified (in code, working principle or purpose, such as, remote boom barrier control was used for jewellery box control), and some were even combined (such as the CO₂ measure device and the Soil humidity measurement device).

Mini projects were designed with the aim to involve various technological areas such as electronics, engineering and science (Appendix B).

Design thinking learning approach [8,9,28] was adopted to solve complex problems by maintaining in-depth learning processes with problem perception and various solutions. Design thinking is currently being identified as a paradigm for dealing with problems in many areas especially in STEM and CS education. Activities started with an introduction followed by primary projects demonstrations. The design thinking model (Appendix C) was used for the implementation of activities for four-days.

The model involves five main activities or stages. The goal of the first phase (**Understand**) is to find the relation between the problem and its context in order to pinpoint the challenge. The second phase (**Synthesis**) aims to define the problem and its context. Particular problems can be interpreted from different perspectives, thus, in this phase all information is filtered into a clear idea. The **Ideate** phase comprises the generation of ideas by applying knowledge and collaboratively transforming it into actionable problem solving ideas. The fourth phase, **Prototype**, focuses on the action of tangible object creation in order to test it, in order to share the abstract idea in physical form. The **Testing** phase involves solution generation after the design process has transitioned into action. During the testing, the focus is on the solution, and it shows how well the problem has been understood. The last phase, **Iteration**, refers to the cyclical and iterative nature of the design thinking process: it is desired to repeat phases and move from any phase to another [13,28]. All the phases were implemented by using the microcontroller activities presented above.

Smart city and smart greenhouse elements were developed and programmed in order to reach the intended aims, such as ecological greenhouse solutions, house and street safety and comfortable or easy control. The girls were working

based on a presented design thinking model and involving engineering, electronics and science areas. First, the smart house concept and main components of the smart house system were presented. Then, the girls were introduced to Arduino kits, all the parts of those kits were presented as a main smart house system component: identifying objects (e.g. RFID), connected devices (e.g. sensors), control system (e.g. Arduino IDE) and controller (Arduino Uno).

Additionally, in order to make the girls to understand the purpose of each part of the kit, they were asked to divide each part into three main groups: parts for data input (e.g. sensors), parts for data output (e.g. LCD, leds, buzzer, servo motor) and parts for data processing (e.g. controller). After this activity, participants were asked to construct simple Arduino prototypes based on electrical schemes by adding wiring parts, such as wires, resistors, and breadboard. Next, the girls were asked to choose the projects they want to implement and to discuss in pairs which problem they want to solve.

After discussion, pairs prepared the visualisation of their projects – presenting the main parts of desired project implementation and the sequence of its operation: data input, action and output, including the kind of materials that they will use in order to implement prototypes. After discussion and final solutions, the girls construct the prototypes for their projects (Appendix D) with Arduino kits, based on electronics schemes (showed how to connect each intended prototype part) prepared by the lecturers.

Later, the successfully developed prototypes were implemented into final projects using 3D pens and carbon boxes, hot glue, felt-tip pens and paints. The last day of summer school was dedicated to a final project presentation, accompanied by posters. The pairs of girls were asked to demonstrate the mini projects operation and explain which STEM subject knowledge they needed in order to be able to implement the project, what kind of issues they had and how they were solved, how the developed device is working and where it can be used, and which materials were chosen for the implementation and why.

3 Research Methodology

3.1 Participants

Before starting physical computing activities, the girls were asked to take part in research and answer pre- and post- survey questions. Out of 32 girls, who participated in the summer school, 30 answered the questionnaire before and 23 answered after the end of the activities. Anonymised identifiers from pre- and post- survey matched for 21 girls, and in this research, we used only the response from those participants as the data set.

The age of the girls ranged from 11 to 15 years: the majority were 12–13 years old. The participants are students from the twelve Vilnius municipality schools. The girls were asked to answer a question about which microcontrollers they have previously used. None of them knew the CodyBug, and only one girl knew the MakeyMakey microcontroller. 10 out of 21 girls knew the Micro:Bit. This higher rate it is related to the national community initiative “Computers

for Kids” started in 2017 with the purpose of donating BBC Micro:bit devices to every 5th grade pupil in Lithuania.

Before the summer school, 3 girls had used Arduino. After the summer school, 12 more girls identified that they now know Arduino (in total 15). But the remaining 6 girls in the data set did not realize that they used Arduino during the summer school. One girl (age 12) answered that “*I haven't used any micro-controllers yet, only STEM*”. This is a critical point: she has not recognized the differences between microcontrollers and STEM. A second girl (age 12) before and after mentioned LEGO, the third girl (age 12) mentioned before and after CodeMonkey, two girls (both age 13) mentioned before and after the Comenius Logo (Logo software used in some schools) and Scratch as microcontrollers. All these 6 girls do not understand the meaning of microcontrollers as a technology, and their understanding has not changed after hands-on activities.

3.2 Instruments

Data for the girls' microcontrollers acceptance aspects were collected using several scales for variables, measured via five-level Likert type ranging from Strongly disagree = 1 to Strongly agree = 5:

- Scales adapted for acceptance of microcontrollers from the UTAUT scale [34]:
 - Effort expectancy (EE) 4-item scale,
 - Social influence (SI) 3-item scale,
 - Behavioural intention (BI) 3-item scale.
- Technological anxiety (TA) 4-item scale, adapted from [27];
- Technological self-efficacy (TSE) is measured using two items, adapted from [1] for microcontrollers and GEM case:
 - “*I am confident of using the microcontrollers even if there is no one around to show me how to do it.*”
 - “*I am confident of using the microcontrollers even if I have never used them before.*”

We also used the “Attitude change” variable “*My attitude toward microcontrollers has...*” (five-level Likert type ranging from Strongly deteriorated = 1 to Strongly improved = 5).

4 Research Methodology

We used construct scores obtained after completing the summer school activities. For technological anxiety, in addition, we used pre-test data. The scores of the constructs are computed as a sum of the ratings of the items in each construct. Table 1 reports the descriptive statistics for each construct and the range of its scores. Spearman's rank correlations have been used for ordinal data in order to find out statistical dependencies between construct scores. Cross-correlation results are presented in Table 2.

Table 1. Descriptive statistics for constructs measured, N = 21.

Construct	Construct score range	Min.	Max.	Mean	Std. deviation
EE	4–20	4	18	12.52	3.430
TSE	2–10	2	10	5.67	2.058
SI	3–15	6	15	9.67	2.288
TA (pre-test)	4–20	7	18	11.29	3,289
TA (post-test)	4–20	4	17	9.52	4.094
BI	3–15	3	15	11.19	3.010
Attitude Change	1–5	2	5	4.10	0.995

Table 2. Spearman’s rank correlations between construct scores, N = 21.

	Age	EE	SI	TSE	TA pre	TA post	Experience	BI
EE	0.044	–						
SI	0.111	0.492*	–					
TSE	–0.004	0.679**	0.342	–				
TA pre	–0.384	0.558**	–0.004	–0.373	–			
TA post	–0.406	–0.477*	–0.196	–0.304	0.660**	–		
Experience	0.175	0.209	–0.057	0.374	–0.475*	–0.351	–	
BI	–0.052	0.631**	0.559**	0.492*	–0.280	–0.462*	0.043	–
Attitude Change	–0.038	0.451*	0.529*	0.291	–0.102	–0.297	0.056	0.781**

Correlation is significant: * at the 0.05 level (2-tailed), ** at the 0.01 level (2-tailed).

The strongest positive relationship (when considering UTAUT model’s constructs) is observed between effort expectancy and intention to use microcontrollers, i.e., behavioural intention ($\rho = 0.631$, $p = 0.002$). Social influence also positively correlates with behavioural intention ($\rho = 0.559$, $p = 0.008$). The next strongest relationship is observed between technological self-efficacy and behavioural intention to use Arduino microcontrollers ($\rho = 0.492$, $p = 0.023$). A negative relationship exists between technological anxiety in post-test and behavioural intention ($\rho = -0.462$, $p = 0.035$). Effort expectancy as a strongest predictor of intention to use microcontrollers by the girls shows positive correlation with technological self-efficacy ($\rho = 0.679$, $p = 0.001$) and social influence ($\rho = 0.492$, $p = 0.024$), whereas negative rank correlation is found with technological anxiety in pre-test ($\rho = -0.558$, $p = 0.009$) as well as in post-test ($\rho = -0.477$, $p = 0.029$). Previous experience using microcontrollers correlates negatively with technological anxiety in pre-test ($\rho = -0.475$, $p = 0.029$). We also see that attitude change towards microcontrollers correlates positively with effort expectancy ($\rho = 0.451$, $p = 0.04$), social influence ($\rho = 0.529$, $p = 0.014$) and behavioural intention ($\rho = 0.781$, $p < 0.0001$).

As behavioural intention depends negatively on technological anxiety only in post-test, we explore the change in technological anxiety between pre-test and

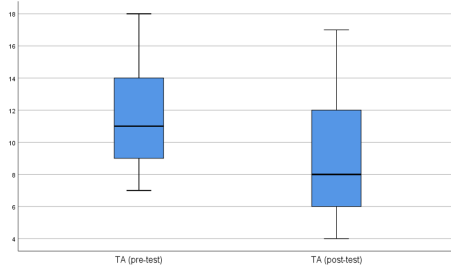


Fig. 1. Technological anxiety score in pre-test and post-test.

post-test (Fig. 1). A Wilcoxon signed-rank test ($Z = -2.466$, $p = 0.014$) confirms a significant (at 0.05 level) change (decrease) in technological anxiety to use microcontrollers after completing the summer school activities.

5 Conclusions

The purpose of this research work was to explore ways to promote Computer Science and Computational Thinking among girls by examining experiences, perspectives, and hands-on activities of 11–15-years old girls who participated in the summer school of Vilnius university. One of the surprising findings of this study was that even these girls who used microcontrollers for constructing objects had very limited knowledge of and experiences with CT or informatics. We predict that many other girls would also have very limited knowledge about CS and CT. Consequently, we highly encourage teachers at all school levels to provide CT education to all students. Otherwise, it will be very hard for students to be interested in, and eventually seek a career in, a CS or information technology field.

Our study results contribute to digital competence education by showing that a girls' summer school with physical computing contribute to girls' attitudes towards CS regarding effort expectancy, social influence, technological self-efficacy and technological anxiety. However, if the goal of a girls' summer school is to promote a more comprehensive picture of CS, further investigation is required. To ensure gender balance in the CS field and meet the job demand can be particularly done by recruiting girls into this profession at schools. Our study highlights the importance of providing engaging CS experiences in schools to retain girls' interest in CS. Findings from this study show that girls' lack of knowledge and experiences in physical computing and STEM resulted in girls' lack of participation in CS.

During the final mini projects presentations, the impression was that the girls were aware of which knowledge and skills they had and which they lacked. However, the results of the questionnaires showed that physical computing in combination with STEM is not so easy to learn. The majority of respondents indicated that activities did not change their opinion, and 24% of girls even

finished the summer school thinking that CS and STEM are not as easy subjects as they thought before. The successful final implementation of the mini projects gives us the chance to draw the conclusion that the majority of the girls nevertheless have embraced their knowledge and skills. In addition, we need to mention that a few of the younger girls have not yet acquainted with some science basics (such as physics or programming) and still coped successfully with their tasks.

In order to answer RQ2, we have explored the relationship of several technology acceptance factors that revealed the following. Girls' technological anxiety about using microcontrollers has significantly decreased after the completion of the summer school activities. Technological anxiety before the completed activities was, as expected, lower for girls who had previous experience in using microcontrollers. However, the remaining girls' high scores in technological anxiety to use Arduino microcontrollers after completion of the mini projects, negatively predict their intention to use microcontrollers in the future. Therefore, while preparing various training activities with physical computing, most attention should be paid to reducing technological anxiety amongst the girls for using microcontrollers which might include: 1) deeper understanding of controllers' functionality and 2) proposing learning strategies where errors, and their correction, are accepted as natural processes of learning.

A significant positive influence factor on girls' intention to use microcontrollers is effort expectancy, i.e., the degree of ease associated with the microcontrollers, which, in turn, correlates negatively with technological anxiety. Therefore, practicing, completing successful microcontrollers' projects and reducing technological anxiety also contributes to the girls' feeling of ease of use as an important predictor of intention to use microcontrollers in future. Technological self-efficacy, i.e., the belief of the student that she is able to complete the task with microcontrollers, is important for the intention to use microcontrollers. This gives one more direction for girls' STEM training activities. Higher levels of technological self-efficacy are also associated with lower levels of technological anxiety.

For the girls that participated in the summer school, the social support appears to be an important factor for using microcontrollers. Therefore, the support from important people in the girls' surrounding is necessary. This suggests to build a school community aimed at supporting physical computing activities. The examples of and real stories from successful women in computing are also very important for girls. Our study showed that the attitude change towards microcontrollers is more positive for girls with higher scores in effort expectancy, social influence and technological self-efficacy.

The main limitation of our study is the limited small sample. The dependencies observed should be tested in larger groups, and this is a future direction of our study, which also includes a deeper investigation of girls' attitudes of computational thinking activities.

Acknowledgement. The authors gratefully acknowledge the support of the European Commission for the funding of the project "Empower Girls to Embrace Their Digital

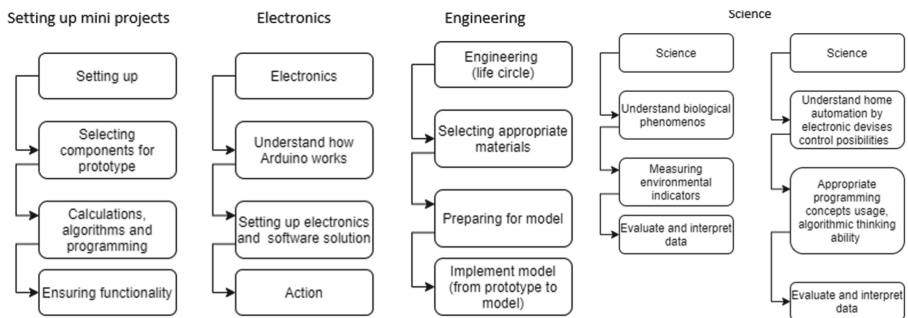
and Entrepreneurial Potential (GEM)” (LC-01380173). We thank Nicklas Anttu from Aalto University, Finland, for proofreading the paper and making suggestions.

Appendices

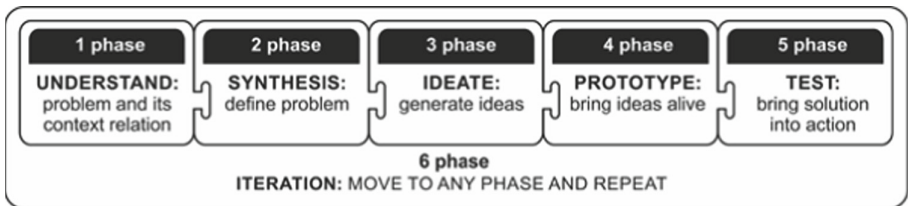
A The Proposed Topics of Arduino Mini Projects

Smart city mini-projects	Smart greenhouse mini-projects
<ul style="list-style-type: none"> • House/garage/street lighting (Arduino+LEDs+push button/remote controller and IR/RFID reader and card/WiFi) • Traffic lights (Arduino+LEDs) • Movement detection in the yard (Arduino+PIR sensor+LEDs/buzzer/LCD) • Alarm signal for non-authorized persons entering (Arduino+buzzer+LEDs) • Building entrance access only for authorized persons (Arduino+servo motor+RFID+LEDs+LCD) • Remote boom barrier control (Arduino+servo motor+LEDs+LCD) 	<ul style="list-style-type: none"> • Environment humidity and temperature measurement device (Arduino+DHT11 sensor+LCD) • Soil humidity measurement device (Arduino+soil humidity detection module+LCD) • Window control device (Arduino+Servo motor+DHT11/LED+Buzzer/WiFi module) • Lights control device (Arduino+LED+Photoresistor) • CO₂ measure device (Arduino+MQ-135 sensor+LCD)

B The Areas of the Mini Projects



C The Design Thinking Model [29]



D Girls Performing the Mini Projects



References

1. Chao, C.M.: Factors determining the behavioral intention to use mobile learning: an application and extension of the UTAUT model. *Front. Psychol.* **10**, 1652 (2019)
2. El-Abd, M.: A review of embedded systems education in the Arduino age: lessons learned and future directions. *Int. J. Eng. Ped.* **7**(2), 79–93 (2017)
3. Garcia-Ruiz, M.A., Santana-Mancilla, P.C., Gaytan-Lugo, L.S.: Integrating microcontroller-based projects in a human-computer interaction course. *Int. J. Comput. Inf. Eng.* **12**(10), 946–950 (2018)
4. GEM - Empower Girls to Embrace their Digital and Entrepreneurial Potential. <https://icse.eu/gem-empower-girls-to-embrace-their-digital-and-entrepreneurial-potential/>. Accessed 23 July 2021
5. González-Pérez, S., Mateos de Cabo, R., Sáinz, M.: Girls in STEM: is it a female role-model thing? *Front. Psychol.* **11**, 2204 (2020)
6. Guggemos, J.: On the predictors of computational thinking and its growth at the high-school level. *Comput. Educ.* **161**, 1–15 (2021)
7. Grover, S., Pea, R.: Computational thinking: a competency whose time has come. In: Sentance, S. (ed.) *Computer Science Education: Perspectives on Teaching and Learning*, Bloomsbury (2018)
8. Henriksen, D.: Creating STEAM with design thinking: beyond STEM and arts integration. *STEAM* **3**, 11 (2017)

9. Henriksen, D., Mehta, R., Mehta, S.: Design thinking gives STEAM to teaching: a framework that breaks disciplinary boundaries. In: Khine, M.S., Areepattamannil, S. (eds.) *STEAM Education*, pp. 57–78. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-04003-1_4
10. Jamieson, P., Herdtner, J.: More missing the boat-arduino, raspberry Pi, and small prototyping boards and engineering education needs them. In: *2015 IEEE Frontiers in Education Conference*, pp. 1–6. IEEE (2015)
11. Jevsikova, T., Stupurienė, G., Stumbrienė, D., Juškevičienė, A., Dagienė, V.: Acceptance of distance learning technologies by teachers: determining factors and emergency state influence. *Informatica* (2021). <https://doi.org/10.15388/21-INFOR459>
12. Jolly, A.: *STEM vs. STEAM: Do the Arts Belong?* - Education Week Teacher. Educ. Week (2014)
13. Juškevičienė, A.: STEAM teacher for a day: a case study of teachers' perspectives on computational thinking. *Inform. Educ.* **19**(1), 33–50 (2020)
14. Juškevičienė, A., Stupurienė, G., Jevsikova, T.: Computational thinking development through physical computing activities in STEAM education. *Comput. Appl. Eng. Educ.* **29**(1), 175–190 (2021)
15. Dwivedi, Y.K., Rana, N.P., Jeyaraj, A., Clement, M., Williams, M.D.: Re-examining the unified theory of acceptance and use of technology (UTAUT): towards a revised theoretical model. *Inf. Syst. Front.* **21**(3), 719–734 (2017). <https://doi.org/10.1007/s10796-017-9774-y>
16. Krueger, J.: *The Link between STEM and Computer Science*. <https://stratostar.com/the-link-between-stem-and-computer-science/>
17. Loyola, P. M.: *Understanding STEM Identity Construction: An ethnography of an all-girls STEM club*. College of Education Theses and Dissertations (2018)
18. Martinez, S., Stager, G.: *Invent to Learn: Making, Tinkering, and Engineering in the Classroom*. Constructing modern knowledge press, Torrance, CA (2013)
19. Martin-Ramos, P., Lopes, M.J., da Silva, M.M.L., Gomes, P.E., et al.: First exposure to Arduino through peer-coaching: impact on students' attitudes towards programming. *Comput. Hum. Behav.* **76**, 51–58 (2017)
20. McDonald, T., Siegall, M.: The effects of technological self-efficacy and job focus on job performance, attitudes, and withdrawal behaviors. *J. Psychol.* **126**, 465–475 (1992)
21. Meelissen, M.R.M., Drent, M.: Gender differences in computer attitudes: does the school matter? *Comput. Hum. Behav.* **24**, 969–985 (2008)
22. Meuter, M.L., Ostrom, A.L., Bitner, M.J., Roundtree, R.: The influence of technology anxiety on consumer use and experiences with self-service technologies. *J. Bus. Res.* **56**(11), 899–906 (2003)
23. Palts, T., Pedaste, M.: A model for developing computational thinking skills. *Inform. Educ.* **19**(1), 113–128 (2020)
24. Papert, S.: *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York (1980)
25. 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*, pp. 265–274. Suksapattana Foundation (2016)
26. Saadé, R.G., Kira, D.: Computer anxiety in e-learning: the effect of computer self-efficacy. *J. Inf. Technol. Educ. Res.* **8**(1), 177–191 (2009)
27. Sáinz, M., Eccles, J.: Self-concept of computer and math ability: gender implications across time and within ICT studies. *J. Vocat. Behav.* **80**, 486–499 (2012)

28. Scheer, A., Noweski, C., Meinel, C.: Transforming constructivist learning into action: design thinking in education. *Des. Technol. Educ.* **17**(3), 8–19 (2012)
29. Sun, L., Hu, L., Yang, W., Zhou, D., Wang, X.: STEM learning attitude predicts computational thinking skills among primary school students. *J. Comput. Assist. Learn.* **37**(2), 346–358 (2021)
30. Nite, S.B., Bicer, A., Currens, K.C., Tejani, R.: Increasing STEM interest through coding with microcontrollers. In: 2020 IEEE Frontiers in Education Conference, pp. 1–7. IEEE (2020)
31. Teo, T.: Modelling technology acceptance in education: a study of pre-service teachers. *Comput. Educ.* **52**, 302–312 (2009)
32. Teo, T., van Schaik, P.: Understanding the intention to use technology by preservice teachers: an empirical test of competing theoretical models. *Int. J. Hum. Comput. Interact.* **28**, 178–188 (2012)
33. Varma, R., Hahn, H.: Gender and the pipeline metaphor in computing. *Eur. J. Eng. Educ.* **33**, 3–11 (2008)
34. Venkatesh, V., Morris, M.G., Davis, G.B., Davis, F.D.: User acceptance of information technology: toward a unified view. *MIS Q. Manag. Inf. Syst.* **27**(3), 425–478 (2003)
35. Wing, J.: Computational thinking. *Commun. ACM* **49**(3), 33–36 (2006)
36. Won Hur, J., Andrzejewski, C.E., Marghitu, D.: Girls and computer science: experiences, perceptions, and career aspirations. *Comput. Sci. Educ.* **27**(2), 100–120 (2017)



Towards a Compulsory Computing Curriculum at Primary and Lower-Secondary Schools: The Case of Czechia

Jiří Vaníček^(✉) 

University of South Bohemia, České Budějovice, Czech Republic
vanicek@pf.jcu.cz

Abstract. A new national curriculum in informatics for primary and lower secondary education has been in force in the Czech Republic since September 2021. This curriculum introduces a compulsory subject, which systematically focuses on computing from the age of 9. This paper shows the path we have taken, the visions and principles on which the reform has been built, and the milestones that have lined the way from a user-led approach to computers to development of computational thinking and understanding the world of computers as a major educational objective.

This paper is structured as follows. We start with an introduction to the Czech educational context and some terminology. In Sect. 2, we describe the initial situation and the first steps towards the reform of school informatics. In Sect. 3, we explain the main concept and describe a project preparing new educational content and the conception of teacher education. An outline of the contents of the curriculum and teachers' first reactions are shown in Sect. 4.

Keywords: Education · Informatics · Computing · Innovation · National curricula · Country report

1 Introduction, Terminology

The paper focuses on the practical implementation of the paradigm shift in computer education in schools from a user-oriented approach to the mandatory inclusion of informatics into the curriculum. The process is shown on the specific situation of the Czech Republic, a Central European country with several hundred years of common history of the school system with other countries of the former Austrian Empire, and several decades in the 20th century under the influence of communist totalitarian ideas and experiments and limited sovereignty. It is a country that has been trying to return to full democracy and individual freedom for the last 30 years. In the paper we describe the initial situation, the ideas that appealed to us, the sources of inspiration and the changes in the system of informatics education.

The author of the paper participated in the innovation of informatics education described in the paper both as a member of the expert group that developed the national curriculum for primary and lower-secondary schools and as the leader of the strategic

PRIM project, which created the conditions for implementation of the proposed changes in practice by, among other, developing teaching materials, innovating undergraduate teacher education and popularization of computing. He was thus present at a number of activities and decisions and can provide “first hand” information.

The paper describes changes in Czech elementary schools, i.e. on ISCED1 and ISCED2 levels. The levels of education in Czechia are organized as follows:

- Primary (ISCED1, 5 years, age 6–11)
- Lower-secondary (ISCED2, 4 years, age 11–15)
- Upper-secondary (ISCED3, 4 years, age 15–19)

In the paper, the following terms are used with the following meaning:

- information and communication technology (ICT) in the sense of user approach to (consumption of) digital technology; the area for development of digital literacy and use of computers as a learning tool. The paper also uses the acronym ICT for the name of the educational area Information and communication technology, as this compulsory subject has been known until today.
- informatics – in the sense of computer science, computing; the basis of the discipline analogical to other scientific disciplines that are included in the STEM disciplines; authorial approach to technology, the field for understanding systems and for development of computational thinking. Informatics is also new name of the subject.

Our conception is close to the terminology used by Sysło [1, p. 144], for whom informatics deals mainly with creating ‘new products’ related to computers (such as hardware, programmes, software ...) and ICT mainly uses ‘informatics (computer related) products’. Similarly to Blaho [2] we perceive the area of ICT as the forerunner of informatics, which builds on ICT skills.

2 The Starting Point

2.1 Situation in Informatics Since 2006

Although ICT and informatics were taught in Czech secondary and primary schools since the early 1990s (it was perceived as “teaching computers”), these were selective subjects or topics included in other educational areas such as mathematics or technology. When Czechia joined the EU in 2004, a new Education Act was adopted, which formally brought Czech education closer to European practices, for example in defining the expected outcomes in the form of competencies or in defining educational areas [3]. A brand-new element in the system was the creation of School Education Programmes, which were binding for the school and had to include the mandatory core defined by the state Framework Education Programme (FEP). The development of School Education Programme was handed over to the schools.

When this Act came in force in 2006, the compulsory educational area of ICT was introduced for the first time at primary and lower secondary levels, where the main topics were searching, processing and use of information and communication [4]. The gradual

introduction of this subject resulted in the situation when, since 2011, the subject of ICT was taught on a compulsory basis in all schools in the extent of 1 lesson per week in one grade of primary and 1 lesson per week in one grade of lower-secondary level. The basis of teaching this subject was the use of office software. The subject did not include any informatics topic, its focus corresponded to the UNESCO model of ICT development in the curriculum of 2002 [5]. A certain autonomy of schools in developing the curriculum made it possible to include extra ICT lessons from the so called disposable lessons, which was eventually the case of almost half of the schools. Schools could also include the basics of computer science in their school education programme, most often in the form of a programming course in the so-called children's programming languages such as SuperLogo or the original Czech iconic language Baltie. However, schools more often included additional ICT topics in these extra lessons, such as digital photo and video editing. The support provided by textbooks and teaching materials, teacher training in the field and investment in school hardware has been weak and unsystematic until today.

We must point out here that the model did not impose any obligation to use digital technology in other school subjects. Thus, computers were often used only in the subject ICT, which resulted in its isolation. This model did not change until 2021.

2.2 First Occurrence of Computing: Bebras Challenge

The first opportunities for ordinary Czech schools to get in touch with informatics content other than programming were informatics tasks from the Bebras challenge [6] contest, in which Czechia has been participating since 2008 and which is designed for pupils aged 9 to 19. The fact that Bebras contest was first full-service electronically implemented competition at Czech schools with hitherto unseen tasks including interactive ones allowing participants to solve them by clicking on or dragging objects on the screen contributed to its popularity. The interest of schools and the number of participants gradually grew to 90,000 participants and the involvement of 16 % of all schools in 2019.

Despite the initial misunderstanding of some teachers to whom the tasks looked mathematical or logical as they were not about computers [7, p. 22], Bebras tasks were in fact the first mass introduction to informatics tasks, moreover embedded in real-word problem situations. The tasks from the contest were soon used by some schools in lessons. Teachers were introduced to the tasks in the professional teacher publications in the form of a series [8]. Some of the tasks are included in informatics textbooks [9]. The Bebras tasks were also used for new teachers preparation, they changed a point of view to informatics which was perceived as a discipline about programming and computers.

2.3 INICT Panel. Models of Inspiration

The INICT panel, an expert group of about 15 people, informatics educators from universities, ICT teachers and headmasters of secondary and primary schools, representatives of the school inspectorate and other education experts has been active for more than 10 years at the National Pedagogical Institute. Its goal is innovation in the ICT curriculum. This group was aware of the insufficient number of lessons, the isolation of lessons with computers from other subjects and the absence of computing content. Being aware of the fact that, unlike in the case of other subjects, upper secondary schools do not

show informatics and computer science as an interesting field of study at university led to the belief that computing should become an integral part of general education. It is this group that was at the background of the gradually created concept of innovation. Later, it created the new National Curriculum for Computing.

When developing ideas on how to innovate teaching in this area in Czechia, we naturally looked for inspiration in the world. We studied the situation in Slovakia, a country that has more than 300 years in common with Czechia and a virtually identical system of education. In 2008, Slovakia introduced computing as a compulsory subject from the 2nd grade of primary school with a much larger number of lessons (1 lesson per week in each grade starting in the third grade), focusing on areas with algorithmic thinking, procedures, problem solving, principles of digital technologies [10]. Thanks to the linguistic closeness and personal relationships, we could be present in the process of creation of the new school subject in a country that was 13 years ahead of us. Slovakia showed us the way not only in the form of a model national curriculum, but especially in the form of the content of teaching, textbooks and pedagogical research as well as by creating a community of teachers and educators around the conference DidInfo, which has been held alternately in Czechia and Slovakia since 2017 [11].

An important source inspiration for how to organize the teaching of computing was the model implemented in the United Kingdom, which was started by publishing the study Shut down or restart? [12] and led to the creation of the subject Computing. This document states that “ICT and informatics, which is a scientific discipline similar to mathematics and physics, are two different subjects in school education with different mission and functions, although they have common areas of synergy” [12, p. 10]. Instead of the term ICT, the term digital literacy was introduced.

2.4 Strategy of Digital Education

In 2014, after expert discussions of computer specialists, educators and teachers, the public, schools, employers and IT companies, the Czech government adopted the strategic document *Strategy for Digital Education until 2020* [13]. This document was binding primarily for the Ministry of Education to bring about innovation in education in this area.

Three main goals were the development of computational thinking, the development of digital literacy and the opening of education to new methods and ways of learning through digital technologies. Computational thinking is mentioned as a relatively new phenomenon and its introduction is justified not only by the need for new IT professionals, but also to enable students to acquire skills related to solving a wide range of problems associated with, for example, automation. The development of digital literacy and computational thinking of pupils and teachers was among the seven main directions of intervention.

This document triggered work that in the following six years prepared the transition to a completely new concept of school informatics and computer-assisted education, at the beginning of which we are now. The nature and extent of these preparations will be described below.

2.5 The Monograph Subject Didactics

In 2015, the first significant scientific monograph focusing on subject didactics entitled *Subject Didactics: Development – Status – Perspectives* [14] was published in Czechia. Representatives of 15 subject didactics describe the state, direction, resources, successes and problems of subject didactics in their field. Representatives of didactics of informatics were also invited to contribute to this monograph. Their chapter was named *Didactics of Informatics at the Starting Line*. Didactics of informatics in Czechia is characterized as emerging, not fully defined and established, with a number of problems of which the unsettled state of the maternal discipline, weak empirical research, narrow base and weak position of didactics at professional departments educating teachers are stressed out. The monograph defines the fields of informatics, ICT and technology in education. The term computational thinking is introduced. The chapter also describes the situation in Czech schools with problems of a low number of lessons, of one-sided focus of the curriculum towards user approach to technology, of poor quality of teacher education without a nationwide concept of in-service education and of the fact that the goal of teaching ICT is conceived as mastering a tool without any attempt at mastery of a deeper conceptual level.

The monograph was published in Czech; the main ideas from the chapter on didactics of informatics from the point of view of the international audience were presented in the paper [15].

3 Conception and Preparation of Changes

3.1 Computational Thinking and Digital Literacy

The use of the existing terms ICT and informatics ceased to be satisfactory because these terms were perceived as semantically equivalent in the general and pedagogical public and also they paid too much attention on the content aspect of education. New terms that emphasize the goals of education and development of the individual began to be used: computational thinking and digital literacy. With the help of these concepts, a model of change was developed, which eventually led to a complete change in the compulsory subject, it turned from a digital literacy-oriented to computational thinking-oriented subject.

The Table 1 illustrates these changes. The upper row shows the situation until 2021, the bottom row shows the situation the coming changes should bring about. Second column represents the compulsory subject focusing on “teaching computers”. Third column represents individual school subjects. The text describe the areas of educational objectives for whose fulfilment the different subjects will be responsible. extracurricular often means organized education outside the school or rarely non-compulsory added part of school education.

3.2 PRIM and DigiGram Projects

The creation of conditions for innovation of education and development of computational thinking and digital literacy was the aim of two major strategic projects in the

Table 1. The educational content of individual subjects with respect to informatics content before and after the planned innovation

Years	Informatics (ICT)	Other subjects	Extracurricular
before 2021	digital literacy	---	computing
after 2021	computing, digital literacy		digital literacy

years 2017–2020. They were implemented by a consortium of all 9 Faculties of Education in Czechia. It is important for necessary changes in pre-service primary and lower secondary informatics teacher education. Both of these projects were funded by the EU.

Main goal of the PRIM project (Podpora rozvoje infromatického myšlení – Support of Development of Computational Thinking) was to prepare changes that will be made in the subject informatics. This project reacted to the increasing necessity both of IT expert and general education of population in the area of computing [16].

3.3 Development and Piloting of a New Set of Textbooks

The main goal of the PRIM project was the creation of a new set of textbooks of computing that would cover teaching at all school levels from ISCED0 to ISCED3. The following demands on the textbooks were formulated:

- They must be targeted at teachers who have never taught or studied computer science – they must, therefore, among other things, present a detailed methodology that will fully support the teacher in self-study.
- As it is not certain that all schools will have the funds needed to purchase robotic aids, the set of textbooks as a whole must lead to the achievement of all FEP outputs without the need to buy any equipment.
- They must teach computing in a modern way, i.e. without long explanations, definitions, memorization. The pupils is expected to work actively and to build their knowledge by discovering, experimenting, creating, discussing, solving problems, cooperating, working on projects.
- They must be piloted in schools by beginner teachers.
- They must be available free of charge. This has been fulfilled by uploading them on the website <https://imysleni.cz/ucebnice> with the possibility of a free download or access to online materials with a CC-BY-SA license [17].

The total of 14 textbooks of computing were created by teams of university educators and teachers. Their content is divided into three major thematic areas:

- Programming and algorithmizing (in Scratch and in Python on upper secondary level)
- Other topics from informatics (work with data, coding, modelling, information systems), including Computer science unplugged activities
- Robotics (using Lego WeDo on primary level, Lego Mindstorms or Micro: bit on lower secondary level and Arduino on upper-secondary level)

These three areas evolved naturally. Before this set of textbooks there were no textbooks for primary and lower secondary schools in Czechia that would teach computer science topics. While there was some experience with programming in primary education, and some earlier ICT textbooks contained programming passages, other computer science topics were not covered at all. Therefore, the area of algorithms and programming was singled out. It was the experience with writing programming textbooks that made it possible to develop new textbooks with a more modern vision, based on building concepts [18]. Programming is perceived here as a training ground for the development of components of computational thinking, such as algorithmizing, abstraction, decomposition, evaluation or generalization. The textbook for the 5th grade of primary school, which was taken over and localized from the outputs of the English project Scratchmaths [19], emphasized the connection between programming and the teaching of mathematical skills [20].

Unlike in the case of programming, it was difficult to find international models for those textbook sections focusing on work with data, modelling or information systems. For example the Swiss textbook *Lösungen finden* by Hromkovič and Lacher [21], which focuses on these topics, was published later. Our textbooks introduce teachers to and guide them through a brand new, in Czechia not previously taught topics [22].

The area of robotics is conceived as non-compulsory as it requires the purchase of hardware by schools. Textbooks, especially for older pupils, work with the previously acquired programming competences and can thus focus on designing a robot and solving problems by programming it [23].

The textbooks were piloted in the years 2018–2020 in three stages. The first stage of piloting was conducted by very experienced teachers simultaneously with the creation of the first version of the textbooks. The authors were present in the lessons and received immediate feedback. The second and the third stages took place all over the country under supervision of individual pedagogical faculties. In total, the textbooks were piloted by over 130 teachers who had never taught computing. The feedback was provided through software into which the teachers entered comments and suggestions for adjustments after each lesson. These comments were assessed by the authors of the textbook for their potential incorporation into the next version. The fourth version of the textbooks was the final one.

It must be stated here that use of these textbooks is not binding for teachers. Every teacher has the freedom and responsibility to choose the methods, environment and aids they want. The same applies to programming languages, which are not prescribed.

3.4 Preservice and In-Service Teacher Education

The PRIM project created a model of comprehensive training of all teachers of computing on lower secondary level and also of all primary school teachers. All faculties of education introduced compulsory subjects in which pre-service teachers of computing focus on didactics of programming and school robotics and pre-service primary teachers learn basics of programming and also study methodology of teaching this subject. For some faculties this was the first time when such courses were given. Introduction of these subjects means, among other, that all graduates of pre-service primary school teacher education will be able to teach computing.

As far as in-service teachers are concerned, the project prepared a nationwide system of informatics training in two variants for primary and lower secondary levels of elementary school. This training in the scope of 24 lessons of presence learning supplemented by e-learning is intended to acquaint teachers with informatics content in the new textbooks. Piloting showed that it is necessary to teach them the basics of programming and to equip them methodologically to be able to teach using the textbooks and teachers' methodological guides. The system of in-service training is organized in two stages: dozens of certified lecturers train teachers in their region using the national methodological network of school support. These lecturers have been trained centrally by the authors of the textbooks. The lecturers draw attention to key passages of the textbooks and show teachers how to implement their lessons. These trainings are currently underway.

4 New National Informatics Curriculum

In January 2021, the Ministry of Education published the new Framework Education Programme for Elementary Education [24] This document defines the national curriculum which includes the new conception of computing. Its approval and its putting in practice were motivated by two facts. Experience from the time of COVID pandemic and long period of distance online education, which Czech teachers were able to cope with, boosted their confidence in the area and improved their attitude to the computer as a teaching aid. The other cause was the preparedness for the shift to new conception of computing, i.e. the existence of the piloted textbooks, of in-service teacher training and availability of finances to purchase of robotic and other teaching aids. The new national curriculum is at this point in force for levels ISCED1 and 2. The approval of the innovated curriculum for general upper secondary education (so-called gymnasium, level ISCED3) is planned for the end of the year 2021, the finalization of it is in progress now. It is necessary to add that new national curriculum for all levels was prepared together by the same group of experts and is interconnected. At secondary vocational schools, this innovation process was interrupted and postponed to the general revision of the Czech national curricula planned for 2023.

We concentrate our description to levels ISCED1 and 2 because informatic content is completely new there. Topics of algorithmization and programming are already included in current edition of the national curricula for some types of upper-secondary schools including gymnasium. The education area of ICT has been renamed as Computing in the new national curriculum. The compulsory minimum number of lessons is 1 lesson a week in each of the 4th to 9th grades, which is 300 % of the previous time allocation.

The content of the subject has been changed completely. It is now divided into four areas (which we illustrate by selected expected outcomes when finishing lower secondary education in the 9th grade):

- Data, information and modelling

The pupil

- defines a problem and determines what information will be needed for its solution
- finds mistakes in other people's interpretation of data
- proposes and compares different ways of coding data
- models a situation using graphs and schemas, finds and corrects a mistake in a model.

- Algorithmizing and programming

The pupil

- having read individual steps finds the problem which is solved by a given algorithm
- breaks a problem into individually solvable parts
- adapts a given algorithm for other problems, proposes various algorithms for the solution of a problem
- creates a programme in a block-oriented programming language, tests it and corrects any errors in it
- uses cycles, branching, variables.

- Information systems

The pupil

- identifies elements of an information system and the relationships among them
- defines a problem and determines how they will use data records to solve it
- designs a table for recording data
- sets sorting and filtering of data in the table.

- Digital technology

The pupil

- describes how a computer works in terms of both hardware and operating system
- stores and manages their data in a suitable format
- selects the most appropriate way to connect digital devices to a computer network
- handles typical computer faults and error conditions
- can manage their activities to minimize the risk of data loss or misuse [24].

The subject Computing was characterized as a subject helping pupils understand the computer and the world around us from the informatic perspective and developing the pupil's computational thinking. In terms of the organization of teaching, it was emphasized, for example, that the pupil should actively construct their knowledge by discovering, discussing, solving problems, etc. Recommended are group activities. There is no emphasis on reproduction of knowledge and memorizing.

Since the new teaching standards were developed parallelly with the new textbooks, the authors of the textbooks were often in an uncomfortable situation, as they had to react

to changes in the developed curriculum while writing. They did not even know how many lessons a week in each grade the innovated subject would be allotted. In consequence, the authors did not manage to write new textbooks for topics perceived as traditional, which had previously been taught, such as data processing using spreadsheets or digital technology.

The other innovated area, digital literacy, has become a so-called key competence in state documents, which means it has been implemented in other educational areas. All other subjects are responsible for achieving this competence. Thus, there is no specific responsibility defined for achieving goals in digital literacy such as implementation of measurement and research in science, typewriting in Czech language, safety and interpersonal relationships on the Internet in social sciences, etc. On the other hand, it is obligatory for all subjects to use the computer as a teaching aid to fulfil their subject educational goals. Digital technologies can be expected to change teachers' working methods.

4.1 Introducing Innovation at Schools

It has been stipulated that from September 1st 2021, schools may start transition to new computing curriculum, with the proviso that from 2024 all grades must be taught according to this new national curriculum. In other words, schools have been given time within which the teaching can be innovated by gradually adding new topics to the original curriculum.

In order for schools to be able to develop their school curriculum, so-called model school educational programmes in computing were designed within the PRIM project. This was not without discussions, as the school curriculum is meant to be created in discussions among the school's teachers, not taken over from somewhere. In the end, the opinion prevailed that it would be too difficult for teachers to define the content of an essentially brand-new subject that few people understand. This allowed creation of four variants of model school programmes, from which schools can choose with respect to their situation and interest: whether the school wants to profile itself as a school supporting informatics, whether it has the needed technical equipment and staff, whether it is rather conservative in its approach to changes and whether it wants to develop its own curriculum and let itself be only inspired. The model school programmes were developed to make maximum use of the new set of textbooks [25].

The publication of the new national curriculum triggered a massive discussion in the professional community, which was taken by surprise by the changes in consequence to insufficient communication from the side of the Ministry of Education before the publication of the changes. The Ministry found itself under pressure from professional teachers' associations in those areas of education whose number of lessons was shortened at the expense of computing. On the other hand, some schools welcomed the changes as a confirmation of their long-term efforts to innovate teaching.

In addition to questions that schools were expected to ask in the discussions (what to teach, who should teach it, who will pay for the changes), ICT teachers asked where to cover the traditional topics such as use of computer, office applications or Internet search. Currently, these topics are not compulsory in computing and it is up to the school whether it will include them in some educational area. Many teachers perceive the teaching of

office applications as the core of informatics and are likely to perceive it in this way some time longer.

5 Perspectives

The first stage of the long journey to the emancipation of computing as a standard area of general education of the citizen of the 21st century seems to have been successfully completed in the Czech Republic. Now, however, the following stage is beginning, in which the prepared changes will be implemented. If this stage is to be successful, a change in the teachers' attitudes and also their willingness to learn are essential. For many of them the change means learning how to teach a completely new subject. Also, a change in the thinking of school headmasters is necessary. They will, for example, have to allow class teachers or practitioners to teach informatics at primary school level instead of specialists in informatics who teach older pupils.

It is also imperative to set a standard for teachers, to develop new textbooks, to use new forms of teacher training, including personal assistance and sharing experience, to engage in research that evaluates the changes and points out their weaknesses as well as looks for new ways to teach informatics. Only a proper completion of innovation with quality feedback, with constant interest of the general public and the support of the ministry and last but not least with a massive training of new informatics teachers at universities will lead to the perception of informatics as a common school subject whose existence springs not only from the current situation but also because it can develop individuals in a modern way.

References

1. Sysło, M.M., Kwiatkowska, A.B.: Introducing a new computer science curriculum for all school levels in Poland. In: Brodnik, A., Vahrenhold, J. (eds.) *Informatics in Schools. Curricula, Competences, and Competitions*. ISSEP 2015. LNCS, vol. 9378, pp. 141–154. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_13
2. Blaho, A.: Informatika v štátnom vzdelávacom programe (Informatics in a state educational programme). In: Kalaš, I. (ed.): *DidInfo 2012*, pp. 7–14. UMB, Banská Bystrica (2012). http://www.didinfo.net/images/DidInfo/files/didinfo_2012.pdf. Accessed 2 June 2021
3. Ministry of education, youth and sports: *The Education Act*. Praha (2005). <https://www.msmt.cz/dokumenty-3/act-no-561-2004-collection-of-law-on-pre-school-basic>. Accessed 10 June 2021
4. NÚV: *Rámcový vzdelávací program pro základní vzdělávání – základní verze* (Frame educational programme for basic education – basic version). NÚV, Praha (2005). <http://www.nuv.cz/file/493/>. Accessed 10 June 2021
5. UNESCO: *Information and Communication Technology in Education—A Curriculum for Schools and Programme for Teacher Development*. UNESCO, Paris (2002)
6. Dagienė, V.: The bebras contest on informatics and computer literacy – students drive to science education. In: *Joint Open and Working IFIP Conference, ICT and Learning for the Net Generation* pp. 214–223. Kuala Lumpur (2008)
7. Vaníček, J.: Potenciální a skutečný dopad informatické soutěže do změn kurikula ICT v České republice (Potential and real impact of informatics contest to ICT curricula changes in Czechia). In: Kalaš, I. (ed.): *DidInfo 2012*, pp. 15–24. UMB, Banská Bystrica (2012). http://www.didinfo.net/images/DidInfo/files/didinfo_2012.pdf. Accessed 10 June 2021

8. Lessner, D., Vaníček, J.: Bobřík učí informatiku (Beaver teaches informatics), series of 7 parts. Matematika – fyzika – informatika (2013 – 2017). ISSN 1805-7705
9. Berki, J., Drábková, J.: Základy informatiky pro 1. stupeň ZŠ (Basic of informatics for primary school). Textbook. TUL, Liberec (2020). <https://imysleni.cz/ucebnice/zaklady-informatiky-pro-1-stupen-zs>. Accessed 10 June 2021
10. Blaho, A., Salanci, L.U.: Informatics in primary school: principles and experience. In: Kalaš, I., Mittermeir, R.T. (eds.) ISSEP 2011. LNCS, vol. 7013, pp. 129–142. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24722-4_12
11. DidInfo Homepage. UMB, B. Bystrica (2021). <http://www.didinfo.net/en>. Accessed 9 July 2021
12. Royal Society: Shut down or restart? The way forward for computing in UK schools (2012)
13. MŠMT: Strategie digitálního vzdělávání (Strategy of digital education). MŠMT, Praha (2014). <https://www.msmt.cz/uploads/DigiStrategie.pdf>. Accessed 10 June 2021
14. Stuchlíková, I., Janík, T., et al.: Oborové didaktiky: vývoj - stav – perspektivy (Field didactics: development – state – perspectives). Munipress, Brno (2015). ISBN 80-210-7769-0
15. Černochová, M., Vaníček, J.: Informatics education: current state and perspectives of development within the system of field didactics in the Czech Republic. ICTE J. **4**(3), 14–31 (2015). ISSN 1805-3726. <https://periodicals.osu.edu/ictejournal/dokumenty/2015-03/ictejournal-2015-3-article-2.pdf>. Accessed 10 June 2021
16. PRIM homepage. JU, České Budějovice (2017). <https://imysleni.cz>. Accessed 10 June 2021
17. PRIM: Učebnice a vzdělávací materiály pro školy (Textbooks and educational materials for schools). JU, České Budějovice (2020). <https://imysleni.cz/ucebnice>. Accessed 9 June 2021
18. Vaníček, J.: Early programming education based on concept building. Constr. Found. **14**(3) 360–372 (2018). <https://constructivist.info/14/3/360.vanicek>. Accessed 1 June 2021
19. Scratchmaths homepage. UCL, London (2015). <https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths>. Accessed 10 June 2021
20. Benton, L., Hoyles, C., Kalaš, I., Noss, R.: Building mathematical knowledge with programming: insights from the ScratchMaths project. In: Sipitakiat, A., Tutiyaphuengprasert, N. (eds.): Constructionism in action 2016. Conference proceedings (2016)
21. Hromkovič, J., Lacher R.: Einfach Informatik 5/6 – Lösungen finden. Textbook. Klett und Balmer, Baar (2018)
22. Lessner, D.: Attitudes towards computer science in secondary education: evaluation of an introductory course. In: Brodник, A., Tort, F. (eds.) ISSEP 2016. LNCS, vol. 9973, pp. 53–64. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46747-4_5
23. Jakeš, T., Bařko, J., Frank, F.: LEGO robotics textbook: solutions for creating constructions and manuals. In: INTED2020 Proceedings, pp. 4230–4236. IATED, Valencia (2020)
24. MŠMT: Opatření ministra školství, mládeže a tělovýchovy, kterým se mění Rámcový vzdělávací program pro základní vzdělávání (Frame educational programme for basic education). MŠMT, Praha (2021). https://www.msmt.cz/file/54860_1_1. Accessed 9 June 2021
25. PRIM: Modelové školní vzdělávací programy (Model school educational programmes). JU, České Budějovice (2021). <https://imysleni.cz/svp>. Accessed 10 June 2021

Teachers' Professional Development



Professional Development for In-Service Teachers of Programming: Evaluation of a University-Level Program

Majid Rouhani^(✉), Miriam Lillebo, Veronica Farshchian, and Monica Divitini

Department of Computer Science, NTNU, Trondheim, Norway
{majid.rouhani, monica.divitini}@ntnu.no

Abstract. Professional Development (PD) organizations provide training programs for computer science teachers through teacher PD. Programming as part of a teacher's PD has grown in importance in K-12 education. As a result, an increasing number of teachers, often without sufficient training, are teaching programming in their schools. It's challenging to learn to program, and it's even more challenging to teach it. Teachers must be able to program and teach programming, which is usually done in the context of multiple disciplines. In this study, we take a closer look at a teacher's perspective on teachers' PD in programming. We focus on the PD program offered by our university, composed of two courses over one academic year, and define the following research questions: Which strengths and weaknesses of the PD program provided are suited for in-service teachers when learning to program? Learning programming takes time. How could this long learning process be supported? This is an interview study of sixteen in-service teachers who joined the program. The main findings seem to be that teachers are comfortable with how the program is set up. They prefer flexible courses so that they can adapt the implementation with their regular teaching. They also emphasize the need for a continuous learning process and a community of practice (CoP) to continue developing. The main contribution of this study is the evaluation of the PD program at the university.

Keywords: In-service teachers · K-12 · Professional development · Programming

1 Introduction

Computer science (CS) in general and programming, in particular, have long been thought to be relevant to information technology (IT) experts. However, in recent years, this viewpoint has shifted. Software and technology are increasingly playing a role in practically every element of society and life at the current rate of digitalisation [1]. The teachers' PD within digital competence will therefore become more critical in the time ahead. Effective PD must prioritise teachers' and students' needs and address various elements at both the individual and situational levels [2]. Match PD to instructors' backgrounds, link it with the course's curriculum and employ effective motivational design

to promote teacher engagement are three recommendations made by Qian, Hambrusch's [3] study for building effective online PD for CS teachers. Learning programming is a difficult task, according to the scientific community [4]. Besides, teaching teachers may be more challenging and complex than other forms of adult education [5]. Therefore, it is essential to have a teacher perspective to map their needs and expectations on PD in programming.

In this study, we take a closer look at a teacher's perspective on their PD in programming and seek the answer to these research questions: (RQ.01) Which strengths and weaknesses of the PD program provided are suited for in-service teachers when learning to program? Learning programming takes time and might be challenging to learn and teach. (RQ.02) How could this learning process be supported? This is an interview study of sixteen in-service teachers who are preparing to teach programming at school. The sample includes teachers who attended a programming course at the university level. We used the thematic analysis method to analyse the qualitative data.

The following section presents the related work and positions the article in the context of the PD of in-service teachers in programming. Section 3 presents the case and research method. In Sect. 4, we present the results and discuss the main findings in Sect. 5. Conclusions and further work is presented in Sect. 6.

2 Related Work

There has been growing interest in incorporating programming into teacher education and PD in recent years [6]. This process, however, has not been easy; various studies have shown difficulties in teaching computer programming, and teachers sometimes do not know where to get proper training or how to use what they have learned in their classrooms [7]. At the same time, we know that learning programming is a time-consuming process, and in-service teachers usually do not have the required time. There are also challenges associated with teaching programming. Furthermore, the tools and technologies used when developing programs change over time and maintaining this knowledge is essential [8]. Teachers must prepare to integrate digital competencies into their teaching as computer science becomes more widely integrated into school curricula in a growing number of countries. This integration is a moving target, with new methods, tools, and applications appearing and disappearing at such a rapid pace that teachers must have the confidence to explore what is brand-new, relevant and plan their educational activities to include digital competencies independently and continuously [9]. PD is widely agreed to be critical for curriculum innovation in computer science teaching. However, in many countries, preparing PD for teachers is problematic due to instructors' lack of technological experience and abilities, pedagogical understanding, and topic knowledge of computer science [10].

Effective PD seems necessary, but what does it mean, and how do teachers view this from their point of view? The American Learning Policy Institute published a report in 2017 that outlines seven characteristics of effective teacher PD that lead to changes in teacher behaviour and improved student learning outcomes: focus on specific content, incorporation of active learning, support for collaboration, use of effective practice models, providing coaching and expert support, offering feedback and facilitating reflection,

and is of sustained duration. Effective professional learning incorporates most or all of these elements [11]. Several studies have found that PD is more successful when six features are included, defined as required or sufficient circumstances, e.g. [12, 13]. Despite minor changes in terminology and disagreements at the periphery, the core statements are highly consistent, as demonstrated by a recent meta-synthesis [14]. Sims and Fletcher-Wood [15] reexamines in their research the evidence that supports this consensus, suggesting that the reviews on which it is based have significant methodological weaknesses, such as using inadequate inclusion criteria and relying on an incorrect inference procedure. As a result, the consensus is likely to be incorrect. According to the argument, researchers might make more headway in identifying characteristics of good PD if they looked for alignment between evidence from basic research on human skill acquisition and aspects of rigorously evaluated PD treatments.

According to a survey of research published in the United States between 2010 and 2014, most computer science PD programs were less than a week-long and not stretched out over time, rather than taking the form of a one-time summer workshop [10]. However, there has been discussion in recent years about the need to move away from this type of traditional intensive training proposal to create spaces where teachers play an active role in learning, requiring studies that address the ‘teaching experience’ in these new training settings [16]. According to numerous research, the most productive context for informal workplace learning is a school culture that supports and values collaborative learning [17]. Cascade training, a “train the trainer” technique in which the first generation of trainers receives training and then delivers the specific content to the next generation of trainers, is a very complimentary paradigm in education. This procedure can be repeated indefinitely [18]. This study’s main contribution is to investigate the PD program provided by the university in programming and how they expect the long learning process to be supported from a teacher’s perspective.

3 Case Description and Research Design

The program is designed for in-service teachers in grades 8 through 13. (secondary school). With web-based lectures and weekly activities, the program is online, with no physical gatherings. Students enrol in these courses with the approval of their school and agree to provide teachers with free time to complete the course. The completion rate of the program was 87%. A nationwide program funded by the Ministry of Education partially covers the higher expenditures for schools¹.

One hundred ninety-two in-service teachers from different districts of the country participated in this PD program. More than 90% of the teachers in this study work in secondary schools, with most of the teaching in upper secondary schools (In the national educational system, this corresponds to the last three years of the K-12 system). STEM-related subjects are taught by more than 80% of the teachers. Others instruct in various areas, including language, history, music, arts and crafts, and so on. In their schools, 24% of participants have already taught programming. Before beginning the first course, 61% of participants had some programming experience. The participants

¹ <https://www.udir.no/kvalitet-og-kompetanse/>.

all are from the same country, although they represent various schools and districts. All schools follow the national curriculum. Participants have varying levels of teaching experience, ranging from new teachers to those who have been teaching for many years. The incentive to enrol in these academic courses varies as well. The majority think they are motivated, but others are less convinced that programming should be included in their classes. The curriculum comprises two courses totalling 15 ECTS credits. The first course (7.5 ECTS) covers the fundamentals of programming (variables, operators, if-and loop statements, lists, functions and libraries). The programming language we use is Python. The textbook “Starting out with Python” [19] is used. Student activities in the first course include three mandatory exercises, a reflection note and a mini-project (two weeks duration). Students can also complete specific assignments from the book. In the second course, we focus on how to teach and apply programming to interdisciplinary subjects. Student activities are four mandatory exercises, a feasibility study and a project. We have organised this course around a 6-week project aiming to define and evaluate activities to be used in teacher’s classrooms when teaching programming. As an outcome, teachers create a teaching plan in programming for their specific subject that they can use in their practice.

3.1 Research Design

The Overall Method. This study is based on sixteen interviews with in-service teachers in K-12. The study uses semi-structured interviews to explore the research questions by capturing teachers’ reflections on their PD program.

Interview Guide. The interview guide was constructed based on the following main elements: Part A (the PD program): (1) The professional development program at the university is divided into two programming subjects. In the first part, the focus is on learning to program, while in the second part, the focus is on teaching programming. What do you think about this form? (2) The second programming course is designed to be flexible, where students can set up their learning trajectories. What is your opinion on learning programming in a subject like this? (3) Do you think it would be necessary to have additional courses which build upon this program? Part B (PD in general): (1) What is the best way for in-service teachers to learn to program? (2) Have you ever been in a situation where you had to learn some programming concepts with your students? How has it affected your role as a teacher? (3) Learning programming takes time. How could this long learning process be supported? (Courses, seminars, CoP, school activities, peer learning, etc.)

Participants. The research team sent an invitation letter to all the participants of the 2020/2021 cohort. Sixteen teachers expressed interest in participating in the study: T1 (F, USS, Mathematics, physics, technology and research theory), T2 (F, USS, Mathematics and science), T3 (M, USS, Mathematics and science), T4 (F, LSS, Mathematics, social studies and Norwegian), T5 (F, USS, Physics), T6 (M, LSS, Arts and Crafts, Programming), T7 (M, USS, Chemistry, mathematics and science), T8 (M, LSS, Programming, mathematics, and science), T9 (M, LSS, Programming, mathematics, and science), T10 (F, USS, Mathematics), T11 (F, LSS, Programming, mathematics, and science), T12 (F,

USS, Mathematics, biology, and science), T13 (F, USS, vocational subjects in business support for ICT service, user support for ICT), T14 (F, LSS, Mathematics, science, technology and design), T15 (M, LSS, Programming, mathematics, and science), T16 (M, LSS, Norwegian, English, media and information, and programming). Interviews were conducted via Zoom, using the service offered internally by our university for GDPR-compliance. The interviews were recorded with a Zoom recorder and then transcribed by the interviewer. The relevant national agency approved the research. All the participants have been informed about the study, their rights and have been explicitly given their consent.

Data Preparation and Coding. We recorded interviews with teachers over zoom. Two of the researchers divided the recordings equally and proceeded to transcribe them individually. Some teachers requested approval of transcriptions before using them in this study. The two researchers coded a few transcriptions individually to create some initial codes and categories. The research team agreed on a joint codebook. Since we only use two coders, they used the method “Percent Agreement for Two Raters².” Approximately 20% of transcriptions were coded by two researchers independently, and the research team calculated the inter-rater reliability. When we achieved an IRR of about 80%, we encoded the rest of the transcripts.

4 Results

Some of the study’s findings are presented in this section. Due to length limits, we can only give a high-level overview of the most frequently used codes. Direct quotes from the reflection notes are translated from Norwegian by the writers.

4.1 Teachers’ PD in the Context of the Programming Courses at the University (Part A)

All interviewees have given their opinion on how the program has contributed to their PD in the area of computer science. We ask teachers about their views on how this form of PD fits and support their advancement in the field. In the following sections, we explain the most frequently codes discussed by participants.

Organization/Structure. Although there was both positive and negative feedback on the organisation and implementation of the courses, most are optimistic about the way the program is set up and performed. The positive aspects are that the program focuses on both learning programming and applying it in practice, flexibility concerning the choice of topics in the course and time for completion of the activities, and focus on programming didactics. The negative aspects are about the connection between lessons and exercises; exercises must be more relevant to teachers’ disciplinary knowledge; flexibility of the course is challenging for some and struggle to find the right balance of work; and desire for even more lectures/follow-ups. Fourteen teachers liked how the

² <https://www.statisticshowto.com/inter-rater-reliability/>.

program was organised or structured. Amongst these teachers, many noted that they appreciated how the program was structured throughout the year, with the fall semester having an introduction to programming and the spring semester focusing on teaching programming in school. E.g., “I think it has been excellent. It was very nice to get that run-up with learning basic programming this fall and then direct it towards teaching this spring. Especially now, considering that it is very concerned about how we should use it further in teaching. So, this spring semester has been beneficial considering that, so I think it’s helpful with the division that has been. It has been beneficial to learn the basics before moving on to thinking about using them for your teaching” (T10).

The Flexibility of the Course. The course, which runs in the spring, is flexibly arranged. This means that teachers essentially choose topics that are relevant to their subject areas. Throughout the semester, they work with subject matter and exercises that build upon each other, and teachers can use that to create a teaching plan for use in their teaching later. Fourteen teachers reflect on the flexible nature of the course. Teachers feel that this allows them to balance their coursework with their work and family life and focus on relevant topics to their subjects. E.g., “... And the spring has been quite flexible in a way, which was very good. It was a bit more to learn, but you got to use it. And that was perhaps the most important thing; not just learning the code, but also using it. Yes, I think there is a good distribution like that” (T13). “Do you mean that we can choose a lot ourselves what we want to do in the exercises? Yes, I think it is essential because we are in different subjects and have different grade levels. We have other focus areas, so I think it was very nice to imagine real situations in my classes” (T4). Few teachers also mention the negative impact of flexibility because it may be confusing and overwhelming. Some teachers may have little knowledge or understanding of programming to decide what is helpful for them to learn. An introduction to each part of the course could be beneficial to make awareness when deciding which areas they have to focus on throughout the course.

Teacher’s Needs and Interests. Although many (14) express that they have learned enough programming and can apply it in their subject areas, some (7) also point to more miniature courses to learn in-depth topics such as Arduino, micro: bit, raspberry pi etc. Some others suggest dividing the program into smaller parts to create classes with more homogeneous students regarding background knowledge and their teaching level. Others say that more courses are not needed now as they need time to absorb what they have learned so far. In addition, it is said that they have come far enough to build on this knowledge by themselves. Many of the teachers (8) were optimistic because they felt that the course had allowed them to learn relevant things for their classes. E.g., “I think it is essential because we are teaching different subjects, and we have different grade levels. We have other focus areas, so I think it was very nice to have the option of using relevant cases in the class” (T5). When it comes to time usage, teachers say that even though it is challenging to find enough time, more time should be allocated through school and privately, which means that one gets to develop more. Few teachers point to the technical infrastructure, and related challenges that might need a bit more attention than they currently get in the course. E.g., “Yes, we have been lucky, so we have bought everything because the principal is so fond of programming and technology. But we

experience technical issues, e.g., updating the firmware etc. It would have been nice if the course focused a little more on technical challenges as well...” (T14). There is a need to support teachers with technical issues, considering that their available infrastructure might vary.

Workload. Many teachers (10) talk about the amount of work in the course and seem to be divided in the middle. Those who think there is too much to do, have not allocated enough time. In addition, it is mentioned that teachers are not aware of how much work it is to learn to program. The others think that the workload is appropriate and emphasize that the flexibility (both in terms of content and times for submitting the exercises) helps to even out the load.

Final Assessment/Collaboration. The examination form in the course is project-based, where several people can collaborate. This seems to many teachers (8) to be positive. E.g., “I really liked the shape of the final assessment, that it is a project. I think it’s incredibly significant because there are a lot of people taking exams right now, and I see how stressed they are, and they memorise, memorise and memorise, and go to the exam, then the exam is finished, and they may forget everything shortly after the exam. But here, as I go through the exam, I learn at my leisurely pace, writing and noting every detail, and afterwards, I can use this project as a finished product. And if I want to repeat and come back, then I can come back, print it, and use it right away. I think this is fantastic with this project” (T12). Many teachers felt that collaboration during the course allowed them to use their strengths and support each other as a group. One teacher suggested that an additional activity in the study could be for teachers to share and present the teaching plans that they have created to their peers. Teachers also report on good collaboration between instructors/teaching assistants, and many felt a low threshold for asking questions and got responses quickly.

4.2 Teachers’ Perspectives on PD (Part B)

The second part of the interviews was related to teachers’ reflections on their future PD in programming after completing the university’s programming courses. By completing these courses, they have gained some experience and may have thoughts and opinions about how they envision the subject area’s development. We asked teachers what they think is the best way of PD in programming for in-service teachers. Learning programming may be time-consuming, and we asked them how they want this process to be supported. Furthermore, we were interested in knowing their view of the development in the subject area and the students since the learning process can extend over a more extended period.

Teachers’ Perspective on How to Learn to Program. All interviewees (16) expressed their opinion on how teachers should learn to program. We have identified 170 code references in transcriptions where teachers discuss different aspects such as the community of practice, collaboration and competency development through participation in courses, seminars, and self-initiatives. Thirteen teachers mention collaboration in the interviews and consider it to be an essential tool in learning programming. They emphasize the

importance of collaboration with fellow teachers to share skills and competencies and motivate each other. Thirteen teachers talk about courses or seminars as a meaningful way to learn and teach programming. Many teachers mention also practising and learning by doing as an essential part of the process. Programming is time-consuming, and teachers need to find the required time to do this. One teacher says that programming should be like a 'hobby': "Learning programming should become a hobby. One must prioritise and spend time to learn and understand the concepts well enough" (T9). Nine teachers expressed that a teacher professional environment may be an important arena for learning to program. Educating teachers could happen internally at the school through courses, or that the school would allocate time for teachers to learn together at school.

Time. When we talk about the PD of teachers in programming, we keep coming back to the time issue. It is crucial to allocate enough time to practice and improve programming skills. Fourteen teachers talk about "time" as a constraint when learning to program. E.g., "The fact that schools must give time to learn this, I think, is essential. Because it does not come by itself, and you cannot expect anyone to sit down to do this on Saturday nights because this is too big for it" (T10).

Willingness to Learn More. Sixteen teachers talk about their desire to learn more and cope with the pupil's motivation to learn to program. E.g., "I could well imagine such an additional in-depth course, but the question is what you are allowed to do as an employer. The employer wants to lift now with a minimum of programming skills. So, we get a 15 ECTS credits course now and then we are supposed to know to program" (T1). Some teachers seem to experience that their work colleagues are unwilling or unable to learn to program and report that this applies especially to older teachers who have started planning their retirement rather than learn programming.

Role as a Teacher. Some teachers emphasize the importance of active learning. Pupils need to be guided in the right directions and also learn themselves during this process. One teacher explained that she was anxious about not having as much knowledge about programming as necessary. At the same time, the teacher felt that there might be some advantages because the pupils might become more motivated by knowing that they are more knowledgeable on a subject than the teacher is. E.g., "I like knowing well what I am going to teach. So being a week in front of the students, it's not something I like. But I think it is also a considerable value because if the students discover that they know something that the teachers do not know, they become extra engaged in it and want to show it off and get even more motivated to learn even more. So I feel a little uncomfortable, but I know it may be excellent" (T10). Fourteen teachers talk about their experience when learning to program together with pupils, and those who have not experienced it but reflect on the possibility of it happening in the future. Twelve teachers described that they have been in situations where they have had to learn together with their pupils. E.g., "I think that I will come across that. Well, programming is relatively new for many pupils. Still, eventually, we will work with it throughout primary school. When they get to high school, they will probably have a completely different base of competence than what they have now, so I imagine that it will not be long before learning something with the pupils" (T10). Few teachers mention that they might have unrealistic

expectations about what they need to know and put too much pressure on themselves, probably lowering their confidence. E.g., “I think, in the textbook, the author says that when we learn to read and write, none of us expects us to be world-famous as Ibsen or Byron, or who at any time. We learn to read and write because we want to write a letter to Grandma. Why do we look at programming so that if we are to learn to program, then we must be like Steve Jobs? And it helped me lower my shoulders and the way the instructor explains here in this course. Very calm with a good pace, do not expect you to have fantastic prior knowledge, we begin” (T12). Another perspective that teachers (6) have talked about is the willingness of the school administration concerning making necessary time and resources available for teachers. Some schools are extra supportive, while others teachers feel they do not have enough back cover.

5 Discussions

Teachers have shared their experiences and highlighted some essential points from their perspectives on the PD in programming at the university. They seem to be comfortable with how the program is set up, i.e. the first part (7.5 ECTS) focuses on teaching them basic knowledge in programming. In comparison, the second part (7.5 ECTS) focuses on didactics, i.e. how to teach programming. Teachers emphasize the importance of *connecting programming to the relevance of their subject areas* from the outset, even if the focus is on fundamental concepts, such as using relevant examples and assignments from the subject areas in upper secondary schools. This can contribute to increased learning outcomes and more significant engagement.

Another aspect that teachers highlight is *the flexibility* of the program. Since time is a constraint, teachers are concerned with the shortest path to the goal, i.e. learning only what is relevant to their situation in the first place to get started with teaching programming in a short time. This offers a flexible solution where participants can set up their learning paths. Several dimensions can make the program’s flexibility valuable: On the personal level, participants vary in terms of their prior knowledge level, attitudes, and motivation; on a pedagogical level, they teach different subjects, school level and focus areas; and in a school context, the support by leadership and technical infrastructure may vary. There is a need to support teachers with *technical issues*, considering that their available infrastructure might vary. This problem could be addressed with (hands-on) tutorials about the more common technologies and troubleshooting sessions.

Teachers also mention the *project-based learning approach and the bridge model* [20] used in this PD program. They develop a teaching plan that they can use directly in their teaching after completing this program. This may increase their self-efficacy in teaching programming. Regarding teachers’ PD in programming after completing the university’s programming courses, we asked them to reflect on what they think is the best way of PD in programming for in-service teachers. What seems to occupy teachers the most is *establishing a community of practice, collaboration and development through participation in courses, seminars, and self-initiatives*. They emphasize the importance of collaboration with fellow teachers to share skills and competencies and motivate each other. Continuous development through collaboration may be an essential factor that may

be facilitated at several levels: between teachers, teachers and pupils, school & teachers internally and externally. Even for teachers who have attended an extensive program, most of them still feel the need to continue their learning process with additional courses, short term workshops, a CoP, collaboration at school etc. It is therefore essential to see any PD as a step in a life-long learning process.

Teachers are concerned with several challenges when learning to program, and the most important one is the *lack of time* that may have several negative aspects. Learning to program can be time-consuming, and in-service teachers need to develop their programming knowledge in parallel with other duties, making it even more challenging. The lack of time can lead to the whole PD becoming complicated and may demotivate the individual. Therefore, teachers must receive the necessary *support from the schools* (both in terms of time and financial support for purchasing equipment, etc.). Good PD is needed to achieve the right level of competence, and that this requires adequate support from the school. Teachers getting the time to attend courses at the university level is beyond what a single school can generally manage. This implies that there is a need for a higher-level commitment. In our case, schools get public support for it. We think it is important to underline that proper qualification of teachers cannot be achieved without a commitment that does not put all the weight on the shoulder and goodwill of individual teachers and schools. Another point is the *expectations of schools* for these types of PD programs. Many schools might think that PD in this area is something to do once due to changes in the national curriculum, rather than as a continuous effort that requires time, dedication, and a clear strategy.

Another practical issue is that PDs that follow the school calendar can create problems for teachers as they already follow their pupils through the same calendar. Thus, the exam may coincide with the exam that the students should have, which complicates the situation for teachers. This reinforces the need for a flexible PD program, but at the same time, we see that more teachers express that there will be challenges in finding out what to do as the scope becomes overwhelming. Instructors must therefore follow them closely. The *role of the teacher* may not be the same anymore. They point to some area that requires changes in the way they think and teach. Teachers might have *unrealistic expectations* about what they need to learn and put too much pressure on themselves, probably lowering their confidence. There might be a need to scaffold teachers' reflection on the competence level that they want/need to achieve. Another point that several teachers emphasize is that active learning should be an essential part of *the teaching process*. Pupils need to get involved, and teachers do not have to worry about not knowing enough about the topic. They will, to a greater extent, learn and develop in the subject together with the students.

6 Conclusion

This paper presented the results from the analysis of sixteen interviews of in-service teachers attending a PD program about learning and teaching programming. Our main objective was to evaluate the PD program offered by the university and asked the research questions: Which strengths and weaknesses of the PD program provided are suited for in-service teachers when learning to program? Learning programming takes time;

how could this long learning process be supported? The main findings seem to be that teachers are comfortable with how the program is set up. They also emphasize the importance of connecting programming to the relevance of their subject areas from the beginning, establishing a community of practice, collaboration and development through participation in courses, seminars, and self-initiatives. They highlight the importance of collaboration with fellow teachers to share skills and competencies and motivate each other. Also, the follow-up initiatives and the need for schools to create supportive environments for cooperative efforts and CoP are essential factors. The role of the teacher may not be the same anymore. They point to some area that requires changes in the way they think and teach. Teachers might have unrealistic expectations about what they need to learn and put too much pressure on themselves, probably lowering their confidence.

The main contribution of this study is the evaluation of the PD program provided by the university in programming. Results may be applicable in a PD program with a similar context but not be easily generalised. All participants are from the same country but teach on different school levels and subjects. The study focuses on teachers who have been given by their school the time to improve their teaching. Further, teachers' perspectives on PD in programming may be seen in the context of the university course already being provided. The findings may be affected by the highly motivated students who participated in this study and were pleased with the course.

As part of future study, it might be interesting to study teachers' perceptions from different institutions in future studies—both teachers who do not participate in any particular PD program and those who do.

Acknowledgments. We thank the teachers who participated in the interviews. The work is partly funded by the Norwegian Directorate for Education and Training (<https://www.udir.no>) and the Norwegian Center for Excellent IT Education (<https://www.ntnu.edu/excited>).

References

1. Nouri, J., et al.: Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Educ. Inq.* **11**(1), 1–17 (2020)
2. Galaczi, E., et al.: *The Cambridge English Approach to Teacher Professional Development* (2017)
3. Qian, Y., et al.: Who needs what: recommendations for designing effective online professional development for computer science teachers. *J. Res. Technol. Educ.* **50**(2), 164–181 (2018)
4. Ring, B.A., Giordan, J., Ransbottom, J.S.: Problem solving through programming: motivating the non-programmer. *J. Comput. Sci. Coll.* **23**(3), 61–67 (2008)
5. Mozelius, P., Humble, N.: Lessons learnt from teacher professional development in programming. In: *INTED 2020. IATED Academy* (2020)
6. Monjelat, N., Lantz-Andersson, A.: Teachers' narrative of learning to program in a professional development effort and the relation to the rhetoric of computational thinking. *Educ. Inf. Technol.* **25**(3), 2175–2200 (2020). <https://doi.org/10.1007/s10639-019-10048-8>
7. Hiltunen, T.: *Learning and Teaching Programming Skills in Finnish Primary Schools-The Potential of Games*. University of Oulu, Oulu (2016)
8. Baxter, D., Simon, B.: Clash of the timelines: lessons learned from the front lines of CS education. In: *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment* (2014)

9. Nordén, L.-Å., Mannila, L., Pears, A.: Development of a self-efficacy scale for digital competences in schools. In: 2017 IEEE Frontiers in Education Conference (FIE). IEEE (2017)
10. Menekse, M.: Computer science teacher professional development in the United States: a review of studies published between 2004 and 2014. *Comput. Sci. Educ.* **25**(4), 325–350 (2015)
11. Darling-Hammond, L., Hyster, M.E., Gardner, M.: *Effective teacher professional development*. Learning Policy Institute (2017)
12. Cordingley, P., et al.: *Developing great teaching: lessons from the international reviews into effective professional development* (2015)
13. Desimone, L.M.: Improving impact studies of teachers' professional development: toward better conceptualizations and measures. *Educ. Res.* **38**(3), 181–199 (2009)
14. Dunst, C.J., Bruder, M.B., Hamby, D.W.: Metasynthesis of in-service professional development research: features associated with positive educator and student outcomes. *Educ. Res. Rev.* **10**(12), 1731–1744 (2015)
15. Sims, S., Fletcher-Wood, H.: Identifying the characteristics of effective teacher professional development: a critical review. *Sch. Eff. Sch. Improv.* **32**(1), 47–63 (2021)
16. Reding, T.E., Dorn, B.: Understanding the “teacher experience” in primary and secondary CS professional development. In: *Proceedings of the 2017 ACM Conference on International Computing Education Research* (2017)
17. Webb, M., et al.: Computer science in K-12 school curricula of the 21st century: why, what and when? *Educ. Inf. Technol.* **22**(2), 445–468 (2016). <https://doi.org/10.1007/s10639-016-9493-x>
18. Karalis, T.: Cascade approach to training: theoretical issues and practical applications in non-formal education. *J. Educ. Soc. Policy* **3**(2), 104–108 (2016)
19. Gaddis, T., Agarwal, R.: *Starting Out with Python*. Pearson, London (2015)
20. Rouhani, M., Divitini, M., Olsø, A.: Project-based learning and training of in-service teachers in programming: projects as a bridge between training and practice. In: *2021 IEEE Global Engineering Education Conference (EDUCON)*. IEEE (2021)



Encouraging Task Creation Among Programming Teachers in Primary Schools

Jacqueline Staub^(✉), Zaheer Chothia, Larissa Schrempp, and Pascal Wacker

Department of Computer Science, ETH Zürich, Universitätsstrasse 6,
8092 Zürich, Switzerland

{jacqueline.staub, zaheer.chothia}@inf.ethz.ch,
{larissas, pwacker}@student.ethz.ch

Abstract. Programming is being widely adopted as a classroom activity to promote computational literacy across the full spectrum of ages. As of now, however, there is a gap between curriculum designers and the teachers that work directly alongside pupils. Educators build their lessons around predefined curricula and programming environments with limited scope for customization. As a result, their involvement is limited to using teaching resources as black boxes and creating tasks that live external to the programming environment. This work presents a small extension to the XLogoOnline programming environment and demonstrates how non-technical users are empowered to define, share and evaluate their own programming tasks. Our proposed tool is targeted at navigation tasks on a two-dimensional grid. Different categories of tasks can be easily assembled in graphical form and submitted solutions are automatically verified. We report from practical experience over a time span of 18 months and give highlights from a collection of 1331 programming tasks. The tool offers value by allowing teachers to design handouts and exams and also encourages teamwork by allowing pupils to challenge their fellow classmates. Beyond their use in the classroom, the idea of collecting task sets is a useful foundation for self-guided learning, exams and even large-scale competitions – which we intend to pursue in future work.

Keywords: Programming education · Turtle graphics · Creating teaching materials · Automatic validation of submissions

1 Programming Has Entered Compulsory Schooling

Computer science is currently being established as a new school subject in many countries around the world, making programming an activity that hundreds of thousands of children and adolescents are now learning as part of their compulsory education. The details of how and when programming is first taught varies from country to country [17]. To take one example, the UK’s National Curriculum contains a dedicated subject termed “*Computing*” that comprises activities such as programming that have deep-rooted origins in computer science. British

children are exposed to those concepts from the very beginning of their schooling career [1]. In contrast, the Swiss Lehrplan 21, which is being implemented by all 21 German-speaking cantons in the country, does not contain a dedicated subject for these activities prior to fifth grade. The curriculum does, however, require that teachers foster algorithmic thinking in an interdisciplinary way from kindergarten on.

Teacher Participation is Key to Enacting Change

The success of introducing a new teaching reform stands and falls with the participation of teachers. Only when teachers actively help and support the ideas of a reform can new ideas take root in the school system. Historically, it is well-known that the implementation of a reform depends largely on whether and how teachers take responsibility for the implementation of new ideas [8].

At the same time, however, there is evidence that teachers often have a sketchy understanding of content when developing their own learning materials and that targeted support is quite useful [5]. With regard to the current introduction of computer science as a new school subject, this is hardly different – teachers are the pillars of this reform and, without their active assistance, meaningful change is not possible.

We argue that most current programming environments do not provide enough opportunity for teachers to participate in the design of teaching materials and teachers are thus forced into the unhelpful role as users of prefabricated materials. Without the ability to respond appropriately to individual circumstances, to generate curricular content themselves, and to integrate it directly into students' learning environments, there is a risk that teachers may not want to support the introduction of a new school subject.

Merits of Integrating Learning Environments with Curriculum

Many different programming languages and environments are used in schools [17]; some of which follow a *free-form* approach while others are *task-based*. These two categories can be distinguished as follows:

- **Free-form environments:** We consider a programming environment to be free-form if it is set up as generalist platform – i.e. it allows for numerous applications while not providing any concrete embedded programming challenges. This category includes several well-known environments such as Scratch and ScratchJr [12], AppInventor [19], TigerJython [9,18], or AgentCubes [6].
- **Task-based environments:** If a learning platform is designed around the idea of a built-in collection of programming challenges that programmers can attempt to solve, we consider the platform to be part of the family of task-based learning environments. Examples in this category include the programming environments Code.org, Lightbot, and Emil [7], as well as the programming competition platforms Pisek [11] and Algorea [10].

Regardless of whether students are taught in a free-form environment or on a task-based platform, the ultimate goal of programming education is typically

to make students explore the most important programming concepts, become proficient in them by creating tangible outputs, and learn how to solve problems in an efficient and elegant way. Most programming curricula used in K–6 revolve around the concepts of sequences, repetition, functions, parameterization, branching, and variables [1]. All of these concepts can be taught using an external textbook or via tasks that are directly integrated into the programming environment.

We argue that neither of these approaches alone adequately addresses the needs of teachers at the moment. Whilst free-form programming environments allow for an infinite variety of different programming activities, decoupling tasks from their associated programming platform has the disadvantage that it is not possible to give pupils automated individual feedback as they progress. Moreover, it is not guaranteed that learners actually use programming concepts in a meaningful way [13]. Task-based learning platforms, on the other hand, integrate a curriculum directly into a digital learning platform which allows for automated feedback. The downside of most platforms belonging to this category, however, consists in their limited number of tasks and lack of customization.

We have addressed these shortcomings by adding a task collection mode to the free-form programming environment XLogoOnline, effectively making it a hybrid task-based and free-form programming platform. The proposed tool allows teachers and students to create their own custom tasks which subsequently can be solved and shared with others. Different categories of tasks can be easily assembled in a graphical user interface and submitted solutions are automatically verified by the programming environment. In the following sections, we will present the technical basis of the tool (Sect. 2) as well as a small selection of tasks that have been created by our user base in the past 18 months (Sect. 3).

2 An Extensible Collection of Programming Tasks

XLogoOnline [14] is a Logo learning environment currently used by roughly 70 000 users every year. Most of these users are primary school students or teachers originating from any of the four language regions of Switzerland. There is, however, also a growing population of XLogoOnline users in various other parts of the world such as Germany, Spain, or Lithuania [15]. The platform was originally designed as a free-form programming environment that is used alongside a programming textbook. The textbooks we usually use [2–4] constitute a spiral curriculum from kindergarten to the end of high school [16].

This work is oriented towards K–2 but nevertheless ties into a larger framework that spans K–6 (children aged 6 to 12 years). Throughout the quoted age range, the environment comprises three stages that guide students from their first timid steps in the world of block-based programming with only a handful of unparameterized instructions to full-featured text-based programming with extended features and the possibility to add any number of abstraction layers thanks to user-defined procedures. Due to their linguistic differences (i. e., parameterized vs. non-parameterized basic commands), the application domain

used with the youngest age group is only remotely reminiscent of classical turtle graphics. Instead of drawing geometric shapes using arbitrary angles and lines, novices ages 6 to 8 years learn to steer the turtle through a two-dimensional grid and solve navigation tasks (see Fig. 1). As we will show, whilst a pared-down command set and grid-based task setting heavily reduce the problem domain for the youngest pupils, with minor tweaks this basic concept still permits a wide assortment of stimulating tasks.

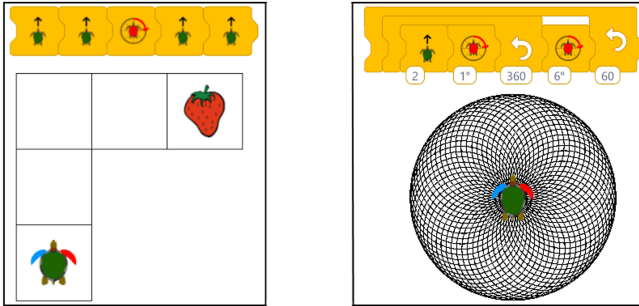


Fig. 1. XLogoOnline contains two different application domains. Students aged 9 years or older are acquainted with traditional turtle graphics where they draw geometric shapes (as shown on the right). Children aged 6 to 8 years, on the other hand, use a simplified vocabulary that does not provide the possibility to draw arbitrary angles. Instead, they learn programming by solving simple navigation tasks as shown on the left.

Programming is not only a challenge for students, but also for their teachers. Children are notorious for only skimming over assignments [20]. As a result, they struggle to verify the correctness of their own solutions and often rely on external guidance to point out mistakes. For teachers with limited experience in programming and holding the responsibility of devoting attention to an entire class, this creates a significant hurdle.

We created a task collection made up of almost 100 predefined tasks¹ which are embedded directly into the XLogoOnline programming environment. All the tasks require students to navigate a turtle on a two-dimensional grid using just the four basic commands `forward`, `back`, `left`, and `right`. To start with, the initial assignments are straightforward and ask the student to find a path to a given object in the grid. Once basic movements have become familiar, additional requirements are introduced such as obstacles that cannot be traversed, multiple objects that need to be collected, or finding a solution with constrained vocabulary. Latter tasks rely on students identifying patterns and writing more concise solutions by identifying symmetry and making use of the command `repeat`.

All of these tasks can be clustered in three rough categories: (i) tasks that contain exactly one target cell, (ii) tasks that contain several target cells which all need to be visited in a specific order or randomly, and (iii) tasks that

¹ <https://xlogo.inf.ethz.ch/release/latest/mini>.

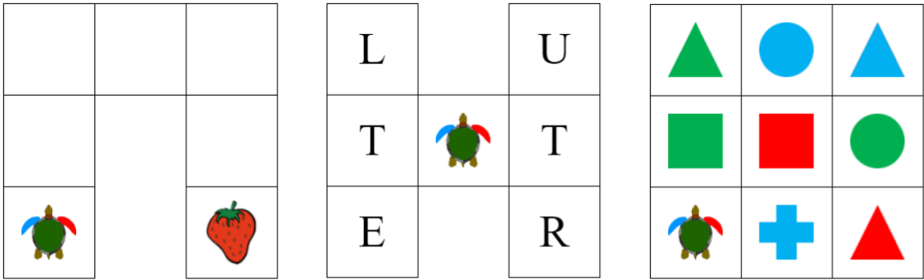


Fig. 2. The three tasks “find the strawberry”, “collect the letters in the order implied by the word TURTLE”, “find a green triangle without standing on a square” represent instances for each of the task categories (single target, multi-target, added restrictions). (Color figure online)

contain additional restrictions such as forbidden cells or a constrained vocabulary. Figure 2 shows some instances of tasks that cover all three of the mentioned categories.

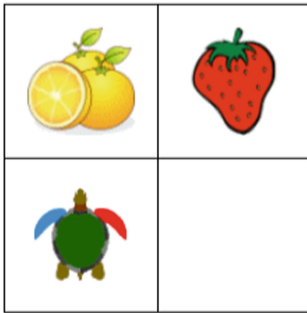
2.1 Finding Common Ground: Task Data Structure

Despite their differing formulation, all the previously-presented tasks share the same fundamental structure. That is, each task consists of three main elements that must be specified individually:

1. **Tiles:** Each task consists of a grid with one or more grid cells that must be arranged contiguously, but not necessarily in a perfect rectangular configuration. Each cell in the grid is uniquely identified by its x- and y-coordinate specifying its absolute location. The coordinate at the top left always corresponds to the location (0,0). All other cells are identified by their horizontal and vertical distance from this cell.
2. **Turtle:** Each task must contain exactly one turtle whose initial location on the grid must be specified in terms of an x- and y-coordinate. The turtle’s initial orientation can be expressed using one of four possible states: 0 (facing North), 90 (facing East), 180 (facing South), or 270 (facing West).
3. **Objects:** As depicted in Fig. 2, there are several kinds of objects that can be used as part of a task assignment. Four categories of objects are provided: (i) strawberry and lemon patches with up to 4 fruit per field, (ii) eight colorful circles, (iii) nine colorful shapes including triangles, squares, and crosses, (iv) Unicode characters such as letters, digits, and emoticons.

Within the programming environment, tasks are represented using an internal data structure that contains detailed information for each of the task elements listed above. Figure 3 shows a side-by-side comparison between a visual task and its corresponding internal representation.

The presented data structure is suitable as a universal representation as it allows millions of different tasks to be described by the same underlying structure. XLogoOnline offers an embedded graphical user interface allowing both



```

"tasks": [
  "tiles": [
    {"x": 0, "y": 0},
    {"x": 1, "y": 0},
    {"x": 0, "y": 1},
    {"x": 1, "y": 1},
  ],
  "turtleData": {
    "x": 0, "y": 1, "dir": 0
  },
  "specialObjects": [
    {"id": "lemon", "x": 0, "y": 0},
    {"id": "berry", "x": 1, "y": 0}
  ],
]

```

Fig. 3. A simple task containing four grid cells, a turtle and two special objects. The code shows how the visual task on the left is represented internally.

teachers and students to intuitively develop and share their own exercises without having to look at the structure inside. The interface is able to interpret and visualize any task written in this format.

Without the possibility to check the correctness of student solutions, the tool permits sharing of ideas but serves only half of its educational purpose. The next logical extension is to explain how the data structure can be extended to automatically verify student submissions and – in the case of faulty solutions – to provide targeted feedback, which we will now elaborate on.

2.2 Solution Verification

In this section, we explain how arbitrary solutions for the task categories presented earlier can be automatically verified. We discuss what different kinds of tasks exist and how each of these task classes can be verified.

So far, we have managed to represent all navigation tasks with one single unified data structure. Therefore, the question arises: Does this conceptual symmetry also apply to the aspect of solution validation? The answer, sadly, is ‘no’. Despite superficial similarities, the correction of seemingly-related tasks often differs substantially and several differing verification strategies are needed to fully cover the spectrum of possible tasks.

Broadly speaking, the realm of different tasks can be clustered into three categories: (i) tasks with one target, (ii) tasks with multiple targets, (iii) tasks with additional restrictions. However, these categories need to be broken down further to actually differentiate all relevant classes from the verification point of view. Specifically, we identify a total of six different task types, namely:

- A: Tasks with one target field.
- B: Tasks with multiple targets.

- B1: Items can be collected in any order.
- B2: Items must be collected in a specific order.
- C: Tasks with additional restrictions.
 - C1: Tasks with obstacles, i.e., forbidden fields.
 - C2: Tasks with walls, i.e., forbidden passage between fields.
 - C3: Tasks with restricted command set.

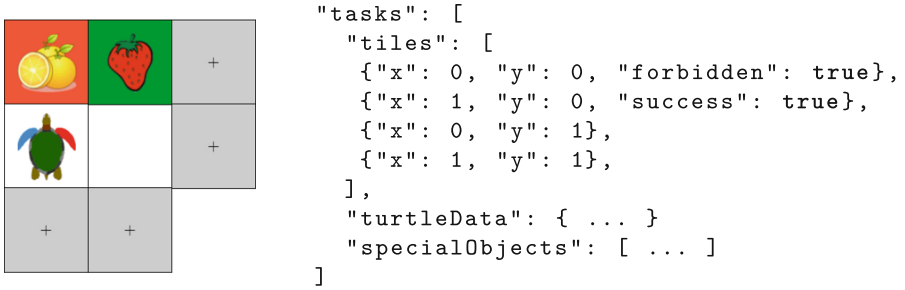


Fig. 4. A simple task containing one success field and one forbidden field.

The constraints and requirements of categories A, C1, and C2 all relate to properties of the grid structure and cells therein. Accordingly, the “tiles” attribute can be annotated with information that describes the specific task type. Any cell in the grid can be labeled as the designated target by setting the “success” flag in it. Likewise, grid cells can be marked as inaccessible terrain by applying a “forbidden” attribute (as in Fig. 4). In addition, the “walls” attribute is used to shield fields from one side only.

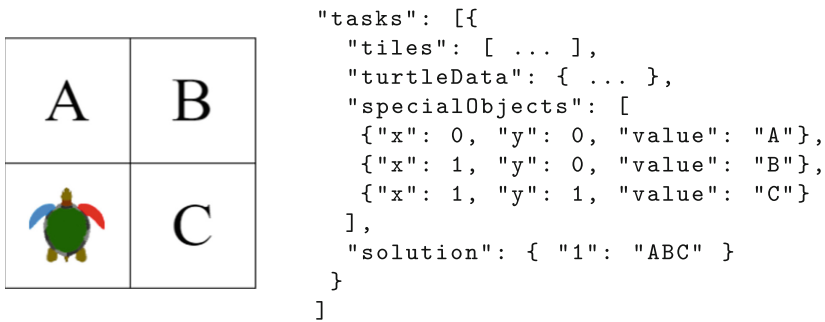


Fig. 5. A task that requires three fields to be visited in a specific order to form the character sequence ‘ABC’.

As soon as a task requires more than one target (as in task types B1 and B2), the question arises whether the individual targets are independent of each other.

If this is the case, it is sufficient to annotate several cells with a corresponding “success” flag. If, however, the cells are to be visited in a specific order (as in Fig. 5), an adapted representation must be used. In this case, the task contains Unicode characters which are accessible via the “value” attribute of a special object. Using these values and a simple concatenation of all visited fields, we are able to distinguish the correct solution “ABC” from an incorrect one like “CBA”.

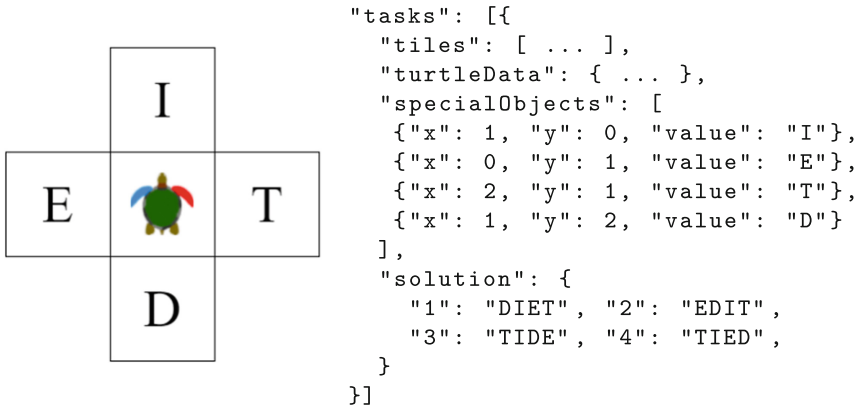


Fig. 6. The task asks for a meaningful concatenation of the four letters ‘D’, ‘I’, ‘E’, and ‘T’. There are, however, multiple correct solutions to this question. We need to note them all as valid solution candidates.

Some tasks may inherently permit more than just one correct solution and there is no technical barrier to prevent users from creating such tasks. Figure 6 depicts a scenario where this is the case: using the four given letters ‘D’, ‘I’, ‘E’ and ‘T’, we are required to form an English word. The obvious solution “DIET” could be substituted by any of the following alternative words: “EDIT”, “TIDE”, “TIED”. All four solutions are valid and should be recognized as acceptable by the automatic verification mechanism. In order to reach this goal, we need to extend the “solution” entry with all possible solutions to the task such that the algorithm is able to validate that the student’s submitted solution comes from this permitted set.

The last category of tasks C2 imposes constraints on the linguistic level (i.e., tasks may restrict the vocabulary that can be used or the number of commands allowed in a solution). This idea is especially intriguing when working with the **repeat** statement which often goes hand-in-hand with a drastic reduction of program length. In order to restrict solutions linguistically, we state which commands are allowed to be used, how many such commands are permitted, or what total length the final program is allowed to have. Figure 7 shows an example of a task with a restricted command set.

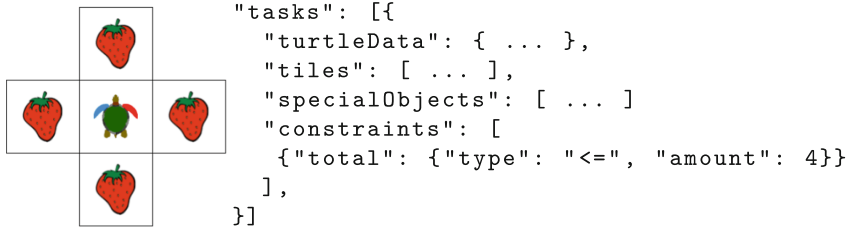


Fig. 7. All 4 berries must be collected with at most 4 commands.

In this section we presented the underlying representation form and mechanisms used to support custom tasks within XLogoOnline’s learning environment. All the previously-presented features are fully integrated and accessible from a graphical frontend. In this way, teachers have the possibility to create custom content for their class and adapt tasks to the linguistic, cognitive, and cultural background of their learners. In the next chapter we dive deeper into this collaborative aspect and present new and imaginative tasks that were created by our user base.

3 Case Study: User Activity During the Past 18 Months

Our tool also offers a graphical frontend from which users can easily design, test and publish custom content directly within the XLogoOnline programming environment. This task creation mode has been publicly available since January 2020 and, from the very beginning, it has been used by both teachers and students alike. Over the past 1.5 years, community contributions have grown to number well over 1331 tasks. To give a glimpse of this valuable dataset, we will now present a few exemplary tasks of particular interest.

One noticeable observation within the data was the presence of numerous misspellings and linguistic errors in a non-negligible proportion of all tasks. These errors are characteristic of our target audience (children aged 6 through 8) and we consequently assume that some teachers actively use the tool with their pupils in class. To help pick apart these differing audiences, we divided the data into two categories using the presence of orthographic errors as a rough indicator of whether the task author was presumably a child or an adult. We acknowledge that this criteria isn’t foolproof and defer a more thorough analysis to future work.

Among the tasks we attribute to adults, we see a broad variety of task types and, in particular, several imaginative examples that rely on transfer from other fields (e.g. spatial reasoning, arithmetic) which were not previously included in our task collection in this form. Figure 8 shows two such examples: (i) The first task requires students to connect digits and mathematical operators in a grid in such a way that the resulting text is both a valid and numerically-correct mathematical statement; (ii) The second task falls into the category of tasks with

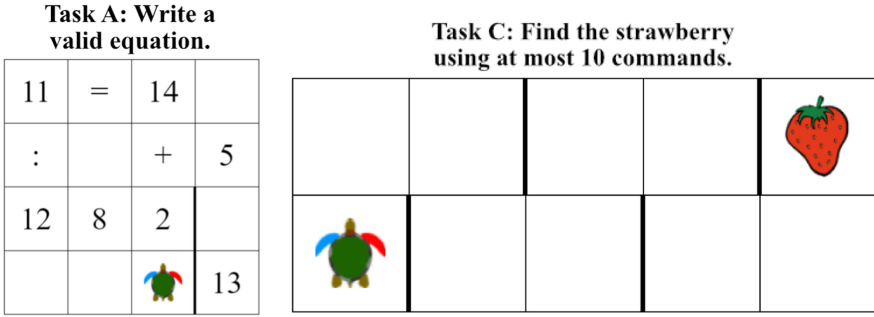


Fig. 8. Two examples of teacher tasks. The task on the left requires pupils to first identify a mathematically valid statement and then make the turtle discover the equation. The task on the right is a natural use case for the repeat statement.

restrictions. In this latter example, the turtle is expected to find the strawberry at the end of a slalom using the shortest possible representation. In order to solve the task, students need to make use the repeat statement. We consider both tasks as a testimony that teachers – when given the opportunity – do indeed come up with highly creative and cognitively-appealing tasks.

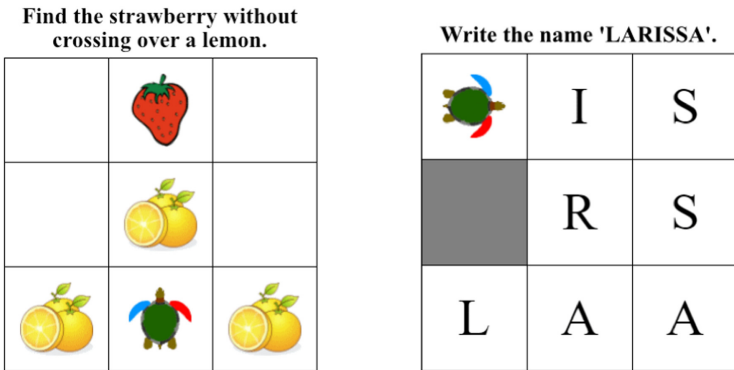


Fig. 9. Two examples of student-created tasks. Both cannot be solved correctly.

Ultimately, tasks are intended to support pupils in their learning. If we instead look at tasks created by children themselves, these are more playful and exploratory in nature. There are numerous interesting examples that show how students explore the limits of the provided rule system or otherwise try to test the feasibility of tasks. Whether intentional or not, there are several instances of tasks that have a clear formulation yet are unsolvable as given and in Fig. 9 we see two such examples. Though the attempt didn't succeed in these cases, this shift to the pupil's point of view is valuable as it gives the teacher a window

into how the pupil operates and what they think. In any case, the tool obviously provides a learning ground for pupils to explore the system they work in and expand their understanding.

4 Conclusion

Computer science has established itself as a vibrant and growing discipline in research, however, is still as-yet unformalized and unfamiliar to the broader population of educators. With many countries having recently introduced Computer Science as a new subject across all age ranges, teachers are being forced to re-orient themselves and are confronted with the daunting task of creating new teaching materials, exercises, differentiation tasks, tests and more. Naturally, there are a range of books, curricula and learning environments that exist, however, these touch on a specific set of topics, present closed facts and overall leave little room for tweaks and adaptation. Precisely this adaptation is vital as it is well-known that educational reforms must be supported and directly driven by teachers to be successful. Formulated differently, this implies that teachers need the possibility to create individual learning opportunities for their students.

In this work, we presented a collection of navigation tasks where pupils direct a turtle around a 2D grid and collect various objects. Aside from demonstrating how the same basic structure permits a range of task difficulties for different ages, we formalized a technological solution that is tightly integrated within the XLogoOnline programming environment. Using the new task collection mode, teachers – or even pupils themselves – are able to playfully design, test, and share their own tasks without any background knowledge. As an added benefit, the environment includes an automatic verification mechanism that provides students with individualized feedback, allows them to progress at their own pace and reduces the burden on teachers. Because of these attributes, the tool is well suited as a platform to host in-class exams or even competitions. We have begun early exploration in this direction and plan to further develop this aspect and eventually create a full-featured programming competition platform that allows pupils to practice in an authentic context, supports the workflow of designing new problem sets, and facilitates distribution and grading of tasks.


References

1. Berry, M.: Computing in the national curriculum: a guide for primary teachers (2013)
2. Gebauer, H., Hromkovič, J., Keller, L., Kosířová, I., Serafini, G., Steffen, B.: Programmieren mit LOGO. ABZ, Ausbildungs-und Beratungszentrum für Informatikunterricht (2015)
3. Hromkovic, J.: Einführung in die Programmierung mit LOGO, vol. 206. Springer, Heidelberg (2010)
4. Hromkovič, J.: Einfach Informatik 5/6: Programmieren. Primarstufe. Begleitband. Einfach Informatik (2019)

5. Huizinga, T., Handelzalts, A., Nieveen, N., Voogt, J.M.: Teacher involvement in curriculum design: need for support to enhance teachers' design expertise. *J. Curric. Stud.* **46**(1), 33–57 (2014). <https://doi.org/10.1080/00220272.2013.834077>
6. Ioannidou, A., Repenning, A., Webb, D.C.: AgentCubes: incremental 3D end-user development. *J. Vis. Lang. Comput.* **20**(4), 236–251 (2009)
7. Kalas, I., Blaho, A., Moravcik, M.: Exploring control in early computing education. In: Pozdniakov, S.N., Dagienė, V. (eds.) *ISSEP 2018*. LNCS, vol. 11169, pp. 3–16. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02750-6_1
8. Kirk, D., MacDonald, D.: Teacher voice and ownership of curriculum change. *J. Curriculum Stud.* **33**(5), 551–567 (2001). <https://doi.org/10.1080/00220270010016874>
9. Kohn, T.: Teaching python programming to novices: addressing misconceptions and creating a development environment. Ph.D. thesis, ETH Zurich, Zürich (2017). <https://doi.org/10.3929/ethz-a-010871088>
10. Léonard, M.: Score et chronomètre sur l'interface: quels effets sur les résultats et la perception d'un concours en ligne? Actes des huitièmes rencontres jeunes chercheur-es en EIAH, p. 56 (2020)
11. Lokar, M.: Pišek - programming with blocks competition a new Slovenian programming competition. In: Kori, K., Laanpere, M. (eds.) *Proceedings of the International Conference on Informatics in School: Situation, Evaluation and Perspectives*, Tallinn, Estonia, 16–18 November 2020. CEUR Workshop Proceedings, vol. 2755, pp. 1–12. CEUR-WS.org (2020). <http://ceur-ws.org/Vol-2755/paper1.pdf>
12. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch programming language and environment. *ACM Trans. Comput. Educ.* **10**(4) (2010). <https://doi.org/10.1145/1868358.1868363>
13. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Habits of programming in Scratch. In: *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011*, pp. 168–172. Association for Computing Machinery, New York (2011). <https://doi.org/10.1145/1999747.1999796>
14. Staub, J.: xLogo online - a web-based programming IDE for Logo. Master's thesis, ETH Zurich, Zürich (2016). <https://doi.org/10.3929/ethz-a-010725653>, masterarbeit. ETH Zurich
15. Staub, J.: Programming in K-6: understanding errors and supporting autonomous learning. Ph.D. thesis, ETH Zurich, Zurich (2021). <https://doi.org/10.3929/ethz-b-000491971>
16. Staub, J., Barnett, M., Trachsler, N.: Programmierunterricht von Kindergarten bis zur Matura in einem Spiralcurriculum. *Informatik Spektrum* **42**(2), 102–111 (2019). <https://doi.org/10.1007/s00287-019-01161-6>
17. Szabo, C., Sheard, J., Luxton-Reilly, A., Becker, B.A., Ott, L.: Fifteen years of introductory programming in schools: a global overview of K-12 initiatives. In: *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, pp. 1–9 (2019)
18. Trachsler, N.: WebTigerJython - a browser-based programming IDE for education. Master's thesis, ETH Zurich, Zurich (2018). <https://doi.org/10.3929/ethz-b-000338593>
19. Wolber, D., Abelson, H., Spertus, E., Looney, L.: *App Inventor*. O'Reilly Media, Inc. (2011)
20. Wolf, M.: *Reader, Come Home: The Reading Brain in a Digital World*. Harper, New York (2018)



Problems, Professional Development and Reflection: Experiences of High-School Computer Science Teachers in Serbia

Vojislav Vujošević^(✉) 

Norwegian University of Science and Technology, Trondheim, Norway
vojislav.vujosevic@ntnu.no

Abstract. Computer Science in K-12 education has had many issues over the years, especially for those who teach—creating lessons, finding resources, lacking official education in the field, keeping up with constant changes in the field, and feeling isolated. These issues have been mapped by various research efforts, where one found that two-thirds of teachers in the US lack a degree in the field. Contributing to the list of issues and their understanding can show a better path towards helping K-12 computer science teachers with their practice. Furthermore, the need to understand the specific context of computer science education in relation to a country is present. It further helps understand the specific needs that teachers have for their practice and professional development. We investigate the challenges in teaching computer science at the high-school level in Serbia while exploring opportunities for professional development and usage of reflection in teachers' practice. We focus on teachers as the audience, which has to stay up-to-date in a fast-changing computer science field while facing numerous issues already found in the literature and voiced at teachers' conferences, seminars, and workshops. Participants expressed having issues with content and ever changing nature of computer science. While they expressed the need to learn and improve, professional development courses are lacking which is taking the toll on the teachers. Reflection, although useful, is not used by everyone and mandatory reflection only adds more paperwork to already high workload.

Keywords: Computer science education · K-12 · Challenges · Professional development · Reflection

1 Introduction

The computer science (CS) field is somewhat young in education, compared to the well-established subjects such as mathematics and physics. While many things are going on at the university level, CS in K-12 is relatively small, and there are many more questions than answers.

CS teachers often do not have a degree in the field, which is true for two-thirds of CS teachers in the US [9]. These and similar findings show the need for

training in-service CS teachers and supporting them in developing their skills and knowledge while also creating a community they can turn to for support [15]. Over the years, many PD efforts have emerged [11, 12]. However, existing programs are usually in the form of summer schools, seminars, and short-term solutions; additionally, it is said that these solutions require using teacher's free time, travel, and additional financial resources [12].

In well-established and less change-prone subjects, it can be expected that the competencies a teacher needs to have are clear and set. However, as CS is a field with constant changes in content, needs, and relevancy, official teacher competencies are absent. Additionally, across countries, CS programs differ, and there is no consensus on what a High-School student should know after graduation.

Understanding what teachers are doing and can do about learning while at work is a crucial point. Teaching is an overall high workload and responsible work. Adding to that constant changes in the field and staying up-to-date, learning in the workplace represents a high potential to bridge the gap between work and PD.

2 Background

2.1 Challenges in Teaching Computer Science

Teachers encounter many challenges in their careers. Having to balance transferring knowledge with paying attention to student needs is in itself a very demanding position to be in. Adding the characteristics of CS to the mix greatly complicates this work. It was shown that CS teachers face problems of different nature; these include feeling isolated, lack of PD opportunities, and lack of official CS teaching training or a degree, to name a few [20]. It is important to note that mentioned challenges represent the perspective of teachers in the US. Since each country is finding its way of dealing with CS and figuring out the best practices, these challenges might not be present elsewhere. A study from the UK says that implementation of computing raised challenges for teachers such as knowledge of the matter, how to approach the teaching and how to assess it [18]. However, the challenges mentioned in the UK relate to the education reform that calls for the inclusion of computing into other subject matters and not computing subject itself.

CS is, compared to other subjects, relatively new and it is still not well established, which is one of the challenges as teachers struggle to identify the subject matter that they are teaching [14]. Authors state that lack of official training, official certification, and a mix of CS with other topics hinders building a sense of identity as a CS teacher. It is worth mentioning that literature is missing further and systematical identification of challenges that teachers face. The most comprehensive list of challenges comes from [20]. The authors identified *content, pedagogical and assessment challenges* as the main ones. In-depth results show limited content knowledge, having a hard time teaching while learning, keeping all students engaged and focused due to the student-centered nature of

CS, student group size, difficulty in addressing students individual CS needs, and lack of suitable assessment tools.

2.2 Computer Science Teachers' Professional Development

Majority of teachers in the US have no or little experience in the field of computer science and with programming [13]. The Committee on European Education stated in their report [5] that CS teachers should have formal competencies. However, the current situation is that there is a lack of CS teachers with formal competencies and they recommend to perform continuous teacher training as one of possible solutions [5]. To that extent, another report states that the major problem among CS teachers is lack of content knowledge [8]. Looking at studies that reported problems among these lines, it is obvious that, however similar the issues are, they need to be seen from the country specific point of view as the differences are vast. For example, curriculum in each country are quite different, which can even be a case from municipality to municipality. Another example is differences between schools - as in Serbian High Schools teachers can choose different books and different programming languages these differences among teachers become even bigger. With these problems in mind we see the need for both pre-service and in-service training for CS teachers, which can not only help them with the content knowledge but also with pedagogy needed for teaching CS. For example, in the US, in comparison to other subject areas such as mathematics, CS PD efforts are modest, and one in 10 schools has a CS teacher in their staff [6, 12]. A review of the studies that were published between 2004 and 2014 states that the PD efforts are on the rise and that they are increasing in number [12], however, it is worth mentioning that the authors looked at traditional PD form - summer schools and workshops. Authors found that most of the PD programs lasted less than one week in addition to more than a half being one-time program and not spread over time. Most of the programs have incorporation of computational thinking as their main goal and right behind is increasing CS knowledge and skills of teachers, especially those with low to no knowledge of CS. Authors, however, did not state to which extent is this part practical and to which is it theoretical and use of particular software in these PD programs was reported as such which might show hands-on part of these PD's.

2.3 Reflection in Education

Reflection has been researched for many years, and it has proven to be an important concept in education. Schon [17] has argued that reflection is an important learning tool in professional education and that, due to the variety of complex and sometimes unusual problems that arise, reflection could be a good solution in practice. Over the years, many definitions of reflection have been established, some general while some concentrated on specific use cases. An accepted definition that is most appropriate from the standpoint of education was given by Boud et al. [2]: *reflection in the context of learning is a generic term for those*

intellectual and affective activities in which individuals engage to explore their experiences in order to lead to new understandings and appreciations. Translating this definition into the practice of a teacher, we can see that, for example, using reflection in the context of experiences in the classroom can lead to new-found knowledge that should bring a change in the behavior of a teacher, thus improving the practice. However, this process is not as straightforward as it can seem to be. Many research efforts have explored and proposed various models and tools to use reflection in education efficiently. The use of reflection in education is said to encourage critical thinking skills, and understanding of self within the practice [16].

One of the ways a person can keep track of their reflections is by keeping a reflection diary. A study reported benefits of said activity for teachers to be *fostering self-awareness, constructing and expanding personal understanding, developing reflection and reasoning skills* [1]. However, it is important to see what are the opposing sides of this activity. The same study reported *complex preparation for writing such a journal and conflict between transmission-oriented schooling and a reflective task*. Although most of the studies are not concerned with CS teachers but rather the ones from a variety of other subjects, characteristics could be transferable before the consideration of the specific needs of CS as a subject.

Another study that used reflection diaries, this time at a university level, for CS teaching assistants, reported positive results. Authors state that reflective diaries are the most prominent of tools for reflective practice with the opportunity for self-education and improves performance [19]. The team found that such a tool supports reflective practice, connects teacher training sessions and actual teaching, and the tool had a broader impact, e.g., being used by TA's from other subjects and faculties. As with any tool, there were negative sides - *dropout and irregular use, clash with other tools that were in use, etc.* [19].

2.4 Study Reasoning

Given the previously mentioned issues and topics, CS has its own set of problems regarding teaching. In a field with swift and irregular changes, it becomes obvious why a teacher would have issues to start with. In addition, having large groups in an individual-centered subject where problem-solving is a dominant way of working, we see that CS teachers have problems of different nature. Exploring mentioned topics with HS CS teachers in the context of country-specific regulations can yield exciting findings towards a more profound understanding of these issues and offer a direction towards future studies in offering a research-based solution. We asked:

What are the challenges that HS CS teachers face in their practice?

Which PD opportunities are offered to teachers and how do they see it?

Do teachers, and in what way, employ reflection in their practice?

2.5 Computer Science Education in Serbia

The following information was obtained from interviews and discussions with participants and professors at the University of Belgrade.

The subject of Computer Science in Serbia is called *Computing and Informatics*, and it is mandatory for students from first to last year of High School (Ages 15 to 19). There are official books for the subject offered by the Ministry of Education from various publishers and authors, out of which schools choose one that their teachers will use for this subject. The program is, therefore, established and followed by the teachers.

The official program from the Ministry of Education of Serbia for CS (natural-mathematical (NM), social-linguistic (SL)):

- First year - 74 classes
 - Information and communication technology in modern society
 - Computing
 - Organizing files and adapting the working environment
 - Crating and editing digital files
 - Programming
- Second year - 74 classes
 - Spreadsheet software
 - Word processor
 - Programming
- Third year - 37 classes (NM & SL)
 - Vector drawing software (NM & SL)
 - Web Design (NM & SL)
 - Java Script basics in relation to Web page presentation and functionality (NM)
- Fourth year - 66 classes (NM), 33 classes (SL)
 - Databases (NM & SL)
 - Computer networks and internet (NM)
 - Servers (NM)
 - Areas of application of modern computing (NM & SL)

The University program that offers education for future CS professors is offered at the Faculty of Mathematics. According to the professor at this university, the numbers of students applying for this program are getting lower each year due to the high salary difference between working in IT Industry and education. However, it is possible to teach CS with education from and university oriented towards engineering, technology, and similar. However, there is a process of certification towards teaching CS, but the process is unclear.

3 Methodology

This study is a part of a Ph.D. thesis research work aiming to explore how reflection can aid in-service teacher practice and professional development. Study with Serbian HS CS teachers is envisioned as an explorative study to determine the issues teachers face, use of reflection and views, and cases of PD opportunities.

3.1 Sample and Recruitment

The target sample was *in-service HS CS teachers in Serbia*. This sample was chosen on the merits of experience in the field that can yield the best possible experience overview regarding the research question.

Sample recruitment was done through several channels: *Ministry of education public list of CS teachers, direct contact with the randomly chosen principal of public high schools in Serbia, contact with instruction manual book authors who collaborated with a larger group of in-service CS teachers (list contained around 100 active CS teachers)*. Therefore, the participation was voluntary, and no compensation for participation was offered. The sample included 11 HS CS teachers, nine of whom teach in general HS while two teach in vocational HS.

3.2 Semi-structured Interview: Reasoning and Procedure

This study is of explorative and descriptive nature, and all of the data collected is qualitative. The semi-structured interview was chosen due to its flexibility as it was important to capture the opinions and the experiences of the sample while allowing them to express themselves more broadly [7]. Additionally, having in mind that the study is a smaller scale and that author wanted to capture the experiences, the choice of the method showed to be appropriate [4].

All of the data in this study was obtained through semi-structured interviews. Due to the global pandemic, most of the interviews were conducted in an online form, using the preferred software solution of the participants. Several interviews were conducted in person where there was a need for it due to the participants' busy schedules.

Participants received an invitation for participation in the research through the mentioned channel with the author's contact information. The invitation contained information about the purpose of the research, the methodology that was used, the way of obtaining the data (through audio recording on a non-internet connected device) as well as the information that no personal details will be collected and that any identifiable information will be omitted in the transcripts. Participants were informed of the right to stop the interview at any point or to withdraw their records at a later time.

3.3 Data Recording and Analysis

As previously mentioned, data was collected using a non-internet connected dictaphone. The recordings were transcribed and analyzed using thematic analysis. The analysis was performed using steps described in [3, 10]. The thematic analysis provides the opportunity to find important and interesting patterns in the recorded data and then use the recognized themes to answer and discuss the proposed research questions [10].

4 Results

Thematic analysis of the interviews with Serbian high school computer science teachers revealed several themes about challenges, competencies, professional development, and reflection in their practice.

4.1 The Need for Learning

These teachers expressed the need for constant learning and improvement within their professional practice, like what students do.

Content Knowledge. The majority of the participants pointed out that a competent CS teacher needs to have as much knowledge as possible. Several participants added that they assume this is true for any subject taught at school, but CS has rapid changes that happen often.

- *“I assume that every teacher needs to have very good knowledge on their subject, same goes for the subject that I teach. It is definitely harder to keep that over the years as you get older as things in this subject are changing, and they are changing fast, and quite quickly you can find yourself being run-down by the “tech kids”.”*
- *“...constant learning. I think that this is imperative for any CS teacher or professional. I need to know as much as possible about all the themes that I teach so I can transfer that knowledge efficiently to the students, and I can then get creative when I feel comfortable.”*
- *“Learning, constant learning! That is what I tell everyone, even my colleagues. There are changes so often, which is also why I like CS, it keeps me on my toes, but it sometimes is hard to keep up, especially with age.”*

Mastering the content knowledge is important for a teacher, but what participants pointed out is a constant need for this in CS. While there is some stability, similarly to other subjects in theory and history, other areas in CS are subject to swift and often irregular changes.

Practical Professional Development. CS teachers in Serbia have to attend a two-day seminar viewed as a licensing and PD course. The Ministry of Education organizes these seminars in collaboration with a private company or an institution in charge of a PD part, while another entity is in charge of presentations.

Most of the participants view these seminars negatively, describing them as some numbers on a canvas during the first day - often not CS directly related, and some quick course on the second day. It was expressed by the participants that they cannot learn anything useful for their class during this PD effort, and they mention the need for a more serious course or summer school.

One participant briefly explained:

- *“The seminars that I mentioned, those are the only organized, so-called PD that I am aware of! I went to a few of them. It is all just talk and no useful advice. I think those are organized out of formality and that some company...”*
“There needs to be more PD opportunities but organized by professionals in the field, and they need to be practical - for example, a crash course on Python would be nice, even an online one but explained in accordance to the book!”

Most of the participants explained that they are eager to learn and improve through courses, but in-person courses are hard to find, they are not counted for working hours. Furthermore, it was said that practical courses would create better teachers and make their jobs easier, and adapt to changes.

Other Knowledge. In addition to content knowledge, these teachers see additional knowledge that a CS teacher should have and improve.

Handling the equipment and the ability to troubleshoot was mentioned by several teachers. It was pointed out that when a computer or a specific software returns an error, a teacher should quickly solve it. Otherwise, they run into the risk of losing valuable time during a class.

- *“...computers are outdated, which does not mean that it is not possible to work on them, but if you do not have good technical literacy as a professor, it will not be easy for you to solve problems that occur with outdated equipment.”*

This quote shows that a teacher should be able to solve a problem on both the software and hardware end. Again, this statement can be true for a teacher of other subjects and their equipment, e.g., a Biology teacher and a microscope.

4.2 Other Challenges

In the previous theme, we saw that in addition to the need that teachers have for PD, learning, and improving, they have challenges with those exact segments. Continuing, we saw other challenges that these teachers discussed.

Poor Instruction Manual - Subject Book. There is a subject book or an instruction manual that closely follows the curriculum, provides instructions, definitions, and knowledge in the written form for students. For teachers, these books are sources of examples and exercises and written forms of knowledge they refer to. However, the books that are approved do not always meet this criterion for various reasons.

- *“There was a good book many years ago, but it is obsolete now. Usually, they make a new book that is a recycled version of an old one, and then we have to fill in a lot from the internet or our own knowledge. What they do now is give us guidelines, for example - go to this website and learn from there.”*

Some participants raised the point of the subject book falling short of what it is supposed to offer. Participants usually discuss political matters and policy problems that refer to the Serbian context when asked to elaborate.

- *“Book with instructions is quite obsolete, let us say. It feels like it is something from the 80’s and there are not a lot of practical or relevant things in there, yet we have to use it.”*

It is important to understand that there are usually several books by different authors in the Serbian school system that are offered for a school year. That is followed by CS teachers of one school discussing and choosing one book listed as the official, mandatory book offered in that one school. CS in another school can be taught with a different book. Hence a teacher from one might not be able to help a teacher from another. One participant pointed this out:

- *“...school can choose a book that the Ministry offers, and there is a choice, but books are wildly different, so if a student from another school would ask me something, I might have no idea what it is about, and nothing about the topic that they are learning. So, two schools 1km apart can have completely different matters as far as CS is concerned.”*

A variety of books led to teachers reporting a lack of uniformity where one can expect a student from any HS to have certain knowledge, or when one teacher enters a discussion with a colleague from another school they might, and quite often are, in different CS themes during a school year.

Curriculum Changes. Each school reform brings a set of changes that are supposed to improve upon previous and further improve the curriculum of various subjects. A school reform is a big job, and it requires time and vision. However, due to various factors, these reforms in Serbia happen irregularly, sometimes with a long break in between or too often. The participants pointed out that current and some previous reforms seem more like a formality than serious and comprehensive work aiming to help teachers and students. This leads to more workload for teachers, radical changes, and it imposes new topics for CS teachers without offering them preparation.

- *“CS reforms are happening too often without teacher involvement. Just before the school starts, they change the content, order of the topics, a book, etc., and I have to adapt in a matter of weeks, even though I disagree with the changes. And this is the fourth reform in a short period of time.”*

When these reforms are imposed on teachers without preparation time or presentation of new materials, it creates a high workload for teachers, possible insecurities while possibly lowering the quality of teaching.

Student Motivation. Several participants have expressed difficulty in motivating the current generations of students. Teachers mainly refer to the outdated equipment being the issue and the generation of students used to mobile technologies and not the stationary computer. Teachers also stated that students do not see or understand the need for any of the materials taught at this level.

- *“Another thing is that it seems to me that students are demotivated when they see outdated computers because their phones are in 90% of the cases more powerful hardware devices...they are mostly interested in an attractive user interface and do not see the benefits of programming...”*
- *“Unmotivated students. They work on old computers and they are impatient and cannot wait for a slow computer to open software. Also, I am not sure that they see much point in many of the themes if they are not interested beforehand. They are mostly interested in their phones. If I could teach them about those, I think they would love it.”*

4.3 I Reflect and I Do Not Reflect

CS teachers in Serbian high schools are required to fill in what is called *pedagogical diaries*. These diaries contain evaluation (without grades) for all students and finally for teachers themselves. This activity has questions that could trigger personal reflection in teachers. However, it has a vast amount of paperwork to be completed that teachers report being a hindrance in their work. Teachers mostly did not express positive opinion towards these diaries as they did not see a real value of having such a document. Without clear access to these diaries one postulate that having a mandatory reflection on each student through a specific format is far too much work for one person, while at the end a teacher cannot dedicate enough time for self reflection and evaluation.

Several participants reported that reflection in general is helpful for them and their practice and that they use it from occasionally to regularly. Most of the participants who reflect do so when the issues in their practice arise.

- *“Today I usually reflect on some small issues, or if I really see that I need to change something in the way I teach...It has been so far quite useful to improve my teaching. For other situations, it has been quite irrelevant.”*

A participant pointed out that an individual cannot solve general issues discussed in previous themes and that reflection is of no use.

Participants who reflect reported that they write their thoughts down in their notebooks while others keep thoughts in their heads. They explained that their style of thinking is in effect here. Those who keep their reflections in their head do so also to avoid more paperwork.

AS far as the *pedagogical diaries* go, teachers are divided with their feelings towards it.

- *“The ministry requires us to use a pedagogical diary in which we should enter problems, goals, progress and the like, but I have neither the will nor the time to deal with the paperwork of that type.”*

- *“I also have to complete self-evaluation, which helps me with some prompting questions when I am tired and do not have much energy to think about it.”*

Those who reflect say that the practice was not due to their education as a teacher but rather something they found helpful over the years of practice or something they heard or saw from their colleagues. Those who reflect additionally stated that reflection helps them progress, tackle issues that arise, and lower the workload in some cases.

5 Discussion

Computer Science as a field differs from others in several areas. Swift and irregular changes in the field, individualistic nature, problem-solving approach, lack of unity of what is taught on a global and local level, to name a few. The voices of teachers are important, given that they are responsible for transferring knowledge in mandatory education. The context of a country is important. Serbia has specificities in how CS is taught, how CS is approached, what curriculum consists of, etc.

The issues that these teachers reported match some of the issues reported in the literature. Namely battling the content knowledge in CS [8,20]. Due to the nature of CS, it is obvious why this presents the problem for in-service teachers and why this is explored for pre-service teachers and those in training. This issue goes directly into the growing need for well-organized, thought-out, and connected with curriculum professional development courses, which is in line with lack of such efforts as reported in the literature [12]. However, with such PD efforts, there is a problem of time consumption for the participants. Having already high workload of teaching, adding PD courses might be problematic for some, which is why we asked about the experiences of reflection and reflective practice. Literature showed the usefulness of reflective practice and its importance in learning on the job, thus gaining professional improvement. This study found that not all participants reflect, mostly said due to time consumption or lack of perceived usefulness. However, those who reported reflecting on their practice voiced their positive outcomes with this activity. In their practice, they gained valuable insights and behavior change in teaching that they saw as improvement, in some cases short term while in other long-term.

At the present state, teachers are trying to tackle these issues on their own using various techniques. However, they reported that it is taking a toll on them, and some of them also reported the loss in their professional motivation.

Acknowledgement. I want to take this opportunity to thank my mentor, Monica Divitini, for help and guidance. I would also like to thank everyone who participated and helped with sample recruitment and their knowledge about the CS situation in Serbia.

References

1. Abednia, A., Hovassapian, A., Teimournezhad, S., Ghanbari, N.: Reflective journal writing: exploring in-service EFL teachers' perceptions. *System* **41**(3), 503–514 (2013). <https://doi.org/10.1016/j.system.2013.05.003>
2. Boud, D., Keogh, R., Walker, D.: *Reflection: Turning Experience into Learning*. Routledge, London (2013)
3. Braun, V., Clarke, V.: Using thematic analysis in psychology. *Qual. Res. Psychol.* **3**(2), 77–101 (2006). <https://doi.org/10.1191/1478088706qp063oa>
4. Drever, E.: *Using semi-structured interviews in small-scale research. A Teacher's Guide*, Scottish Council For Research In Education (1995)
5. The Committee on European Computing Education (CECE): *Informatics education in Europe: are we all in the same boat?* Technical report (2017). <https://doi.org/10.1145/3106077>
6. Guzdial, M.: We may be 100 years behind in making computing education accessible to all. (Blog post published at the communications of the ACM) (2014). <https://bit.ly/3AcTyLU>
7. Horton, J., Macve, R., Struyven, G.: Qualitative research: experiences in using semi-structured interviews. In: *The Real Life Guide to Accounting Research*, pp. 339–357. Elsevier (2004). <https://doi.org/10.1016/B978-008043972-3/50022-0>
8. Lang, K., et al.: *Bugs in the system: computer science teacher certification in the USA report by the computer science teachers association and the association for computing machinery*. Technical report, Computer Science Teachers Association (2013). <https://aaas-arise.org/resource/bugs-in-the-system/>
9. Leyzberg, D., Moretti, C.: Teaching CS to CS teachers: addressing the need for advanced content in k-12 professional development. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. SIGCSE '17*, pp. 369–374. Association for Computing Machinery, New York (2017). <https://doi.org/10.1145/3017680.3017798>
10. Maguire, M., Delahunt, B.: Doing a thematic analysis: a practical, step-by-step guide for learning and teaching scholars. *All Ireland J. High. Educ.* **9**(3) (2017). <http://ojs.aishe.org/index.php/aishe-j/article/view/3354>
11. Martinez, M.C., Gomez, M.J., Moresi, M., Benotti, L.: Lessons learned on computer science teachers professional development. In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016*, pp. 77–82. Association for Computing Machinery, New York (2016). <https://doi.org/10.1145/2899415.2899460>
12. Menekse, M.: Computer science teacher professional development in the united states: a review of studies published between 2004 and 2014. *Comput. Sci. Educ.* **25**(4), 325–350 (2015). <https://doi.org/10.1080/08993408.2015.1111645>
13. Milliken, A., Cody, C., Catete, V., Barnes, T.: *Effective computer science teacher professional development: beauty and joy of computing 2018*. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 271–277 (2019). <https://doi.org/10.1145/3304221.3319779>
14. Ni, L., Guzdial, M.: Who am I? Understanding high school computer science teachers' professional identity. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, SIGCSE 2012*, pp. 499–504. Association for Computing Machinery, New York (2012). <https://doi.org/10.1145/2157136.2157283>

15. Ni, L., Guzdial, M., Tew, A.E., Morrison, B., Galanos, R.: Building a community to support HS CS teachers: the disciplinary commons for computing educators. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, SIGCSE 2011, pp. 553–558. Association for Computing Machinery, New York (2011). <https://doi.org/10.1145/1953163.1953319>
16. Scanlan, J.M., Chernomas, W.M.: Developing the reflective teacher. *J. Adv. Nurs.* **25**(6), 1138–1143 (1997). <https://doi.org/10.1046/j.1365-2648.1997.19970251138.x>
17. Schon, D.A.: *The reflective practitioner* (1991)
18. Sentance, S., Csizmadia, A.: Computing in the curriculum: challenges and strategies from a teacher’s perspective. *Educ. Inf. Technol.* **22**(2), 469–495 (2017). <https://doi.org/10.1007/s10639-016-9482-0>
19. Ukrop, M., Švábenskỳ, V., Nehyba, J.: Reflective diary for professional development of novice teachers. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 1088–1094 (2019). <https://doi.org/10.1145/3287324.3287448>
20. Yadav, A., Gretter, S., Hambrusch, S., Sands, P.: Expanding computer science education in schools: understanding teacher experiences and challenges. *Comput. Sci. Educ.* **26**(4), 235–254 (2016). <https://doi.org/10.1080/08993408.2016.1257418>

Author Index

- Chothia, Zaheer 135
- Dagienė, Valentina 16, 95
- Datzko, Christian 83
- Divitini, Monica 123
- Farshchian, Veronica 123
- Grgurina, Nataša 3
- Jevsikova, Tatjana 95
- Juškevičienė, Anita 95
- Kadijevich, Djordje M. 45
- Karpielová, Mária 57
- Kastner-Hauler, Oliver 29
- Komm, Dennis 69
- Lillebo, Miriam 123
- Meškauskienė, Asta 95
- Miková, Karolína 57
- Rouhani, Majid 123
- Sabitzer, Barbara 29
- Schrempp, Larissa 135
- Šiaulyš, Tomas 16
- Staub, Jacqueline 135
- Stupurienė, Gabrielė 95
- Tengler, Karin 29
- Vaniček, Jiří 109
- Vujošević, Vojislav 147
- Wacker, Pascal 135
- Yeni, Sabiha 3