

Jean-Philippe Pellet · Gabriel Parriaux
Editors

Informatics in Schools

ISSEP 2023 Local Proceedings

16th International Conference on Informatics in Schools:
Situation, Evolution, and Perspectives, ISSEP 2023
Lausanne, Switzerland, October 23–25, 2023

Preface

These local proceedings contains full and short papers as well as poster descriptions and titles of workshops presented at the 16th International Conference on Informatics in Schools: Situation, Evolution and Perspectives (ISSEP 2023), which was held at the University of Teacher Education (HEP Vaud) in Lausanne, Switzerland, from October 23 to 25, 2023. They complement the proceedings published by Springer (Lecture Notes in Computer Science, vol. 14296, doi:10.1007/978-3-031-44900-0), which contain other full papers also presented at the conference. This preface, and the following Organization section, are largely identical to the one published in the Springer proceedings.

The ISSEP conference series is a forum for researchers and practitioners in the area of informatics education, in both primary and secondary schools. The conference provides an opportunity for educators and researchers to reflect upon the goals and objectives of this subject matter, its curricula, various teaching and learning paradigms and topics, as well as the connections to everyday life — including the various ways of developing informatics education in schools.

The conference series started in 2005 in Klagenfurt, Austria. Initially planned as a one-time event, interest was such that subsequent editions were organized in thirteen different countries to this day: in Vilnius, Lithuania (2006); Torun, Poland (2008); Zürich, Switzerland (2010); Bratislava, Slovakia (2011); Oldenburg, Germany (2013); Istanbul, Turkey (2014); Ljubljana, Slovenia (2015); Münster, Germany (2016); Helsinki, Finland (2017); St. Petersburg, Russia (2018); Larnaca, Cyprus (2019); Tallinn, Estonia (2020); Nijmegen, the Netherlands (2021); and Vienna, Austria (2022).

As with the previous edition in Vienna, a doctoral consortium was organized for ISSEP 2023, which received 12 applications from Ph.D. students. The doctoral consortium is a place for Ph.D. students to present and discuss their research ideas, meet each other as well as other senior researchers, and get constructive feedback from peers and researchers prior to the conference itself. This year's doctoral consortium was held on October 22, 2023, and was chaired by Engin Bumbacher from HEP Vaud.

On the first day of the conference, local teachers were also invited to attend. Practical workshops were organized on that day in addition to presentations. These workshops are listed at the end of this volume. We believe closer interactions between teachers and researchers are important, on the one hand, in order to ensure that research is relevant to the classroom, and, on the other hand, to provide teachers with a view on the latest developments in the field. The Swiss Society for Informatics in Education (SVIA-SSIE-SSII), mainly composed of computer science teachers, supported the event and also organized its annual general assembly on the first day of the conference in Lausanne.

The ISSEP 2023 program committee received 73 submissions in total (including poster and workshop proposals), out of which 47 were paper submissions.

Each of these was reviewed by between 3 and 5 members of the program committee. In total, 171 double-blind reviews were provided. We are extremely thankful for the dedicated and timely work of the reviewers! Based on the ratings and comments, 14 full papers were selected for publications in the Springer proceedings, and 15 more were selected for these local proceedings. The submission, review, and selection process was managed using the EasyChair conference management system.

Once again, we would like to thank the members of the Program Committee for the work they have done in reviewing the submissions and providing feedback to the authors. We thank the authors for their numerous, high-quality submissions. We are also very grateful to the members of ISSEP's steering committee for their advice and support. Andreas Bollin and Gerald Futschek, as organizers of the previous edition of ISSEP, provided particularly helpful guidance on a whole range of organizational matters and deserve special thanks. We also thank our colleagues and members of the local organizing committee for their work in the concrete organization of the physical conference, as well as our institution's Grants Office, which helped us look for funding and provided support for the publication of these proceedings.

Finally, we would like to thank our partners and sponsors for their generous contributions, without which ISSEP 2023 would not have been what it was: the Swiss National Science Foundation, Google, the Hasler Stiftung, the SVIA-SSIE-SSII association, and the Centre de Compétences romand de Didactique Disciplinaire.

October 2023

Jean-Philippe Pellet
Gabriel Parriaux

Organization

Conference Chairs

Gabriel Parriaux Univ. of Teacher Education, Lausanne, Switzerland
Jean-Philippe Pellet Univ. of Teacher Education, Lausanne, Switzerland

Steering Committee

Erik Barendsen Radboud University and Open University, The Netherlands
Andreas Bollin University of Klagenfurt, Austria
Valentina Dagienė Vilnius University, Lithuania
Gerald Futschek TU Wien, Austria
Yasemin Gülbahar Ankara University, Turkey
Juraj Hromkovič ETH Zurich, Switzerland
Ivan Kalaš Comenius University, Slovakia
Mart Laanpere Tallinn University, Estonia
Sergei Pozdniakov St. Petersburg Electrotechnical University, Russia

Program Committee

Andreas Bollin University of Klagenfurt, Austria
Andrej Brodnik University of Ljubljana, Slovenia
Julien Bugmann Univ. of Teacher Education, Lausanne, Switzerland
Engin Bumbacher Univ. of Teacher Education, Lausanne, Switzerland
Špela Cerar University of Ljubljana, Slovenia
Morgane Chevalier Univ. of Teacher Education, Lausanne, Switzerland
Christian Datzko Wirtschaftsgymnasium und Wirtschaftsmittelschule
Basel, Switzerland
Monica Divitini Norwegian Univ. of Science and Technology, Norway
Gerald Futschek TU Wien, Austria
Micha Hersch Univ. of Teacher Education, Lausanne, Switzerland
Ivan Kalaš Comenius University, Slovakia
Kaido Kikkas Tallinn University of Technology, Estonia
Dennis Komm ETH Zurich, Switzerland
Mart Laanpere Tallinn University, Estonia
Martina Landman TU Wien, Austria
Peter Larsson University of Turku, Finland
Olivier Lévêque Swiss Institute of Technology, Lausanne, Switzerland
Nina Lobnig University of Klagenfurt, Austria
Birgy Lorenz Tallinn University of Technology, Estonia

Maia Lust	Tallinn University, Estonia
Kati Mäkitalo	University of Oulu, Finland
Tilman Michaeli	TU Munich, Germany
Mattia Monga	Università degli Studi di Milano, Italy
Stefan Pasterk	University of Klagenfurt, Austria
Biljana Petreska von Ritter	Univ. of Teacher Education, Lausanne, Switzerland
Hans Põldoja	Tallinn University, Estonia
Sergei Pozdniakov	St. Petersburg Electrotechnical University, Russia
Ralf Romeike	Freie Universität Berlin, Germany
Giovanni Serafini	ETH Zurich, Switzerland
Eva Schmidthaler	Johannes Kepler Universität Linz, Austria
Vipul Shah	ACM India Pathshala Initiative
Gabrielė Stupurienė	Vilnius University, Lithuania
Reelika Suviste	University of Tartu, Estonia
Maciej Sysło	Warsaw School of Computer Science, Poland
Svetlana Unković	TU Wien, Austria
Patrick Wang	Univ. of Teacher Education, Lausanne, Switzerland
Michael Weigend	University of Münster, Germany
Markus Wieser	University of Klagenfurt, Austria

Additional Reviewers

Walter Gander	ETH Zurich, Switzerland
Tobias Kohn	TU Wien, Austria
Alexandra Maximova	ETH Zurich, Switzerland

Doctoral Consortium Committee

Engin Bumbacher	Univ. of Teacher Education, Lausanne, Switzerland
Andreas Bollin	University of Klagenfurt, Austria
Valentina Dagienė	Vilnius University, Lithuania
< Gerald Futschek	TU Wien, Austria
Violetta Lonati	Università degli Studi di Milano, Italy
Tilman Michaeli	TU Munich, Germany
Jean-Philippe Pellet	Univ. of Teacher Education, Lausanne, Switzerland

Local Organizing Committee

Gabriel Parriaux	Univ. of Teacher Education, Lausanne, Switzerland
Jean-Philippe Pellet	Univ. of Teacher Education, Lausanne, Switzerland
Engin Bumbacher	Univ. of Teacher Education, Lausanne, Switzerland
Biljana Petreska von Ritter	Univ. of Teacher Education, Lausanne, Switzerland
Morgane Chevalier	Univ. of Teacher Education, Lausanne, Switzerland
Julien Bugmann	Univ. of Teacher Education, Lausanne, Switzerland
Patrick Wang	Univ. of Teacher Education, Lausanne, Switzerland
Claire Matti	Univ. of Teacher Education, Lausanne, Switzerland
Catherine Audrin	Univ. of Teacher Education, Lausanne, Switzerland
Giovanni Serafini	ETH Zurich, Switzerland

Table of Contents

Papers

Artificial Intelligence in Primary and Secondary Education: a Review of Educational Activities Development	3
<i>Sébastien Combéfis</i>	
Breaking Gender Barriers in Computer Science: Exploring the Impact of Digital Fabrication Workshops in Smart Environments	15
<i>Nadine Dittert, Mareike Daeglau, Nils Pancratz, and Ira Diethelm</i>	
Easy Coding in Biology: Combining Block-Based Programming Tasks with Biological Education to Encourage Computational Thinking in Girls	27
<i>Eva Schmidthaler, Corinna Hörmann, David Hornsby, Anneliese Fraser, Martin Cápay, and Barbara Sabitzer</i>	
Supporting Gender Equality in Computer Science Through Pre-Introductory Programming Courses	37
<i>András Margitay-Becht and Udayan Das</i>	
Investigating Code Smells in K-12 Students' Programming Projects: Impact on Comprehensibility and Modifiability	49
<i>Verena Gutmann, Elena Starke, and Tilman Michaeli</i>	
Supporting Non-CS Teachers with Programming Lessons	61
<i>Svetlana Unkovic and Martina Landman</i>	
MazeMastery – A Python Framework for Teaching Maze-Traversal in High School	75
<i>Raphaël Baur, Jens Hartmann, and Jacqueline Staub</i>	
Computer Science Education with a Computer in the Background	89
<i>Maciej M. Sysło</i>	
Effects of the Use of Robots on Algorithmization, Decentration and Locating in the Plane Skills	103
<i>Emma Schenkenberg van Mierop, Acsa-Loriane Schmidt, and Morgane Chevalier</i>	

Teaching Quantum Informatics at School: Computer Science Principles and Standards 117
Giulia Paparo, Regina Finsterhoelzl, Bettina Waldvogel, and Mareen Grillenberger

Measuring Didactical Competencies for Informatics Education among Prospective Primary School Teachers 129
Christin Nenner and Nadine Bergner

Computational Thinking from Preschool to University: The Versatility of UML Modeling in Education 139
Nina Lobnig and Corinna Mößlacher

Identifying Computational Thinking Behaviors in the Robotics Programming Activity 151
Megumi Iwata, Kateryna Zabolotna, Kati Mäkitalo, Jari Laru, and Jonna Malmberg

Computational Thinking Readiness of Incoming High School Students in Taiwan 167
Greg C. Lee, Jia-Yi Chen, and Yu-Wen Yang

Insights and Conclusions from Analyzing the Hungarian Bebras Initiative in 2021-2022 175
Zsuzsa Pluhár and Bence Gaál

Poster Descriptions

Integrating Computational Thinking with Mathematical Problem Solving	189
<i>Arnold Pears, Javier Bilbao, Valentina Dagienė, Yasemin Gulbahar, András Margitay-Becht, Marika Parviainen, Zsuzsa Pluhar, and Pál György Sarmasági</i>	
A Constructionist Approach for Transitioning to College-Level Mathematics Education	193
<i>András Margitay-Becht</i>	
Enhancing Teacher Education Through STEAM Integration in Informatics	197
<i>Anita Juškevičienė</i>	
GeNIUS: Conditions for Successfully Teaching Computer Science Infused Natural Science Classes in Schools	203
<i>Elena Yanakieva, Annette Bieniusa, Christoph Thyssen, Thomas Becka, Julia Albicker, Niklas Westermann, Barbara Pampel, and Johannes Huwer</i>	
From Wooden Blocks to Whimsical Robots: The “Programmieren spielend entdecken” Series to Nurture our Future Innovators	207
<i>Fatmir Racipi, Stephanie Eugster, and Mathias Kirf</i>	
Finding Patterns in Productive Failure Steps? An Explorative Case Study in a Teaching Learning Lab for Computer Science	211
<i>Frauke Ritter and Nadine Schlomske-Bodenstein</i>	
Gender Differences in Problem Solving Observed in Logo Novices	215
<i>Jacqueline Staub and Angélica Herrera Loyo</i>	
Teaching the Von-Neumann Model with a Simulator	221
<i>Martin Weinert, Jan Hendrik Krone, and Johannes Fischer</i>	
An Approach to Introduce High-School Students to the P-vs-NP Question	225
<i>Jisoo Song, Soyeon Oh, Soyeon Jeong, and Seongbin Park</i>	
Promoting Artificial Intelligence and Data Literacy within Teacher Education	229
<i>Valentina Dagienė, Martin Kandlhofer, Vaida Masiulionytė-Dagienė, Viktoriya Olari, and Ralf Romeike</i>	

Exploring Students' Preinstructional Mental Models of Machine Learning:
Preliminary Findings 233
Erik Marx, Thiemo Leonhardt, Nadine Bergner, and Clemens Witt

Teachers' Experience Regarding Digital Threats for Children and Teenagers 237
Julian Taupe, Verena Knapp, and Andreas Bolln

Exploring the Relationship between Digital Competences and Understanding
of Informatics Education: A Study on Primary School Teachers 241
Gabrielé Stupuriené

Teaching an Elective Course about Quantum Computing 247
*Jihyun Kim, Chaeyeon Lee, Jisoo Song, Chaeyoung Sim, and
Seongbin Park*

Teaching Quantum Computing at a Middle School 251
Sunrim Lee, Yuri Kim, Soyeon Jeong, and Seongbin Park

From Verbalization in Problem Solving on Computational Thinking Tasks
to the Abstraction of Block Programming Concept under Scratch 255
Karima Sayeh

From Tree to Forest: Determining the Probability of Scoring a Goal in
Football Games 259
Jan Hendrik Krone and Johannes Fischer

Workshops

List of Workshops 265

Papers

Artificial Intelligence in Primary and Secondary Education: a Review of Educational Activities Development

Sébastien Combéfis^{1,2}[0000–0002–8987–9589]

¹ Computer Science and IT in Education ASBL, 1348 Louvain-la-Neuve, Belgium

² AEI Consulting, 1348 Louvain-la-Neuve, Belgium

sebastien@combefis.be

<https://sebastien.combefis.be>

Abstract. Intelligent systems are widespread in everyday life. Today, more than ever, artificial intelligence (AI) is being applied to many domains and its societal relevance is growing rather rapidly. It is therefore important to include AI early in education, as a subject for pupils to apprehend and learn. Future citizens must be capable to understand the technology behind intelligent systems, at least globally. This paper reviews the activities and tools that are being developed to teach AI to young pupils in primary and secondary schools. Its goal is to identify the various kinds of activities designed by researchers, like games, unplugged activities, workshops, etc. It also aims at analysing what are the subfields of AI covered by developed activities. To conclude, this paper draws up perspectives on future development the research community may investigate further, to better educate young pupils to AI.

Keywords: Artificial intelligence · Education · School.

1 Introduction

Intelligent systems are an integral part of the society and widespread in everyday life [23]. With the rise of cyber-physical systems, intelligent machines equipped with artificial intelligence (AI) are spreading [28]. A direct consequence is the need for current and future citizens to have some knowledge on these subjects.

Artificial intelligence has been established as an academic discipline in the 1950s [17]. It was only recently it left the scientific obscurity to reach the business world and the public at large. Text generation, image recognition, self-driving vehicles, intelligent household appliances, smartphones and smart speakers with embedded assistants are just a few examples of concrete applications of AI that can be used by anyone today [2]. On the one hand, many people know about the existence of devices and services based on AI but, on the other hand, only a few individuals understand the technology behind them. The underlying process used by artificial intelligence, and more specifically machine learning (ML), is a black box for many users [18]. Since AI and ML concepts are not trivial, there is a justified reason to “black box” them in consumer products.

As a direct consequence of this invasion of “hidden” AI, everyone needs the competencies to better understand it. More precisely, people should be able to be aware of the impact, opportunities, and limits of AI on their personal lives and our society [27]. This also results in a big challenge for education, starting with younger pupils [20, 38]. Ever-younger children have indeed become active users of online services, like YouTube, WhatsApp, Instagram, Spotify, Snapchat and TikTok, which are using user data and machine learning for privacy-intrusive purposes. As AI-based services become more ubiquitous, it is increasingly urgent to build familiarity with AI technologies to all people, including children, since they will be interacting more and more often with them in the near future [41].

Learners should both be able to explain AI-related phenomena that they observe in their lives and, in some extent, to use AI-based tools to actively and creatively shape the so-called digital world in which they are living. However, learners should not be overwhelmed because of the complexity of an unknown subject [18, 24]. This is especially true for young learners, who do not have enough backgrounds nor prerequisites to be able to understand all the complexity of artificial intelligence. For early stage pupils, the need is therefore to introduce them to AI concepts without burdening them with inner complex details.

1.1 Related Work and Motivations

This paper is focused on teaching artificial intelligence to young pupils. Research on how to introduce AI concepts to them started in the eighties, with the focus mainly put on expert systems [33]. Later, in the nineties, the tic-tac-toe game was used, again with expert systems [31] or with artificial neural networks [14], to introduce AI to middle and high school pupils. Since then, AI has been developing rapidly, and it is crucial that its inclusion in education as a subject follows at least the same rhythm. There has been a growing interest in research proposing curricula to teach AI in schools. This paper is not focused on curricula, but on the design of activities to teach young pupils AI-related concepts. However, explaining these concepts to K-12 through traditional methodologies such as lectures and books is challenging [45]. Many researchers are developing numerous kinds of activities to overcome this challenge. However, what to teach learners, at what age, and how, are some of the open questions being explored.

The motivation of this paper is to offer researchers a global overview of the kinds of activities currently being developed to teach AI to young pupils. As detailed in [7], existing tools and resources to teach and learn computer science are not easy to find nor well advertised. The same observation applies for artificial intelligence, even if several reviews have been written. These latter are usually research-oriented and focused on a specific angle, missing the opportunity to bring a broader and more pragmatic vision. Several reviews and survey papers have already been realised by various researchers. Six pieces of research, published between 2020 and 2023, have been identified as roughly covering a similar goal as the present paper [26, 25, 34, 46, 10, 40]. The main differences are that the latter are either focused on a particular region of the world, or on a specific subfield of AI such as machine learning, or on a given age group.

1.2 Methodology

To fulfil the objectives of this paper, an extensive literature review has been conducted, following similar strategies to those used in [4]. Papers have been found on Google Scholar and on various widespread publishers, including ACM, IEEE, Springer and MDPI, using the following keywords on their search engine: “artificial intelligence K-12”, “machine learning K-12”, “children teaching artificial intelligence”, “activity to learn artificial intelligence children.” Relevant references of the papers found with those keywords have also been examined. Papers from the initial set have then been filtered out. Only those in English and published from the year 2000 have been kept. All the kinds of papers have been considered, whether they have been peer-reviewed or not and whatever type they are (full and short paper, poster, extended abstract, etc.). Then, only those related to the development of activities for children (up to 18 years old) have been used for this review. Some papers about the development of curricula have been filtered out, only those also containing propositions of activities have been kept.

After this introduction, the remainder of the paper is as follows. Section 2 presents the subfields of AI covered by the developed activities discovered by this review. Section 3 categorises them according to their kinds. Section 4 then summarises and discusses the findings. Finally, Section 5 concludes the paper.

2 Artificial intelligence subfields

The analysed papers reveal that many subfields of artificial intelligence (AI) are covered by activities to teach them. This section goes through them.

2.1 Data structure

Many AI algorithms are relying on specific *data structures*, like trees, graphs, forests and matrices. Going deep in their understanding is perhaps unsuitable for young children. However, it is worth teaching them about elementary AI algorithms since the explanations can be very visual thanks to the simple underlying data structure. For example, getting how some simple decision tree learning algorithms work is easy for people understanding the notion of tree. Not many pieces of research analysed for this review are focusing on the data structures used in AI. In [21], the authors present the development of unplugged activities to teach AI. The first one can be used to introduce the tree data structure, in the specific context of decision tree learning.

2.2 Learning

Nowadays, *machine learning* (ML) has become the new engine that revolutionises the practices of knowledge discovery [34]. As a consequence, it is important for everyone, in particular children, to be able to cope with the central paradigms of ML. The majority of recent research on activities to teach AI is

about ML. Some developed activities are trying to explain the core concepts and intuitions, while others are focused on describing the technical parts. It has also been shown that making children involved in the training of accessible ML systems support them to better understand basic ML processes [19]. Several activities related to ML are therefore focusing on the training part.

Decision tree learning It is important for K-12 pupils to learn about the core ideas and principles of ML. However, it is an enormous challenge for this age group to directly delve into the complexity of ML. Focusing of *decision tree learning* (DTL) provides a more suitable entry point. More precisely, it can be used to exemplify the idea of supervised learning. In [27], the authors propose a teaching concept starting with the understanding of decision tree learning, combining several activities and tools.

Supervised learning Many of the developed activities do include content to teach about *supervised learning* techniques or applications. This is likely due to the fact that the training process can be easily transposed into interactive activities. In [32], the authors present a game they developed to teach supervised learning, gradient descent and k -nearest neighbour classification. Their approach is limited to teaching the definition and core concepts, without inner details like underlying mathematics or jargon. In [29], the authors report on the design of an activity using block-based programming to control educational robots to introduce supervised, unsupervised and reinforcement learning. In [43], the authors are focused on teaching the insights of image recognition and supervised learning to very young pupils, with direct demonstration of how a classifier can identify objects they drew. Finally, in [16], the activity presented by the authors is the development of a scavenger hunt game run on a smartphone. The application relies on machine learning to perform image recognition to identify whether the correct object has been detected or not.

Neural network learning Activities specifically dedicated to *neural network* (NN) learning are also being developed by researchers. The interest about NN is that explaining them can easily be done visually, which makes it more suitable for younger. In [36], the authors propose a three-part learning module for pupils to be taught about artificial neural networks. A constructionist approach is followed, leading pupils to first use neural networks, then modify predefined ones and finally construct their own. This module is carried on a programmable learning environment based on Scratch programming.

2.3 Data mining

Since the rise of big data, it is interesting for people to grasp how algorithms can extract valuable knowledge from them. *Data mining* (DM) is another subfield of AI that can be taught to young pupils. Other subjects like privacy issues can

also be discussed in relation with DM. In [11], the authors develop a learning module based on the RapidMiner tool to provide an introduction to DM to young pupils. Their module includes a Hollywood Movie Recommendation activity to make learners understand how to collect, analyse and use data.

2.4 Data science

Analysing data to make decisions with data analytics and machine learning is becoming a widespread activity in the industry. It is therefore also important to teach pupils about what is *data science* and decisions that can be made based on the results of data analyses. Understanding data-driven intelligence is consequently an interesting competency for today's youth to acquire. In [37], the authors present a half-day camp tutorial in which they expose pupils to the full cycle of a typical supervised learning approach used for data analyses. They designed the tutorial as an exciting hands-on introduction to data science.

3 Activities and tools

A second analysis that has been performed on the analysed papers made it possible to highlight the kinds of activities that have been or are being developed, and in which context they are organised.

3.1 Activities

Various kinds of activities are being developed to teach pupils AI-related concepts. Some of them rely on software systems, others on tangible devices, and still others only on pen and paper.

Programming AI concepts can be taught through *programming*. Several activities have been developed where pupils have to experiment by themselves, creating or configuring an AI model. In [29], the authors develop activities for pupils to learn ML concepts by writing programs with a block-based programming tool. Following constructionism ideas, making them programming helps them to experience the concepts in practice. In [16], the authors explain how they designed an activity where pupils are asked to develop a smartphone-based game with AppInventor. In [13], the authors present a workshop using Scratch programming to teach pupils data clustering and artificial NN learning. Their idea is to get learners to partially code AI algorithms to make them aware of how intelligent systems work through construction and experimentation.

Game *Games* have been used for educational purposes for decades, as they contribute to increase their players' motivation to learn [30, 5]. In particular, they are a good mean to have pupils learn about programming, but it is also true for AI. There are various kinds of games that can be used, including tangible,

computer and video games. Teaching concepts through them makes learning fun without overwhelming learners with inner details of the concept. In [35], the authors present a prototype tangible toy pupils can play with to understand some basic concepts on ML and on the internet of things. The toy consists of two cubes, one being a sensor and the other one an actuator, both communicating the MQTT protocol. Computer and video games are also being used. For example, *ML-Quest* is a 3D video game with a quest theme designed to teach the definition and working of three concepts of ML [32]. Each level of the game ends with the definition of a concept and a mapping of it with the task performed by the player in the level. The authors tested the game with higher-secondary pupils, the majority considering that playing it contributed to enhance their understanding of machine learning. Programming can be learned on online game platforms, such as *Leek Wars*, which is focused on writing an IA to win a fight [5].

Robotics As it is the case with programming, programmable robots are being used to teach AI, should they be physical or virtual. In [47], the authors reports on an activity where pupils are taught about reinforcement learning by building controllers for both physical and virtual robots. The physical robot comes from the LEGO SPIKE Prime robotics kit, and the virtual one was from a web-based platform developed by the authors. In [29], pupils are writing programs to control virtual educational robots in the Open Roberta Lab.

Device AI is being embedded in many *devices* of everyday life, or at least accessible to the public at large. Therefore, it is relevant to develop activities manipulating similar intelligent devices. Direct experience with physical objects can facilitate the understanding of abstract concepts [48]. In [19], the authors present an activity based on the manipulation of a digital stick-like device. Pupils are asked to produce gestures and label them, to train an ML model. Thanks to this activity, pupils learn the two core concepts of ML that are data labelling and evaluation. Other approaches based on sticks are being developed [1].

Unplugged Activities not requiring any technology to be run, referred to as *unplugged activities*, are a common way to teach computer science. They have become valuable for many reasons: low cost, ease of implementation, possibly playful, possibly relying on physical interaction, easily deployable, etc. Unplugged activities served as a low-barrier entry to the topic [32]. In the context of AI, they can be used to broaden the access for educators to AI-related learning experiences at a lower cost. Such activities are also usually more engaging for novice audiences, which is often the case for AI in schools. For example, in [22], the authors present two unplugged assignments to broadly understand AI on the one hand and to be introduced to knowledge representation and reasoning on the other hand. In [44], the author presents an activity where pupils are physically acting out different generative adversarial networks. The goal of the activity is to lead their participants to sketch realistic-looking fake images. Until now, approaches to make AI tangible for students without actually programming an AI

system have been rare. AI Unplugged provides unplugged activities presenting AI ideas and concepts without using computers [21].

3.2 Tools

Researchers are implementing *tools* based on which activities can be developed to teach AI. For example, *Google Teachable Machine* is used to design workshops where pupils are creating their own ML application [42, 12]. *Teachable machine* refers to interfaces that do not require programming but makes it possible for its users to train and test an algorithm iteratively [12]. In [39], the authors test an approach using several tools to teach machine learning through design fiction. They used Scroobly (an AR tool), Teachable Machine and Adacraft (a Scratch-based coding environment compatible with ML extension blocks).

3.3 Contexts

The aforementioned activities can be organised in diverse *contexts* where pupils are learning, which can either be in schools or outside of them. This section presents various contexts where the presented activities can be organised.

Course One possible approach to teach artificial intelligence is to develop a teaching unit that includes both theoretical and hands-on components. For example, in [3], the authors present an AI course called *IRobot* and that covers major topics of artificial intelligence. Their goal is to have this seven weekly teaching units of two hours course integrated in secondary science education.

Competition Competitions are a motivating and challenging way to teach concepts, in particular to young people [9]. They are often used to learn programming skills, for example as online game platforms [5]. One example of a challenge targeted to young pupils and through which AI concepts can be taught is the *Bebras Challenge* [8, 15]. Other competitions include games such as Leek Wars [5], where players have to implement an intelligent behaviour for their leeks.

Event Organising *workshops* is also an interested way to teach AI. They can either be organised with pupils in the classroom or at events external to the school context, depending on the workshop total duration. For example, in [42] the authors propose a workshop based on Google Teachable Machine to teach machine learning principle to primary school pupils with three 2.5 hours-session workshop spread over three days. In [43], the authors report on a workshop they designed and held in SciFest, the largest science fair for children, in Finland. This 15-minute workshop was open to any visitor, without needing to book a time slot nor a seat. It consists of an activity where children were drawing animals and then presented them to an image recognition system to identify the drawn animal. The goal of the workshop is to teach basic concepts related to image recognition and supervised learning. Other kinds of event can be thought of to teach AI, like escape games or rooms, for example [6].

4 Discussion

The conducted review shows many different kinds of activities have been or are being developed to teach artificial intelligence (AI) to young pupils, covering several subfields of AI. Without surprise, machine learning (ML) is the most popular subfield. It is probably due to the fact that ML is a large part of modern AI. This popularity is also probably related to the fact that ML is the most used subfield in broad applications available to the public at large, and in particular young people. Data structure is not of direct interest for pupils as a broad subject, but it is an easy and possibly visual way to introduce AI-related concepts. Data mining and data science related activities are not very common, possibly because they may require basic AI knowledge beforehand.

Regarding the activities, the most common ones are related to programming, either with physical objects or in virtual simulation environments. Programming simple AIs to control agents in games is also quite popular. Unplugged activities are also developed, more specifically for younger pupils. The main advantage is that they can be more easily organised, without specific equipments. Competitions and games to teach AI are also quite popular, as they are very motivating ways to learn. All these activities can be organised in several contexts, courses offering the longest training time. Short workshops are also interesting since they can put pupils into action during a small amount of time, keeping them focused on the activity. Of course, other contexts may be explored, such as summer camps and other trainings outside of schools.

5 Conclusions

To conclude, the review presented in this paper covers recent pieces of research related to the development of activities to teach artificial intelligence (AI) to young pupils. This paper reveals both the subfields of AI that can be taught and the different kinds of activities used to teach them.

Future work includes refining the analysis of the existing activities to take into account their targeted age groups. Further research should also be conducted to identify whether some activities are best-suited for a given subfield of AI, or a given age group, when used in a specific context.

References

1. Agassi, A., Erel, H., Wald, I.Y., Zuckerman, O.: Scratch nodes ML: A playful system for children to create gesture recognition classifiers. In: 2019 CHI Conference on Human Factors in Computing Systems (2019). <https://doi.org/10.1145/3290607.3312894>
2. Broering, A., Niedermeier, C., Olaru, I., Schopp, U., Telschig, K., Villnow, M.: Toward embodied intelligence: Smart things on the rise. *Computer* **54**, 57–68 (Jul 2021). <https://doi.org/10.1109/MC.2021.3074749>

3. Burgsteiner, H., Kandlhofer, M., Steinbauer, G.: IRobot: Teaching the basics of artificial intelligence in high schools. *Proceedings of the AAAI Conference on Artificial Intelligence* **30**(1) (2016). <https://doi.org/10.1609/aaai.v30i1.9864>
4. Combéfis, S.: Automated code assessment for education: Review, classification and perspectives on techniques and tools. *Software* **1**, 1–28 (2022). <https://doi.org/10.3390/software1010002>
5. Combéfis, S., Beresnevičius, G., Dagienė, V.: Learning programming through games and contests: Overview, characterisation and discussion. *Olympiads in Informatics* **10**, 39–60 (2016). <https://doi.org/10.15388/oi.2016.03>
6. Combéfis, S., de Moffarts, G.: Learning computer science at a fair with an escape game. In: 12th International Conference on Informatics in Schools: Situation, Evolution and Perspectives. pp. 93–95 (2019)
7. Combéfis, S., de Moffarts, G., Jovanov, M.: TLCs: A digital library with resources to teach and learn computer science. *Olympiads in Informatics* **13**, 3–20 (2019). <https://doi.org/10.15388/oi.2019.01>
8. Combéfis, S., Stupurienė, G.: Bebras based activities for computer science education: Review and perspectives. In: 13th International Conference on Informatics in School: Situation, Evaluation, Problems. pp. 15–29 (2020). https://doi.org/10.1007/978-3-030-63212-0_2
9. Combéfis, S., Wautelet, J.: Programming trainings and informatics teaching through online contests. *Olympiads in Informatics* **8**, 21–34 (2014)
10. Druga, S., Otero, N., Ko, A.J.: The landscape of teaching resources for ai education. In: 27th ACM Conference on on Innovation and Technology in Computer Science Education. pp. 96–102 (2022). <https://doi.org/10.1145/3502718.3524782>
11. Dryer, A., Walia, N., Chattopadhyay, A.: A middle-school module for introducing data-mining, big-data, ethics and privacy using rapidminer and a hollywood theme. In: 49th ACM Technical Symposium on Computer Science Education. pp. 753–758 (2018). <https://doi.org/10.1145/3159450.3159553>
12. Dwivedi, U.: Introducing children to machine learning through machine teaching. In: *Interaction Design and Children*. pp. 641–643 (2021). <https://doi.org/10.1145/3459990.3463394>
13. Estevez, J., Garate, G., Graña, Jr., M.: Gentle introduction to artificial intelligence for high-school students using scratch. *IEEE Access* **7**, 179027–179036 (2019). <https://doi.org/10.1109/ACCESS.2019.2956136>
14. Fok, S.C., Ong, E.K.: A high school project on artificial intelligence in robotics. *Artificial Intelligence in Engineering* **10**(1), 61–70 (1996). [https://doi.org/10.1016/0954-1810\(95\)00016-X](https://doi.org/10.1016/0954-1810(95)00016-X)
15. Futschek, G., Dagienė, V.: A contest on informatics and computer fluency attracts school students to learn basic technology concepts. In: 9th World Conference on Computers in Education (2009)
16. Guerreiro-Santalla, S., Mallo, A., Baamonde, T., Bellas, F.: Smartphone-based game development to introduce K12 students in applied artificial intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence* **36**(11), 12758–12765 (2022). <https://doi.org/10.1609/aaai.v36i11.21554>
17. Haenlein, M., Kaplan, A.: A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review* **61**(4), 5–14 (2019)
18. Hitron, T., Orlev, Y., Wald, I., Shamir, A., Erel, H., Zuckerman, O.: Can children understand machine learning concepts?: The effect of uncovering black boxes. In: 2019 CHI Conference on Human Factors in Computing Systems. pp. 1–11 (2019). <https://doi.org/10.1145/3290605.3300645>

19. Hitron, T., Wald, I., Erel, H., Zuckerman, O.: Introducing children to machine learning concepts through hands-on experience. In: 17th ACM Conference on Interaction Design and Children. pp. 563–568 (2018). <https://doi.org/10.1145/3202185.3210776>
20. Kandlhofer, M., Steinbauer, G., Hirschmugl-Gaisch, S., Huber, P.: Artificial intelligence and computer science in education: From kindergarten to university. In: 2016 IEEE Frontiers in Education Conference (2022). <https://doi.org/10.1109/FIE.2016.7757570>
21. Lindner, A., Seegerer, S., Romeike, R.: Unplugged activities in the context of AI. In: 12th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives. pp. 123–135 (2019). https://doi.org/10.1007/978-3-030-33759-9_10
22. Long, D., Moon, J., Magerko, B.: Unplugged assignments for K-12 AI education. *AI Matters* **7**(1), 10–12 (2021). <https://doi.org/10.1145/3465074.3465078>
23. Makridakis, S.: The forthcoming artificial intelligence (AI) revolution: Its impact on society and firms. *Futures* **90**, 46–60 (Jun 2017). <https://doi.org/10.1016/j.futures.2017.03.006>
24. Mariescu-Istodor, R., Jormanainen, I.: Machine learning for high school students. In: 19th Koli Calling International Conference on Computing Education Research (2019)
25. Marques, L.S., Gresse Von Wangenheim, C., Hauck, J.C.R.: Teaching machine learning in school: A systematic mapping of the state of the art. *Informatics in Education* **19**(2), 283–321 (2020)
26. Martins, R.M., Gresse Von Wangenheim, C.: Findings on teaching machine learning in high school: A ten-year systematic literature review. *Informatics in Education* (2023)
27. Michaeli, T., Seegerer, S., Kerber, L., Romeike, R.: Data, trees, and forests – decision tree learning in K-12 education. In: 3rd Teaching Machine Learning Workshop (2022)
28. Müller, H.A.: The rise of intelligent cyber-physical systems. *Computer* **50**, 7–9 (Dec 2017). <https://doi.org/10.1109/MC.2017.4451221>
29. Olari, V., Cvejovski, K., Eide, O.: Introduction to machine learning with robots and playful learning. *Proceedings of the AAAI Conference on Artificial Intelligence* **35**(17), 15630–15639 (2021). <https://doi.org/10.1609/aaai.v35i17.17841>
30. Overmars, M.: Teaching computer science through game design. *Computer* **37**(4), 81–83 (2004)
31. Pilgrim, R.A.: TIC-TAC-TOE: Introducing expert systems to middle school students. *ACM SIGCSE Bulletin* **27**(1), 340–344 (1995). <https://doi.org/10.1145/199691.199853>
32. Priya, S., Bhadra, S., Chimalakonda, S., Venigalla, A.S.M.: ML-Quest: a game for introducing machine learning concepts to K-12 students. *Interactive Learning Environments* (2022). <https://doi.org/10.1080/10494820.2022.2084115>
33. Reynolds, C.F.: Introducing expert systems to pupils. *Journal of Computer Assisted Learning* **4**(2), 79–92 (1988). <https://doi.org/10.1111/j.1365-2729.1988.tb00268.x>
34. Sanusi, I.T., Oyelere, S.S., Vartiainen, H., Suhonen, J., Tukiainen, M.: A systematic review of teaching and learning machine learning in K-12 education. *Education and Information Technologies* (2022)
35. Scheidt, A., Pulver, T.: Any-cubes: A children’s toy for learning AI: Enhanced play with deep learning and MQTT. In: *Mensch und Computer 2019*. pp. 893–895 (2019). <https://doi.org/10.1145/3340764.3345375>

36. Shamir, G., Levin, I.: Neural network construction practices in elementary school. *KI-Künstliche Intelligenz* **35**, 181–189 (2021). <https://doi.org/10.1007/s13218-021-00729-3>
37. Srikant, S., Aggarwal, V.: Introducing data science to school kids. In: 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 561–566 (2017). <https://doi.org/10.1145/3017680.3017717>
38. Su, J., Zhong, Y.: Artificial intelligence (ai) in early childhood education: Curriculum design and future directions. *Computers and Education: Artificial Intelligence* **3** (2022). <https://doi.org/10.1016/j.caeai.2022.100072>
39. Tamashiro, M.A., Van Mechelen, M., Schaper, M.M., Iversen, O.S.: Introducing teenagers to machine learning through design fiction: An exploratory case study. In: *Interaction Design and Children*. pp. 471–475 (2021). <https://doi.org/10.1145/3459990.3465193>
40. Tedre, M., Toivonen, T., Kahila, J., Vartiainen, H., Valtonen, T., Jormanainen, I., Pears, A.: Teaching machine learning in K–12 classroom: Pedagogical and technological trajectories for artificial intelligence education. *IEEE Access* **9**, 2169–3536 (2021). <https://doi.org/10.1109/ACCESS.2021.3097962>
41. The Royal Society: *Machine Learning: the Power and Promise of Computers that Learn by Example*. The Royal Society, Great Britain (2017)
42. Toivonen, T., Jormanainen, I., Kahila, J., Tedre, M., Valtonen, T., Vartiainen, H.: Co-designing machine learning apps in K-12 with primary school children. In: 2020 IEEE 20th International Conference on Advanced Learning Technologies. pp. 308–310 (2020). <https://doi.org/10.1109/ICALT49669.2020.00099>
43. Toivonen, T., Jormanainen, I., Tedre, M., Mariescu-Istodor, R., Valtonen, T., Vartiainen, H., Kahila, J.: Interacting by drawing: Introducing machine learning ideas to children at a K-9 science fair. In: 2022 CHI Conference on Human Factors in Computing Systems. pp. 1–5 (2022). <https://doi.org/10.1145/3491101.3503574>
44. Virtue, P.: GANs unplugged. *Proceedings of the AAAI Conference on Artificial Intelligence* **35**(17), 15664–15668 (2021). <https://doi.org/10.1609/aaai.v35i17.17845>
45. Yang, W.: Artificial intelligence education for young children: Why, what, and how in curriculum design and implementation. *Computers and Education: Artificial Intelligence* **3** (2022). <https://doi.org/10.1016/j.caeai.2022.100061>
46. Yue, M., Jong, M.S.Y., Dai, Y.: Pedagogical design of K-12 artificial intelligence education: A systematic review. *Sustainability* **14**(23) (2022). <https://doi.org/10.3390/su142315620>
47. Zhang, Z., Willner-Giwerc, S., Sinapov, J., Cross, J., Rogers, C.: An interactive robot platform for introducing reinforcement learning to K-12 students. In: *International Conference on Robotics in Education*. pp. 288–301 (2022). https://doi.org/10.1007/978-3-030-82544-7_27
48. Zuckerman, O.: Designing digital objects for learning: Lessons from froebel and montessori. *International Journal of Arts and Technology* **3**(1), 124–135 (2009). <https://doi.org/10.1504/IJART.2010.030497>

Breaking Gender Barriers in Computer Science: Exploring the Impact of Digital Fabrication Workshops in Smart Environments

Nadine Dittert¹[0000-0001-7735-7823] Mareike Daeglau²[0009-0007-4599-698X],
Nils Pancratz³[0000-0001-7358-4148], and Ira Diethelm¹[0000-0002-5586-8566]

¹ Carl von Ossietzky Universität, Abt. Didaktik der Informatik, 26111 Oldenburg

² Carl von Ossietzky Universität, Neuropsychology Lab, 26111 Oldenburg
firstname.lastname@uol.de

³ University of Hildesheim, Universitätspl. 1, 31141 Hildesheim,
pancratz@imai.uni-hildesheim.de

Abstract. Gender barriers in Computer Science (CS) are undeniable. The lack of girls and (young) women participating in CS is unacceptable in several ways. Reasons to explain this gap are given in the literature: First, CS has a reputation as a heavily technology-focused subject with no particular social or practical application references. But in particular, also computer scientists are highly stereotyped and these clichés often are at odds with girls' and young women's self-concepts. These causes yield targets for interventions through extracurricular activities that are presented in this paper.

In total, 90 digital fabrication workshops were evaluated through pre-post-questionnaires to develop a richer understanding of the effect of digital fabrication workshops in smart environments designed to reduce biases and attract girls and young women aged 9 to 18 to CS. In particular, the change of the participants' image of CS in general, their self-efficacy in dealing with technology, the perceived social relevance of CS, and the perceived relevance of CS for later professional life were investigated. Our results indicate that digital fabrication workshops can indeed contribute both to a richer understanding of what CS is and to a small increase in technic related self-efficacy. The findings also provide a basis for deriving implications for the realization of Informatics or CS as a subject matter in primary and secondary education.

Keywords: Computer Science Education · Participation · Gender · Digital Fabrication · Smart Environments

1 Introduction

Computer Science Education (CSE) in schools is handled as diverse as possible in Germany: it ranges from compulsory subject to elective subject to no subject. Initiatives like the Informatics for All Coalition claim for informatics as a foundational discipline in schools across Europe⁴ and also various German committees

⁴ <https://www.informaticsforall.org/>, retrieved June 6th, 2023

are eager to promote CS as a compulsory subject throughout the country⁵. A very strong argument why CS should be a compulsory subject in schools is to empower everyone to actively participate in today's digitalized world [10, 17]. Moreover, it has been stated that this point is crucial for female participation which addresses a very important aspect in today's society [5].

Today's western world is widely driven by technology in almost every area of life: traffic, policies, journalism, culture, and also personal life that includes well-being, security or organizing. Hence, it is important to give women and girls a voice in technology development. Past product developments have shown that if women are not involved in designing them, products risk to fail at addressing women's needs. This can be annoying, derogatory, or even dangerous for women and diverse other people [18, 26]. It is when a homogeneous group of people is delegated to design for the future that diverse needs risk being ignored.

Accordingly, a compulsory subject could be an important step to more female participation in CS. Nevertheless, it can be taught in diverse ways. Our ideas of modern CSE include digital fabrication activities as they offer the possibility to 'do' CS while relating to the discipline and its applications in a hands-on activity. In our project, we focused on digital fabrication activities for girls to find out what aspects might contribute to more female participation in CS.

In this paper we present reasons for the lack of female participation in CS and how digital fabrication might address them. We then describe workshop activities that were designed, conducted and evaluated with respect to attract girls to CS. Results regarding the image of CS, girl's self-efficacy, and the perceived relevance of CS are presented and discussed. We conclude with arguments for digital fabrication as part of compulsory CSE at schools.

2 Background

In Europe we face the problem that only about 20% of all undergraduate CS students are female [27]. These numbers vary depending on the level of education (bachelor or master) and also on the European country. In German senior classes in CS only 16% of school students are female [11]. Regarding the question of why there are so few women in CS there are many different explanations that cannot be regarded separately but as a combination of many aspects. Vainionpää et al. present aspects that affect girls' career choices regarding IT that include influences by people in the girls' environment as well as historical issues such as stereotypes and images of the field [28]. Further studies confirmed that it is partly the girls' personal environment that prevents girls from choosing a career in IT or that it is more the image of people than doing CS that hampers more diversity in the field [12, 29]. Another important aspect when it comes to girls and their perception of CS is self-efficacy [2, 8]. While boys tend to overestimate themselves, girls undervalue their abilities which results in girls being less confident in their skills and enjoying CS activities less than boys [24].

⁵ <https://www.shetransformsit.org/>, retrieved June 6th, 2023

However, studies reveal that girls perform better in computer and information literacy tasks and also that they show higher CS abstraction abilities after a CS introductory course than boys [9, 23]. Nevertheless, another study shows that there is a significant difference in abilities between boys and girls when there is no compulsory CSE in schools [25].

Hence, the question arises how to create a CS environment that is more appealing to girls than it has been so far. Therefore, in diverse projects digital fabrication activities have been used to raise more interest in STEM, especially technology, in a creative and informal way. Here, we refer to digital fabrication as creative interdisciplinary construction activity with physical computing kits that comprise the design, building (constructing) and programming of smart objects using hardware (controllers, sensors, actuators), desktop programming environments and crafting materials including 3D-printers and laser-cutters and that allow for creativity and personally meaningful projects.

Digital fabrication activities, such as creating personally meaningful smart textile artifacts show potentials to raise girls' self-efficacy in dealing with technology [16]. Further, learning activities with e-textiles combine traditional crafting practices with CS and show potential to broaden participation in IT [13, 4]. Generally, physical computing toolkits have been utilized to introduce CS concepts in a meaningful, hands-on way for a long time [3]. Relating computing activities to participants' everyday life is another crucial point that can be implemented in digital fabrication activities [17]. Furthermore, physical computing adds state-of-the-art topics to the CS curriculum, such as sensing and control or ubiquitous and embedded systems [20].

3 Breaking gender barriers in CS: The SMILE project

The SMILE project aimed at enhancing female participation in CS by cultivating a positive image of CS through digital fabrication workshops. By means of the innovative topic of smart environments and a coordinated didactic concept, in digital fabrication workshops girls and young women from the age of nine to 18 were encouraged to create their own intelligent environmental artifacts, such as smart pillows, lights, closets, plants, or magical rooms and houses. The project was conducted over three years in three cities in Germany, namely Hamburg, Bremen, and Oldenburg. Construction of artifacts included programming microcontrollers, i.e. Arduino (Lily Pad), Calliope mini, ESP8266, connection of sensors and actuators if necessary, and integration into objects such as pillows or plants. In most of the workshops, additional digital fabrication tools were used, e.g. to 3D-print artificial flowers [19] or to create personally designed smart pillows using vinyl-cutters [6].

Topics that are appealing to girls within the scope of smart environments were worked out in pre-workshops [14] and chosen according to age, previous knowledge, and lab facilities. Each workshop involved 12-15 girls or young women, comprised up to 20 hours, and was accompanied by two to three facilitators. Workshops took place in university settings in living labs, university FabLabs,

or alike and were conducted by seven different partners. Drawing from previous studies and research, the overall workshop concept incorporated the highlighting of specific aspects of CS during the interventions, including an introduction to the fundamentals of CS as well as the practical relevance of CS in our everyday lives. If possible and suitable, research labs (mostly ambient assisted living labs or living places) were visited to better illustrate what CS research is about and how it relates to real life.

Preliminary findings indicated a positive impact of digital fabrication workshops on girls' attitudes of CS in terms of stereotypes, scope, and its presence. In particular, participating in a digital fabrication workshop made the field appear more female and more ubiquitous than before [15] and resulted in higher confidence regarding programming skills [22]. In the present study, data from all 90 workshops carried out over the project period of three years is analyzed, providing a comprehensive overview compared to previous partial studies.

For the evaluation of the project, participants completed computer-based questionnaires at the beginning and the end of each workshop. Since answering questions was optional, the number of included participants in the pre-/post-comparison differs between these items. Additionally, the data evaluated fulfilled the following two conditions: the participants were taking part in the workshop for the first time and the specific questions were answered at both measuring points.

To test for pre-/post-changes in the items image of CS in general, self-efficacy in dealing with technology, social relevance of CS and relevance of CS for later professional life, paired sample t-tests were conducted. Effect sizes were calculated according to Cohen's d as implemented in the R-package "effsize". To test whether changes between pre- and post-measurement points were related to the participants age, attendance of CS classes in school or parental background, Spearman correlations were conducted between those ratings and pre-/post-changes in the respective variables of interest. Sample sizes differ from 262 to 500 between the tests, because only complete cases were included into the analyses.

4 Results

Within the scope of this paper, we focus on the following concepts and questions, which were surveyed in the pre- and post-questionnaire: image of CS in general, self efficacy in dealing with technology, social relevance of CS, and relevance of CS for later professional life.

4.1 Image of CS in general

The following visual analog scale⁶ statements have been combined to assess the image of CS in general:

⁶ As with the following items (cf. Sec. 4.3 and 4.4), the values of the sliders were mapped to a scale of 1-100 to ensure comparability.

Computer Science is ...
 ... *difficult/easy*
 ... *narrowly focused/just about anywhere*
 ... *boring/exciting*
 ... *uninteresting/interesting*
 ... *illogical/logical*
 ... *monotonous/diversified*

Higher values indicate a more positive image of CS.

To test whether the rating of the image of CS in general was higher in the post- compared to the pre-testing we performed a paired t-test. We found a significant effect (CS-image-pre: $M = 68.67$, $SD = 13.71$, CS-image-post: $M = 73.48$, $SD = 13.62$; $t_{1,261} = 7.15$, $p < .001$, $d = 0.4$). This indicates that the participant's image of CS in general was more positive in the post- compared to the pre-testing. Participant's views changed most on the statement that CS is just about anywhere (from 55.5/100 to 65.5/100). Moderate changes occurred on the questions on CS being exciting, interesting, and diversified, whereas the initial values were rather high (70/100 to 82/100) and increased by 3.8 to 5.9 points. Almost no change was registered on the question of CS being difficult or easy, whereas values pointed slightly more towards difficult (45.8/100 to 46.5/100).

A Spearman correlation did not reveal any significant association between pre-post changes of the image of CS in general and age ($\rho_{364} = 0.05$, $p = .78$).

To test whether participants' change of the image of CS in general from pre- to post-measurements was related to whether the girls attend CS classes at school, we performed a Spearman correlation, no significant association was obtained ($\rho_{365} = 0.02$, $p = .78$).

4.2 Self-efficacy in dealing with technology

To determine the girls' self-efficacy in dealing with technology, we used the questionnaire "Control beliefs in dealing with technology" (KUT) [1], which was adapted to the age of the girls and newer technologies (e.g. tablet instead of video recorder in the description; modified KUT). The following questions were queried to rate on a 5-point Likert scale and polarity reversed if necessary:

1. *I can solve quite a few of the technical problems I face on my own.*
2. *Technical devices are often complicated to operate and understand.*
3. *I really enjoy facing technical challenges.*
4. *Because I have never had problems with technology, this will also be the case in the future.*
5. *I feel so helpless in dealing with technical equipment that I don't even try it alone.*
6. *Even if I do not manage to operate a technical device immediately, I do not give up.*
7. *When I solve a technical task on my own, it is usually by chance.*
8. *Most technical tasks are so complicated that it makes no sense for me to deal with them on my own.*

Higher values indicate higher self-efficacy while 8 represents the lowest and 40 the highest possible scoring.

To test whether the self-efficacy in dealing with technology (modified KUT) scoring was higher in the post- compared to the pre-testing, we performed a paired t-test. We found a significant effect (KUT-pre: $M = 28.02$, $SD = 4.8$, KUT-post: $M = 28.42$, $SD = 5.03$; $t_{1,499} = 2.22$, $p = .03$, $d = 0.1$). In other words, our data provide evidence that the participant's self-efficacy in dealing with technology were slightly higher in the post- compared to the pre-testing.

A Spearman correlation revealed a significant positive association between pre-post changes of self-efficacy in dealing with technology and age ($\rho_{498} = 0.12$, $p = .009$). That is, our data indicate that a stronger positive gain in self-efficacy from pre- to post-measurement can be found in older participants compared to the younger ones.

A Spearman correlation showed a significant positive association between pre-post changes of self-efficacy in dealing with technology and whether the girls attend CS classes at school ($\rho_{498} = 0.13$, $p = .009$). That is, our data indicate that participants who state to attend CS classes in school have a stronger positive gain in self-efficacy from pre- to post-measurement.

A significant negative association between pre-post changes of self-efficacy in dealing with technology and pre-scores regarding parental technology communication and encouragement ($\rho_{495} = -0.10$, $p = .009$) was indicated by a Spearman correlation. That is, our data indicate that participants who hardly communicate about technical related topics with their parents and experience little encouragement to occupy oneself with technology have a stronger positive gain in self-efficacy from pre- to post-measurement.

4.3 Perceived social relevance of CS

To investigate how relevant girls consider CS for society, we asked them to rate the following statement on a visual analog scale with higher values indicating higher relevance:

*Computer Science is ...
... unimportant for society/important for society*

To test whether participants rated CS as more socially relevant in the post-compared to the pre-testing, we performed a paired t-test. No significant effect was obtained (soci-relevance-pre: $M = 76.67$, $SD = 22.10$, soci-relevance-post: $M = 75.74$, $SD = 26.50$; $t_{1,432} = 0.71$, $p = .50$, $d = 0.03$).

To test whether participants' change of the social relevance of CS from pre- to post-measurements was related to the girls' age, we performed a Spearman correlation, no significant association was obtained ($\rho_{431} = 0.02$, $p = .72$).

A Spearman correlation did not reveal any significant association between pre-post changes of the social relevance of CS and whether the girls attend CS classes at school ($\rho_{431} = 0.02$, $p = .67$).

4.4 Perceived relevance of CS for later professional life

To assess how relevant girls consider CS for their later professional life, we asked them to rate the following statement on a visual analog scale:

Computer Science is ...

... important for my future professional life/unimportant for my future professional life

Higher values indicate higher relevance.

To test whether participants rated CS as more relevant for their later professional life in the post- compared to the pre-testing, we performed a paired t-test. No significant effect was obtained (prof-relevance-pre: $M = 62.01$, $SD = 26.89$, prof-relevance-post: $M = 62.01$, $SD = 29.13$; $t_{1,405} = 0.02$, $p = .85$, $d < 0.001$).

A Spearman correlation was conducted between pre-post changes in ratings of relevance of CS for later professional life and the participant's age. We did not find a significant association ($\rho_{404} = 0.05$, $p = 0.29$).

To test whether participants' pre-post changes in ratings of relevance of CS for later professional life was related to whether the girls attend CS classes at school, we performed a Spearman correlation, no significant association was obtained ($\rho_{403} = 0.08$, $p = .11$).

A significant negative association between pre-post changes of rating relevance of CS for their later professional life and pre-scores regarding parental technology communication and encouragement ($\rho_{380} = -0.15$, $p = .003$) was indicated by a Spearman correlation. That is, our data indicate that participants who hardly communicate about technical related topics with their parents and experience little encouragement to occupy oneself with technology have a stronger positive gain regarding rating relevance of CS for their later professional life from pre- to post-measurement compared to the ones who communicate more with their parents at home.

5 Discussion

In the present study, we aimed to develop a richer understanding of the effect of digital fabrication workshops in smart environments designed to attract girls and young women aged 9 to 18 to CS. Specifically, we expected a change from pre- to post-measurement towards a more positive image of CS in general, higher self-efficacy scorings, and a higher assessment of the social relevance of CS as well as the relevance for their own later professional life. Results show a significant positive change for the image of CS in general and technology related self-efficacy but neither for the social relevance of CS nor for the relevance of CS for later professional life in general. Nevertheless, we argue for digital fabrication workshops as part of regular CS education as ways of engaging more girls into CS because of these findings which we will discuss here.

5.1 Discussion of Results

The positive change regarding the image of CS among the participating girls is in line with our previous research in this field and in this project [7, 15]. At this point, we can infer that the topic of smart environments is as convenient to get an insight into CS as topics from former studies that use other scenarios. We should keep in mind though that during the workshops it was further discussed what CS is and who is working in this field. Hence, we cannot say that it is merely creating smart environment objects that leads to a more positive image of CS. We argue that the whole workshop process needs to be related to CS processes, such as designing a new IT product as a solution for a specific kind of problem. Further, diverse people in the field should be mentioned including women, people of color, etc.

A closer look on the separate questions regarding the image of the field supports the general idea that digital fabrication workshops are a way to show the prevalence of CS in today's (western) world and its diversity. Further, it seems to be a way to highlight appealing aspects of CS. Anyhow, it also indicates that CS does (almost) not appear easier than before, which might be a realistic estimation of CS contents.

Self-efficacy among girls also raised after they participated in a digital fabrication workshop. This effect has been shown in previous research before (e.g. [7, 16, 21]) and has been proved here to be valid for smart environment settings as well. Self-efficacy is an important issue when it comes to girls and CS because research has also revealed that girls and women tend to underestimate themselves which is regarded as one of the reasons why they do not aim for a career in STEM related fields [28, 2]. Hence, we argue for CS activities that raise girls' self-efficacy in dealing with technology as much as possible and see clear benefits in digital fabrication workshops where girls are empowered to create their own smart object.

Most interestingly, in this study, we found correlations between self-efficacy and girls' family surroundings and whether they attend CS classes at school or not. First, our results indicate that self-efficacy raises more among girls from families that do not communicate about technology at home than their peers who do. One reason might be that the ones with parents who communicate about technology already have a better understanding about technology and their own relation to it than the ones who do not have the opportunity to talk about technology at home. Therefore, we argue that it would be advisable to run digital fabrication workshops (extensively) in schools so that every girl has the possibility to relate herself to technology and to raise her self-efficacy in dealing with technology. This could be a step towards more equal opportunities. Further, the ones who attend CS classes at school benefited more regarding their increase of self-efficacy than the ones without CS classes at schools. This effect might be rooted in the current style of CS classes which some of the girls reported as less interesting or only learning how to use IT instead of solving problems or creating new artifacts. Hence, we would suggest integrating digital fabrication into CS classes at school where possible.

Against our expectations, we did not find significant improvements regarding the social relevance of CS. We ascribe this to the rather high initial values (77/100) and that the participants were mostly aware of the relevance of CS in people's everyday life before participating in one of our digital fabrication workshops.

Finally, there were no general improvements regarding the relevance of CS for their own later professional life, but a small negative correlation to parents' communication and encouragement. Again, as a step towards more equal opportunities in terms of creating communication opportunities for girls who do not talk to their parents about technology, we argue for digital fabrication workshops in schools.

One reason for not finding general improvements regarding the relevance of CS for participant's own later professional life might be that the professional career of our participants is still rather far away (age M: 12.7, SD: 1.9). Another explanation might be that many of the objects that were built during the workshops did not have the character of a real-life technology or a serious functionality like the talking plants or the smart pillows. Although some of the objects targeted a real-life purpose such as smart bookshelves or book recommendation systems, the workshops might have been regarded more as "play" than as serious work. At this point, more research is needed on how to make use of digital fabrication workshops for opening up minds for possible career paths, especially for girls.

5.2 Limitations

Our study is limited in several aspects. First of all, we cannot clearly say that all our findings are only related to the workshops. Other impacts might have come from their surroundings during the runtime of the workshop as well, such as influences by their peers, parents or others, or even by extremely hot weather, which might have led to a loss or gain of motivation, etc. Further, the workshops were all short-term interventions and changes were measured between the beginning and the end of the workshop. We cannot say anything about long term effects of workshops on girls' mindsets and future behaviors. More research is needed here.

The survey was implemented as a pre-post-online-test, whereas girls started their workshop experience by filling out the first questionnaire. Tutors were aside for helping with rather technical questions, but they were instructed not to talk about contents in order to avoid influencing the participants. This way, some questions were skipped and we cannot assure that the girls understood all details of questions appropriately.

Further, all of the workshops that were conducted differ in their details. Generally, they all follow the same concept, but tutors, locations, technology, lab visits and topics differed within the workshops. Also, acquisition of the participating girls differed which leads to diverse motivational aspects. That is, we cannot say which detail exactly led to a specific change and which did not. Our

results rely on the wholistic concept of engaging girls in creating smart environment objects including relating it to CS. Impacts of specific details remain unknown.

6 Conclusion

Summing up, these mostly extracurricular digital fabrication activities have shown different positive effects for girls, that are interesting for CSE in general. First, it was shown that the activities led to a more positive attitude regarding the image of CS, which is very important when it comes to participation and learning. Therefore, we see potential of digital fabrication activities as part of CSE in schools, especially when the subject is perceived as abstract and boring by school students.

Second, it was confirmed that attending a digital fabrication workshop can lead to a higher self-efficacy. As self-efficacy increases even more with a higher age of the participants and in some states of Germany CS lessons start at an older age, again it might be beneficial to implement such activities at school, because even at an older age, self-efficacy can be addressed appropriately. Further, a higher increase was measured when participants attended CS lessons in schools. Hence, we argue for digital fabrication activities as part of CSE because it seems to be a reasonable methodological supplement to what has been part of participants' school lessons before.

Finally, from our results we infer that digital fabrication activities in a compulsory school subject contribute to more equal opportunities for girls and all school students. The ones who do not speak with their parents about technology show a higher increase of self-efficacy regarding technology and also the perceived relevance of computer skills for later professional life increase only in this case. That is, the activities offer valuable opportunities for later life, which would be less available for students whose parents do not speak about technology with them.

Overall, we argue that the compulsory school subject CS should contain digital fabrication activities in means of contemporary CSE. This 'doing' CS conveys an appropriate image of what CS is and allows for equal opportunities for all when it comes to participating in a digitally networked world.

References

1. Beier, G.: Kontrollüberzeugungen im Umgang mit Technik. Report *Psychologie* **24**(9), 684–693 (1999)
2. Beyer, S.: Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education* **24**(2-3), 153–192 (2014). <https://doi.org/10.1080/08993408.2014.963363>
3. Blikstein, P.: Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design. *Foundations and Trends® in Human-Computer Interaction* **9**(1), 1–68 (2015). <https://doi.org/10.1561/11000000057>

4. Buechley, L., Eisenberg, M., Catchen, J., Crockett, A.: The LilyPad Arduino: Using Computational Textiles to Investigate Engagement, Aesthetics, and Diversity in Computer Science Education. In: CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. p. 423–432. CHI '08, ACM, New York (2008). <https://doi.org/10.1145/1357054.1357123>
5. Commission, E.: Informatics education at school in Europe. Publications Office of the European Union (2022). <https://doi.org/doi/10.2797/268406>
6. Dittert, N., Katterfeldt, E.S.: Diversity in Digital Fabrication: Programming Personally Meaningful Textile Imprints. In: Proceedings of the Conference on Creativity and Making in Education. p. 112–113. FabLearn Europe'18, Association for Computing Machinery, New York (2018). <https://doi.org/10.1145/3213818.3219812>
7. Dittert, N., Wajda, K., Schelhowe, H.: Kreative Zugänge zur Informatik: Praxis und Evaluation von Technologie-Workshops für junge Menschen. Staats- und Universitätsbibliothek Bremen, Bremen (2016), <http://nbn-resolving.de/urn:nbn:de:gbv:46-00105551-11>
8. Doubé, W., Lang, C.: Gender and stereotypes in motivation to study computer programming for careers in multimedia. *Computer Science Education* **22**(1), 63–78 (2012). <https://doi.org/10.1080/08993408.2012.666038>
9. Fraillon, J., Ainley, J., Schulz, W., Friedman, T., Duckworth, D.: Preparing for Life in a Digital World - IEA International Computer and Information Literacy Study 2018 International Report. Springer Nature, Singapore (2020). <https://doi.org/10.1007/978-3-030-38781-5>
10. Gallenbacher, J.: Ohne Informatik keine Allgemeinbildung. *Informatik Spektrum* **42**(2), 88–96 (2019)
11. de Groot, M., Riemann, V., Schwarze, B., Struwe, U.: Mädchen und Frauen in die Informatik: Aktivierungspotenziale und Erfolgsfaktoren Handlungsempfehlungen Bildung. Kompetenzzentrum Technik-Diversity-Chancengleichheit e. V., Bielefeld (2023)
12. von Hellens, L., Clayton, K., Beekhuyzen, J.P., Nielsen, S.H.: Perceptions of ICT Careers in German Schools: An Exploratory Study. *Journal of Information Technology Education* **8**, 211–228 (2009). <https://doi.org/10.28945/168>
13. Kafai, Y.B., Lee, E., Searle, K., Fields, D., Kaplan, E., Lui, D.: A Crafts-Oriented Approach to Computing in High School: Introducing Computational Concepts, Practices, and Perspectives with Electronic Textiles. *ACM Trans. Comput. Educ.* **14**(1) (3 2014). <https://doi.org/10.1145/2576874>
14. Katterfeldt, E.S., Dittert, N.: Co-Designing Smart Home Maker Workshops with Girls. In: Proceedings of the Conference on Creativity and Making in Education. p. 100–101. FabLearn Europe'18, Association for Computing Machinery, New York (2018). <https://doi.org/10.1145/3213818.3213833>
15. Katterfeldt, E.S., Dittert, N., Ghose, S., Bernin, A., Daeglau, M.: Effects of Physical Computing Workshops on Girls' Attitudes towards Computer Science. In: Proceedings of the FabLearn Europe 2019 Conference. FabLearn Europe '19, Association for Computing Machinery, New York (2019). <https://doi.org/10.1145/3335055.3335066>
16. Katterfeldt, E.S., Dittert, N., Schelhowe, H.: EduWear: Smart Textiles as Ways of Relating Computing Technology to Everyday Life. In: IDC '09: Proceedings of the 8th International Conference on Interaction Design and Children. p. 9–17. IDC '09, Association for Computing Machinery, New York (2009). <https://doi.org/10.1145/1551788.1551791>

17. Katterfeldt, E.S., Dittert, N., Schelhowe, H.: Designing digital fabrication learning environments for Bildung: Implications from ten years of physical computing workshops. *International Journal of Child-Computer Interaction* **5**, 3–10 (2015). <https://doi.org/10.1016/j.ijcci.2015.08.001>, digital Fabrication in Education
18. Nicol, A., Casey, C., MacFarlane, S.: Children are ready for speech technology – but is the technology ready for them. In: Bekker, M., Markopoulos, P., Kersten-Tsalkalkina, M. (eds.) *Interaction design and children: Proceedings of the International Workshop "Interaction design and children"*, August 28–29, 2002, Eindhoven, The Netherlands. Shaker-Verlag, Maastricht (2002)
19. Pancratz, N., Fandrich, A., Chytas, C., Daeglau, M., Diethelm, I.: Blöcke, Blumen, Mikrocontroller und das Internet of Things. In: Pasternak, A. (ed.) *Informatik für alle*. pp. 295–304. Gesellschaft für Informatik, Bonn (2019). <https://doi.org/10.18420/infos2019-c15>
20. Przybylla, M., Romeike, R.: Physical Computing and its Scope - Towards a Constructionist Computer Science Curriculum with Physical Computing. *Informatics in Education* **13**(2), 225–240 (2014). <https://doi.org/10.15388/infedu.2014.14>
21. Qiu, K., Buechley, L., Baafi, E., Dubow, W.: A Curriculum for Teaching Computer Science through Computational Textiles. In: *Proceedings of the 12th International Conference on Interaction Design and Children*. p. 20–27. IDC '13, Association for Computing Machinery, New York, NY, USA (2013). <https://doi.org/10.1145/2485760.2485787>
22. Seraj, M., Katterfeldt, E.S., Autexier, S., Drechsler, R.: Impacts of Creating Smart Everyday Objects on Young Female Students' Programming Skills and Attitudes, p. 1234–1240. Association for Computing Machinery, New York (2020). <https://doi.org/10.1145/3328778.3366841>
23. Statter, D., Armoni, M.: Learning Abstraction in Computer Science: A Gender Perspective. In: *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*. p. 5–14. WiPSCE '17, Association for Computing Machinery, New York, NY, USA (2017). <https://doi.org/10.1145/3137065.3137081>
24. Stoet, G., Geary, D.C.: The Gender-Equality Paradox in Science, Technology, Engineering, and Mathematics Education. *Psychological Science* **29**(4), 581–593 (2018). <https://doi.org/10.1177/0956797617741719>
25. Suessenbach, F., Schröder, E., Winde, M.: *Informatik für alle! Stifterverband für die Deutsche Wissenschaft e.V.*, Essen (2022)
26. Tatman, R.: Gender and Dialect Bias in YouTube's Automatic Captions. In: *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*. pp. 53–59. Association for Computational Linguistics, Valencia (2017). <https://doi.org/10.18653/v1/W17-1606>
27. Tikhonenko, S., Pereira, C.: *Informatics Education in Europe: Institutions, Degrees, Students, Positions, Salaries. Key Data 2012–2017. An Informatics Europe Report*. Tech. rep., Informatics Europe, Zurich (2018), <https://www.informatics-europe.org/component/phocadownload/category/10-reports.html?download=78:informatics-education-europe-data-2012-2017>
28. Vainionpää, F., Kinnula, M., Iivari, N., Molin-Juustila, T.: Girls' Choice – Why won't they pick IT? In: *Proceedings of the 27th European Conference on Information Systems (ECIS)*, Stockholm & Uppsala, Sweden, June 8–14, 2019. Association for Information Systems eLibrary, St. Petersburg, FL, US (2019), https://aisel.aisnet.org/ecis2019_rp/31
29. Wong, B.: 'I'm good, but not that good': digitally-skilled young people's identity in computing. *Computer Science Education* **26**(4), 299–317 (2016). <https://doi.org/10.1080/08993408.2017.1292604>

Easy Coding in Biology: Combining Block-Based Programming Tasks with Biological Education to Encourage Computational Thinking in Girls

Eva Schmidthaler¹[0000–0001–9633–8855],
Corinna Hörmann¹[0000–0002–4770–6217], David Hornsby¹[0000–0003–2080–5706],
Anneliese Fraser²[0009–0000–9400–0666], Martin Cápaj³[0000–0002–8352–0612], and
Barbara Sabitzer¹[0000–0002–1304–6863]

¹ Johannes Kepler University Linz, Altenbergerstraße 69, 4040 Linz, Austria
eva.schmidthaler@jku.at, corinna.hoermann@jku.at, david.hornsby@jku.at,
barbara.sabitzer@jku.at

² University of Passau, Innstraße 41, 94032 Passau, Germany
anneliese.fraser@uni-passau.de

³ Constantine Philosopher University, Trieda Andreja Hlinku 1, 949 74
Nitra-Chrenová, Slovakia mcapaj@ukf.sk

Abstract. This mixed-method research aims to address the gender gap in the Computer Science (CS) field by developing an interdisciplinary STEAM workshop combining practical elements with block-based programming (BBP) tasks on biological topics. The study presents two exploratory workshops called “Easy Coding in Biology” conducted with secondary school students in Slovakia and Austria. The workshops utilized a learning environment called <colette/>, which incorporates block-based coding and augmented reality (AR) features. Forty-seven (female = 23) 11-19-year-old students completed an evaluation questionnaire and observations were made during the workshops to improve the workshop content and process. Preliminary results indicate the potential of combining BBP with biological topics to promote computational thinking (CT) and CS skills in secondary school girls. Some participants faced challenges with the learning environment and programming language, especially younger students, and the use of loops. Adaptations are being made to cater to younger students in science education and include additional CT tasks and experiments. Future courses of the workshop series “Easy Coding in Science” (Physics, Chemistry, and Biology) will be conducted in other countries and with different BBP programs (e.g., MakeyMakey, OzoBlockly, and Micro:bit) in 2023 and 2024.

Keywords: Block-Based Programming · Biology · Coding · Education · Computational Thinking · STEAM

1 Introduction

In 20th-century Europe, women filled roles in mechanics, armament factories, and handicraft businesses due to men’s absence during WWII. However, they

were later relegated to traditional social positions in the 1950s and 1960s. Interestingly, women predominantly worked as mathematicians for NASA and IBM before computers were developed. The construction of the first calculating machine “ENIAC” in 1946 shifted the focus to male operators. It wasn’t until the 1970s that the lack of women in IT became a concern [14]. When looking at female scientists and engineers across EU members, the highest numbers were found in Lithuania (52%) and Bulgaria, Latvia, and Portugal (each 51%), whereas the lowest were discovered in Hungary (33%) and Finland (31%) [7]. In 2021, more than half (54%) of individuals and women (52%) in the EU possessed basic or above basic overall digital skills: proficiency in information and data literacy, communication and collaboration, digital content creation, safety, and problem-solving [7]. Among girls aged 16-19, this percentage significantly increased to 70% [6]. The “Digital Compass” aims for citizens to gain at least 80% basic digital skills by the year 2030. By now, the highest scores of females with basic or above basic digital skills have been found in Malta (98%), Croatia, and Finland (each 93%), the Czech Republic (89%), and Austria (87%), Slovakia (65%), whereas the lowest shares were registered in Luxembourg (60%), Italy (59%), Bulgaria (51%), and Romania & Germany (each 47%) [6]. In Germany, there are 6.5% fewer first-year students in STEM subjects compared to the last year. But if you take a closer look at the gender distribution and the individual subjects, in the year 2014 over 300,000 German students studied mathematics or natural sciences with 40% female students. Since 2015 the number of students dropped but mathematics and natural sciences students increased to 50% in 2021 [20]; a similar trend can also be observed in Austria, especially in the teaching degree “Biology” [22]. Taking a closer look at Austria, there is still a major difference between male and female STEM students at public universities. In the winter term of 2020/21 there were 143,251 enrolled male students in total at public Austrian universities. Of those, 63,064 (44%) were studying STEM subjects, whereas in contrast only 23,787 (17%) were enrolled in liberal arts. In contrast, the winter term 2020/21 counted 166,815 female students – only 45,987 (28%) were enrolled in STEM majors, while 55,096 (33%) studied liberal arts. Especially when picking out the field of “Computer Science and Communication Technology”, there were 18,974 (79%) male and 4,953 (21%) female students in all Austrian public universities. We can even find lower numbers when looking at graduations in 2019/20 in the field of “Computer Science and Communication Technology” (2,203 male, 504 female) [19]. Since girls are still not strongly represented in the STEM field in many European countries [22], [20], there is a high need for teachers and scientists to develop new creative approaches to specifically support young women. Therefore, this work-in-progress aims to show new teaching possibilities to combine BBP and scientific topics from the subject of biology (“Easy Coding in Biology” [17]) in order to promote both disciplines (biology and CS) and to arouse and increase interest in both subjects.

2 Computational Thinking in Science Education

By incorporating CT and CS into biology, students can deepen their understanding of scientific concepts while developing important CT skills applicable across various fields, innovatively and creatively [26], [12]. CT can be taught interdisciplinary in science education through multiple approaches: (1) scientific modeling, where students engage with models to grasp conceptual understanding of phenomena; but still, most teachers teach *about* models, rather *with* them, which includes memorizing models to show conceptual understanding of a certain phenomenon [4] [11] [18]. Another approach is (2) integrating technology by using programming languages, AR/VR applications, educational apps, and data analysis software in science lessons [23], [17]. Additionally, (3) project work and (4) interdisciplinary collaboration, including experiments with STEAM (Science, Technology, Engineering, Arts, Mathematics) teachers, offer further avenues for incorporating CS concepts and supporting CT skills [26], [12].

3 Programming and Science Education

CS and CT are usually not associated with science subjects but student and teacher studies have already demonstrated how to teach and implement CS and CT concepts through workshops and courses in science education in an unplugged or plugged way [13], [3], [9]. Educational applications using BBP, such as MakeyMakey, <colette/>, Scratch, OzoBlockly, or Micro:bit, are promising and possible ways to teach CS skills such as programming. BBP offers shows advantages over text-based programming, making it easier for novice programmers to understand and memorize commands, and reducing syntax errors [28], [27]. In a study comparing block-based and text-based coding, high school students improved their results after attending a coding workshop, with better outcomes and increased interest seen in the BBP group [25]. In 2011, a first attempt was to correct the absence of CT and CS in biological education by introducing a “Computational Biology” course to advanced biology classes, aiming to show high school students how CS is used in the subject of biology and why basic digital skills are necessary for research in many fields of biology [8]. Recent K12 courses have combined text-based programming with biology, such as “Programming in Biology” in 2018 [29] and “Biology Meets Programming: Bio-informatics for Beginners” in 2023 [21]. Block-based coding has been successfully integrated into biology education with tools like “BioBlocks” [10], allowing the programming of biological protocols using visual tools based on Google Blockly and Scratch. Furthermore, the “BioCode” bio-informatics program combines CS and biology by introducing students to CT and programming concepts in the context of biological problems, aiming to enhance students’ understanding of biological concepts while developing their programming skills in Python [2]. Another research also showed the successful integration of BBP into STEM subjects like Biology [1]. In a 2020 study, teachers emphasized the importance of BBP in science classes but struggled to design authentic coding activities [23].

4 Methodology

The tasks described in this paper are based on a newly developed mobile Augmented Reality application (mAR) <colette/> (Computational Learning Environment for Teacher in Europe) (available on Android and iOS).

4.1 Research Aim

Women are underrepresented in STEM fields, but the number in natural sciences such as biology is still much higher than in informatics. The “Easy Coding in Biology” workshop series aims to bridge the CS gender gap by blending CS concepts like block-based programming (BBP) with biology education. The participating students are introduced to fundamental concepts such as monocots and dicots, seed germination, and plant biology. This project develops computational thinking (CT) skills among female students while teaching fundamental biology concepts. Through BBP activities, students engage with both subjects, fostering a comprehensive learning experience. Incorporating creativity and interdisciplinary methods, the broader “Easy Coding in Science” initiative addresses various science topics, integrating CS concepts into chemistry, physics, and biology lessons. This ongoing mixed-method pilot study utilizes participatory observation [15], descriptive statistics [24], and Davis’ Technology Acceptance Model (TAM)-inspired questionnaires [5]. The goal is to deliver life science content and CT education, focusing on CS competency and gender gap reduction. This report showcases findings from the workshop tasks. The coding exercises were meticulously designed for three-dimensional observation, combining cubes to create biological structures. Each task allowed singular block placement with constant values or sequence assembly via loops and variables. The workshop targets programming skills, CT, and biology understanding. For instance, the pyramid’s structure reveals layers, squares, and columns. This systematic analysis led to optimized algorithms through loops and variables, spotlighting advanced solutions as a vital research avenue.

4.2 Experimental Design, Data Collection, and Processing

The first pilot workshop (WB1) “Easy Coding in Biology” from the coding series “Easy Coding in Science” took place in February 2023 in Slovakia [17], at the Constantine Philosopher School in Nitra/Slovakia, and the second workshop (WB2) took place in March 2023 at the Johannes Kepler University in Linz/Austria. In WB1, only the BBP tasks were tested - in Austria also the biological ones. The students were observed by the authors [15], and students’ progress, performance, suggestions, and end results were documented. Data collection and processing took place from February-April 2023. The final codes, the answers to the questionnaire, and the log data (number of trials, success rate, use of loops) of the BBP <colette/> learning environment were also analyzed. For the evaluation, a questionnaire (Appendix; Duration=15min) based on Davis’ Technology Acceptance Model (TAM) was employed [5]. Descriptive

statistics were used to gather and process quantitative data (e.g., opinions on entertainment, difficulty, and interest) [24]. During the workshops, one or two instructors supervised the participants. For assessment, the students used the BBP <colette/> application and its AR feature on their mobile devices. The workshop is divided into four phases: Introduction, Task Assessment, Discussion, and Evaluation (Appendix). The didactic approach, based on Sabitzer's COOL informatics concept, combines hands-on Biology and CS activities, individuality, discovery, cooperation, group discussions, and CT and problem-solving tasks [16]. The project's success is measured by the student's ability to grasp the biology and CS concepts covered during the workshop, achieving this by employing formative assessments via the app (correctness of the code), guided exploration, and application-oriented tasks, which are designed to ensure students' active engagement and attainment of the learning goals.

4.3 Sampling

The study included 47 students from Austria and Slovakia with prior programming experience in Scratch and Python. In February 2023, 37 Slovak secondary school students (19 females), aged 16-19, completed Workshop 1 (WB1) and the 15-minute evaluation questionnaire. Workshop 1 lasted 45 minutes. In Austria, Workshop 2 (WB2) took place in March 2023 with ten gifted students aged 11-12 (four females) from diverse secondary schools. WB2 lasted 35 minutes, followed by a 15-minute evaluation and a 30-minute craft session.

5 Preliminary Results

According to the log data, it took one pair or one student working alone 3.6 trials to successfully complete one of the tasks. Furthermore, the majority of the students in Slovakia and Austria successfully completed the tasks (75%), whereas 50% were using loops. Quantitative data showed that the majority of the participants stated neutral, agreed, or strongly agreed that they liked to collaborate in their regular biology class and in the coding workshop with biological tasks. Around 48% of the female students stated, that they collaborated even more in the coding workshop than in their regular biology class. Twenty-six percent of the girls found the workshop and tasks motivating, and over 56% interesting and/or entertaining because the app is "fun to use". The female participants also stated that 60% had no issues with the BBP app and tasks, while 74% had no problems with the task introduction because they were "easy to understand" (Figures and Preliminary Results, Appendix). Regarding the observations, according to the instructor during WB1, students demonstrated proficiency in completing individual biologic BBP tasks and enjoyed collaborative coding workshops. However, the size of AR markers on the CT scans caused difficulty in task three (Tree), as students had to hold the scan far from their devices to utilize the AR view. Frustration arose when codes were unintentionally deleted due to app switches or crashes. Students were observed debugging their

code and incorporating patterns or loops from other codes. Spatial orientation and variable placement posed challenges in using loops. The instructor noted that students felt stressed and required more time for task assessment [17]. In WB2, participants were engaged in tasks despite initial technical difficulties. QR code reading and AR scan issues caused frustration and disappointment again. Still, participants displayed motivation and familiarity with the programming interface. No notable gender differences were observed in BBP, performance, or workshop design. Loop usage varied, with some students mentioning them but not utilizing them unless prompted. The advanced task (Egg) proved challenging for younger students, even those with programming experience. Technical issues persisted with the learning environment (app) and AR feature. The evaluation questionnaire was completed immediately after programming, not after the crafting session.

6 Discussion

In February 2023, during the pilot phase, distinct URLs were assigned to tasks for description access. However, from July 2023, the mobile <colette/> app brought a significant change to task-solving. It offers structured task progression with specific path codes for accessing tasks. The app also allows direct comparison of student solutions with the apps' sample solutions, enhancing students' engagement. It now accommodates various task formats like Parsons puzzles and error identification. Educators can establish digital classroom sessions to monitor student attempts and submissions, refining pedagogical strategies. Findings suggest comparing higher and lower-gifted students may not be appropriate due to their varied abilities. Individual skills and capabilities should be considered when evaluating task performance. After the first workshop, tasks were modified for younger students, with haptic elements for spatial imagination and creativity. Bias potential exists due to the timing of the questionnaire in Austria, warranting adjusted testing for more accurate insights. BBP introduction via Scratch and prior coding experience could lead to coding difficulties due to language transition. Encouraging loop use and emphasizing their significance could mitigate this. Future studies might explore technical alternatives for smoother learning and clearer loop instruction. Time management adjustments were made based on participants' struggles during the workshop. These findings underscore the need to consider individual abilities, address biases, and refine instruction for effective BBP workshop outcomes.

7 Conclusion and Outlook

Important insights were gained from the two workshops, informing the future implementation of "Easy Coding in Biology" workshops across Europe, as for example the time management and task assessment were modified. Tasks were

further developed to cater to younger students, and haptic task parts were introduced to enhance spatial imagination and creativity. Overall, the interdisciplinary tasks using the BBP app <colette/> proved interesting for secondary school students. However, technical issues with the learning environment, such as AR markers and scans, need further improvement. Despite these challenges, the majority of participants were able to complete the tasks, use loops and engage with the process. Collaboration in regular biology classes did not show significant changes compared to the workshop. The students in the Austrian and Slovakian pilot study had fun, worked together, and were interested and motivated during the workshop, especially in combination with the creative craft part. Based on the first two workshops, webinars for the teachers will be offered in advance to standardize the timing (evaluation after the workshop). In addition, the teachers involved are interviewed during the online training and the interview is recorded. For the introduction to BBP, the commands of the program were printed out and set up like a puzzle system to facilitate the introduction of the blocks. Further workshops in the subjects of physics and chemistry will follow in 2023 and 2024. More than 200 pupils are expected to take part in the “Easy Coding in Science” workshop series in Germany, Austria, and Slovakia. Consideration is also being given to extending the series to text-based programming.

8 Acknowledgement

This research was partially funded by Slovak Research and Development Agency, grant number APVV-20-0599, by the Ministry of Education, Science, Research and Sport of the Slovak Republic, grant number 015UKF-4/2021.

A Appendix

A.1 Preliminary Results and Course of the Workshop

The workshop “Easy Coding in Biology” is divided into four phases:

1. Introduction: Discussion and introduction of the biological content, task design, the app, and the BBP languages; What is the difference between monocots and dicots (examples of representatives); What does it take to germinate seeds, and how fast do cress seeds germinate? What ingredients does cress have? How to build a plant bed with block-based programming? What commands are required and how can you cleverly shorten the code and incorporate loops?
2. Task Assessment 1 and 2: Solving biological (WB2), crafting (WB2), and CS tasks of the biological items (WB1-2)
3. Discussion: Discussing the issues, benefits, final approaches, problems with the tasks, BBP, and app (WB1-2)
4. Evaluation: Filling out the questionnaire (WB1-2)

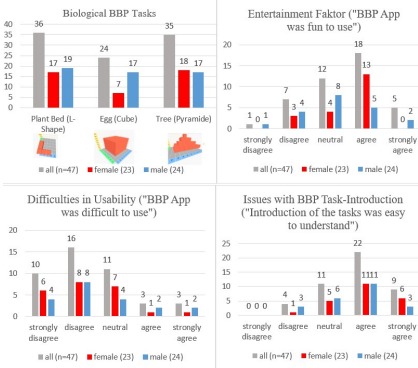


Fig. 1. Number of tasks attempted by students (upper left), students’ perceptions on the entertainment factor (upper right), usability (bottom left), and task introduction (bottom right)

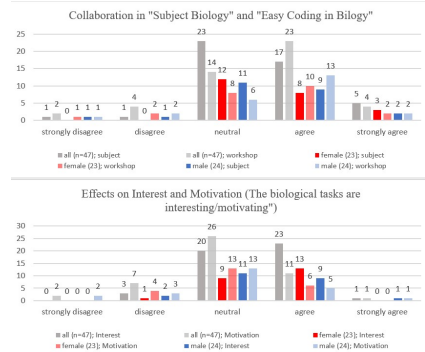


Fig. 2. Students’ perceptions on collaboration in regular biology class and in the workshop (above), and on the effects regarding interest and motivation (below)

A.2 Questionnaire

1. Age (open-ended question)
2. Gender (Select one: female, male, non-binary, no gender, no answer)
3. Which task(s) did you work on? Please tick the appropriate answer(s)
 - Create an Algorithm for an Egg
 - Create an Algorithm for a L-Shaped
 - Create an Algorithm for a Conifer
4. It is enjoyable to collaborate in my regular Biology (Science) classes
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
5. It was enjoyable to collaborate in this Biology coding workshop
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
6. The biological tasks were very interesting
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
7. It was easy to understand the instructions
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
8. It took a long time to learn to use the app
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
9. This app is difficult to use
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
10. The app is clear
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
11. This app is fun to use
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
12. The app easily does what I want
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
13. I would like to use this app in school

- Strongly disagree - Disagree - Neutral - Agree - Strongly agree
- 14. I would like to use this app outside of school
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
- 15. The app has apparent faults
 - Strongly disagree - Disagree - Neutral - Agree - Strongly agree
- 16. If so, please explain why (open-ended question)

References

1. Andersen, R., Mørch, A.I., Litherland, K.T.: Learning domain knowledge using block-based programming: Design-based collaborative learning. In: Fogli, D., Tetteroo, D., Barricelli, B.R., Borsci, S., Markopoulos, P., Papadopoulos, G.A. (eds.) *End-User Development*. pp. 119–135. Springer International Publishing, Cham (2021)
2. BioCode: Learn bioinformatics, <https://www.biocode.ltd/>
3. Blum, L., Cortina, T.J.: Cs4hs: An outreach program for high school cs teachers. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. p. 19–23. SIGCSE '07, Association for Computing Machinery, New York, NY, USA (2007). <https://doi.org/10.1145/1227310.1227320>
4. Buckley, B.C.: Model-based learning. *Encyclopedia of the sciences of learning* **5**, 2300–2303 (2012)
5. Davis, F.D.: A technology acceptance model for empirically testing new end-user information systems: Theory and results. Doctoral dissertation, Massachusetts Institute of Technology (1985), <https://dspace.mit.edu/handle/1721.1/15192>
6. Eurostat: 70% of EU girls had basic or above basic digital skills, <https://ec.europa.eu/eurostat/en/web/products-eurostat-news/w/edn-20230427-1>
7. Eurostat: Eu had almost 7 million female scientists in 2021, <https://ec.europa.eu/eurostat/web/products-eurostat-news/w/ddn-20230210-1>
8. Gallagher, S.R., Coon, W., Donley, K., Scott, A., Goldberg, D.S.: A first attempt to bring computational biology into advanced high school biology classrooms. *PLoS Computational Biology* **7**, 1–7 (2011), [10.1371/journal.pcbi.1002244](https://doi.org/10.1371/journal.pcbi.1002244)
9. Goldberg, D.S., Grunwald, D., Lewis, C., Feld, J.A., Hug, S.: Engaging computer science in traditional education: The ecsite project. In: *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*. p. 351–356. ITiCSE '12, Association for Computing Machinery, New York, NY, USA (2012). <https://doi.org/10.1145/2325296.2325377>
10. Gupta, V., Irimia, J., Pau, I., Rodríguez-Patón, A.: Bioblocks: Programming protocols in biology made easier. *ACS synthetic Biology* **6**, 1230–1232 (2017), <https://doi.org/10.1021/acssynbio.6b00304>
11. Harrison, A.G., Treagust, D.F.: A typology of school science models. *International journal of science education* **22**(9), 1011–1026 (2000)
12. auf der Heide, F.M., Curzon, P., McOwan, P.W.: Computational Thinking; Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln. *Mathematische Semesterberichte* 2019 **66:2** **66**, 259–260 (2 2019). <https://doi.org/10.1007/S00591-019-00249-0>
13. Lockwood, J., Mooney, A.: Computational thinking in education: Where does it fit? a systematic literary review (2017)
14. Maier-Rabler, U.: Frauen als Nicht-(Mit-)Gestalterinnen der digitalen Transformation. *Die digitale Transformation der Medien* pp. 95–116 (2022)

15. Musante, K., DeWalt, B.: Participant Observation: A Guide for Fieldworkers. AltaMira Press (2010), <https://books.google.at/books?id=ymJJUkR7s3UC>
16. Sabitzer, B.: A neurodidactical approach to cross-curricular open learning. cool informatics. Habilitation (2013)
17. Schmidthaler, E., Stäter, R., Cápaj, M., Ludwig, M., Lavicza, Z.: Easy coding in biology: Pilot workshop design and experiences from block-based programming with colette in secondary education. *Journal of Research in Science, Mathematics and Technology Education* **6(SI)**, 177–206 (2023). <https://doi.org/https://doi.org/10.31756/jrsmte.619SI>
18. Schwarz, C.V., Reiser, B.J., Davis, E.A., Kenyon, L., Achér, A., Fortus, D., Shwartz, Y., Hug, B., Krajcik, J.: Developing a learning progression for scientific modeling: Making scientific modeling accessible and meaningful for learners. *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching* **46(6)**, 632–654 (2009)
19. Statistik Austria: Bildung in Zahlen – Tabellenband, https://www.statistik.at/fileadmin/pages/325/Bildung_in_Zahlen_20_21_Tabellenband.pdf
20. Statistisches Bundesamt: Studienverlaufsstatistik 2022, <https://www.destatis.de/>
21. University, C.S.D.: Biology meets programming. bioinformatics for beginners, <https://www.coursera.org/learn/bioinformatics>
22. University of Vienna: Zahlen, Daten und Fakten zur Universität Wien. Gender im Fokus, <https://personalwesen.univie.ac.at/organisationskultur-gleichstellung/publikationen/#c486668>
23. Vasconcelos, L., Kim, C.: Preparing preservice teachers to use block-based coding in scientific modeling lessons. *Instructional Science* **48**, 765–797 (2020), <https://doi.org/10.1007/s11251-020-09527-0>
24. Vetter, T.R.: Descriptive statistics: Reporting the answers to the 5 basic questions of who, what, why, when, where, and a sixth, so what? *Anesthesia & Analgesia* **125**, 1797–1802 (2017)
25. Weintrop, D., Wilensky, U.: Comparing block-based and text-based programming in high school computer science classrooms. *ACM Trans. Comput. Educ.* **18(1)** (oct 2017). <https://doi.org/10.1145/3089799>, <https://doi.org/10.1145/3089799>
26. Wing, J.M.: Computational thinking. *Communications of the ACM* **49**, 33–35 (2006). <https://doi.org/10.1145/1118178.1118215>
27. Xu, Z., Ritzhaupt, A.D., Tian, F., Umaphy, K.: Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. *Routledge* **29**, 177–204 (7 2019). <https://doi.org/10.1080/08993408.2019.1565233>
28. Yamashita, S., Tsunoda, M., Yokogawa, T.: Visual programming language for model checkers based on google blockly. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10611 LNCS**, 597–601 (2017)
29. Zurich University: Programming in biology, <https://mnf.openedx.uzh.ch/courses/>

Supporting Gender Equality in Computer Science Through Pre-Introductory Programming Courses

András Margitay-Becht^{1,2} and Udayan Das¹

¹ Saint Mary's College of California, 1928 St. Mary's Road, Moraga, CA 94575, USA

² Eötvös Lóránd University, Pázmány Péter sétány. 1/C, 1117, Budapest, Hungary
abecht@inf.elte.hu

Abstract. The past thirty years have seen an exponential growth in the presence of computers in everyday life. However, some studies find that introductory programming courses in university curricula might reduce students' willingness to engage with the material further. To complicate matters, despite extensive efforts of the past decades, there are still corners of the computer science landscape that are unwelcoming to female practitioners, implicitly or explicitly discouraging female students from the profession.

This paper discusses an experimental university-level pre-introductory course targeting non-computer science students with limited or no background in programming. The explicitly stated goal of the course was to reduce trepidation regarding programming. The course saw equal participation from male and female students. The results were positive: students reported a reduced perception of difficulty regarding programming and an increase in the subjective importance of the area. Even more encouragingly more female students experienced a reduction in the perception of programming difficulty than male students.

The paper will discuss the structure of the course, the unique set of approaches that led to success, the lessons learned, and the new iteration of the class we are going to offer the upcoming year.

Keywords: Gender equality · Programming education · Computational thinking.

1 Introduction

There is a global shortage of programmers and computer science professionals, in spite of the relatively high salary. There are many reasons for this, but a contributor is the low and decreasing engagement of females with the discipline. In the mid 80-s, 37% of computer science students were female [1]. This shrank to 27% by 1997, then to 20.7% by 2006, and hit 18.7% in 2016 [2]. The total number nearly doubled between 1997 and 2016, from 6900 to 12,200, but this growth is significantly lower than the growth experienced among male students.

This paper will discuss why this discrepancy might exist, and what are some possible ways of reducing this effect. We will also introduce our approach that targeted the fear of computer science in general, but ended up also empowering female students in the pursuit of computer science knowledge.

2 The Problem with Introductory Classes

One of the problems of teaching computer science, whether for male or female students, is the fact that the topic is generally considered scary (for a few examples see [3–5]). Of particular interest is the research done by Alford et al in 2017[6]. The original purpose of their research was to investigate the gender gap, but they found that students in general, irrespective of gender start out intimidated by programming. The student population they investigated were mostly engineering students who had to take an introductory programming class (so many of them were there because they had to and not because they wanted to). The total number of students investigated were close to 2000, of whom about 35% were female. Importantly, they found no real difference between the interest of male and female students before taking the class. When asked what grade they were expecting, male and female students again responded similarly, but to a question regarding how confident they were that they can successfully complete the course, the male students responded more positively, in alignment with previous findings. Interestingly, when asked how confident the students were in their ability to succeed in the course, the pre-test results showed greater confidence for men than women, but the post-test saw men drop in their self-assessment more than women (indeed, in some samples women increased their confidence throughout the course). The greatest takeaway for us was hidden in the phrasing of the previous sentence: men and women both left the introductory classes less interested and confident and more intimidated than at entry time. This is in agreement with LaBouliere’s findings from 2015: middle school girls increased in their programming understanding and confidence, but greatly decreased in their interest in programming[7]. Similarly, Rubio reports a loss of interest for female students as well[1].

It appears that introductory courses might not be the best places to provide an in-spiring experience for students. These classes have to discuss a significant curriculum of complex topics, covering those usually takes up the whole course, and there is little time to slow down and spend some time working on an interesting project or real life problem. Introductory classes are excellent for students who are already interested in the subject and determined to follow through. Indeed, they are designed to serve this audience, so the speed of the classes and the amount of material they cover are scaled to support these students. But this leads to the above situation where students who are not already positively predisposed towards the discipline might be further turned off by the introductory classes.

A solution is to start with a pre-introductory experience, like the one discussed in [8]. As Schindler and Muller state: “The first contact with programming is crucial to keep students in the long run”, so if the curriculum design or educational system allows for it, creating a “first contact” experience might be valuable.

Our approach [9] is similar: we are trying to provide an educational experience that is primarily focusing on showcasing the value, and indeed the joy of programming, while covering some basic concepts of coding. We have created

our class, “Coding is Fun”, not as a replacement for introductory classes, but as a place for curious students to explore their curiosity, see if they find value and joy in programming, in order to provide them with the mindset that will set them up for success in the introductory programming class. Our approach differs from the above cited Schindler and Müller approach by utilizing the advantage presented by the fact that we are teaching at a small liberal arts university with class sizes around 20, allowing us to give control of a large portion of our class over to our students. Instead of teachers, we became guides, and as we guided them throughout the basics of programming along their own interests, pivoting as they wanted, we succeeded in increasing their interest and reducing their apprehension. It also allowed us to use the approachable Scratch programming environment, something that would have been unsuitable for an introductory class at university level. Our course also serves as an optional course unattached to any program, so it is accessible not only to computer science students, but for anyone interested in general.

We have designed this experience for all students, irrespective of gender. We have found, however, that our female students seem to have benefitted more than our male students from this approach. In this paper we will quickly discuss some of the issues and prejudices affecting female students of computer science, then discuss our course and the outcomes of the pre-introductory experience.

3 Key Issues for Female Students

3.1 Skills

First and foremost it is important to discuss whether there is a capability gap between male and female computer science students. Murphy et. al., investigating the gender differences among male and female students at a computer science program in 2006 found that women in general started the program knowing less than men did, but as they progressed through the program, both their knowledge and academic performance (GPA) caught up to that of the male students. In 2015 Akinola completed an empirical study of programming skills between male and female groups of two and four, finding no statistically relevant difference between the genders based on either efficiency or accuracy [10]. Akinola concluded that the underrepresentation might come from different interests or fear, lack of confidence.

In a 2018 study Kallia and Sentance compared mostly 11th grade students from 7 different UK schools, finding that boys and girls performed roughly the same [11]. A 2019 study at the Graz University of Technology found similar results: freshmen female computer science students performed just as well as males did[8].

Study after study confirms our personal experiences, that female students are not worse at programming and computer science related tasks and courses than their male counterparts. The underrepresentation must come from another source.

3.2 Interest

A frequent component of research projects focusing on the gender gap is the apparent lower interest of female students in the field. Pau et. al. reports on the generally held preconception that bad experiences with programming classes can alienate female students from pursuing the discipline [12]. They found that with proper support and structure, programming courses can be empowering to female students.

Funke et. al. surveyed 63 Bavarian computer science teachers [13]. Most of them reported no dissimilarities between boys and girls, only three categories showed meaningful differences: girls were perceived as more structured but less confident and interested. Master et. al. report on an experiment enriching the computer science experience of 6-year-old students with the use of robotics[14]. Like Funke, they also found that boys had greater intrinsic interest than girls, but the introduction of robotics increased girls' interest significantly more than boys', drastically reducing the interest gap between the two groups. Braga and Motti explored a similar age group, 7-10 year olds [15]. The children were invited to participate in programming exercises inspired by the worlds of Frozen, Minecraft and Angry Birds. They found that if the environment is engaging enough, both girls and boys are equally likely to participate. Moreover, "girly" themed experiences – as the article calls them – were not a major motivational factor for the girls.

3.3 Confidence

Female students, overall, tend to be less confident than their male counterparts, often undervaluing their own capabilities. This was the central question of Kallia and Sentence's research, finding that despite similar performance, girls regularly underestimate themselves [11]. More worryingly, girls also scored lower on self-efficacy than boys, indicating need of better support. Similar results were found in [8, 13] as well: female students matching the performance of male students, but rate themselves worse than males do.

3.4 Possible Solutions

If lower confidence and motivation keep highly competent girls and women from pursuing computer science, it is important to address these issues directly. Rubio et. al. suggested creating separate introductory tracks for people with different back-grounds, enabling easier transition into the discipline [1]. Alternatively, they suggested more contextualized classes like Media computation, robotics, or animation, where the utility of the field is easier to experience. Their own approach focused on using physical computing, which led to a reduced failure rate for females in the class, although it still caused a reduction in interest in computing among female students, like the alternative programming approach did.

Pau et. al. discuss the key issues they found that can increase the positive experience of an introductory programming class for female students [12]:

1. Programming tasks that are connected to real-life issues and actual problem solving are a lot more engaging, similar to Rubio's recommendations above
2. When time pressure is removed and students are allowed to work from home, they find the class a lot more beneficial
3. Parental support from home
4. Higher mathematics performance helps transitioning into programming

We have incorporated some of these findings into our course design to make our class more appealing – to female and male students alike.

4 Pre-Introductory Programming: a Combined Solution

4.1 Goals of the Course

The primary purpose of our course, called “Coding is FUN” was to show that, as the title describes, coding can be fun. At our university, we offer a mandatory January Term experience for the students. The instructors are encouraged to create unusual and experimental courses, and we designed this class to provide an opportunity for students who have not yet tried their hands at programming to explore the area a little bit. The course description itself that was available to the students before enrolling explicitly stated this.

4.2 Recruiting and Student Population

The course was originally intended for freshmen students just entering college, to help them learn about potential major opportunities like Computer Science (CS) or Data Science (DS). In the end, we ended up with a significantly more diverse group spanning from freshmen to seniors. The recruitment of students for the course was largely based on word-of-mouth. The university's Tech Club overseen by one of the instructors held two short super-introductory programming workshops. The upcoming course was announced at these workshops. Two of the students involved in these workshops also ended up becoming peer tutors. Instructors also shared the course information with peers in other departments. Many students in the CS program and DS programs shared this upcoming course with roommates and friends. One CS senior for example had 2 roommates attending the course.

Since our university is a small liberal arts college, class sizes are traditionally between 15 and 25 students. We were initially worried that there might not be enough interest for the course, but in the end we ended up with 23, close to the maximum class size allowed.

A point of pride is the diversity of students we managed to address, both by gender and by major of study. We had 7 students of Business-adjacent majors, 4 students of varied STEM majors, 4 Psychology majors and 4 liberal arts majors in the class, in addition to the 4 freshmen who did not yet have declared a major. The gender breakdown ended up being 9 female and 14 male students completing the class.

4.3 Structure of the Class

The course was offered every Wednesday, 2.5 hours per meeting, during the January of 2023. The class was delivered in a synchronous online format, allowing students the ability to work from home. To reduce the impact of time pressure further, the course required the students to create only a single program as their final project; no tests, quizzes or homeworks were assigned. They were also allowed to work on their project whenever they wanted, with ample support being provided for them both online and offline.

The intended layout of the course was a session introducing Scratch programming, then a session of more advanced programming topics based on student choice, then a session on design and computational thinking and a final session for the students to showcase their content. Due to great student interest, this plan expanded by 50% to provide them with the extra opportunities they asked for.

We decided to use Scratch as the initial introductory language as it is extremely approachable even for 5-year-olds, and it all but removes any concern about syntax in programming. It is also complex enough that the basics of imperative programming can be found in it: variables, conditional statements, loops, functions/methods and lists, all in an approachable, graphical environment. It is also built around events, so students also learn event-based control. Since the class aimed to be a pre-introductory course to be followed up by a traditional introductory course, the short-comings of the programming language were considered less important than the immense ease-of-access benefit the language provided. This was a popular choice among students, and most of them stuck to using Scratch throughout the class.

During the first class session we set up a Slack channel to improve communication with the students. After the first class, a poll was posted there, asking if they want more Scratch practice during the second class, or want to see the basics of Spread-sheet programming or Python. To our surprise, while Visual-Basic based spreadsheet programming was entirely unpopular, many students showed interest in both more Scratch and learning some Python, so we decided to pivot the course plan: during the normal class time a Scratch session took place, and an optional Python period was added in the afternoon. Highlighting student interest, this optional period was well attended, just like a second optional Python session, that provided further details on the language. This was great feedback for us, as students seemed to have wanted to engage with the material significantly more than we expected – an insight that will be incorporated into the next iteration of the class.

During the third and fourth weeks additional scaffolding was provided to the students in the form of consultation periods both synchronously and asynchronously over Slack or e-mail. The third class period focused on how the design and computational thinking principles can be utilized to come up with a final project, and iteratively design, implement and test it. During the final class period the students showcased projects far more complicated than we expected or imagined, demonstrating that they have spent a significant amount of time

outside the classroom learning additional techniques just for the amusement of themselves.

4.4 Scaffolding and Support

Student support is a critical element for bridging-the-gap for students who may not have as much exposure to programming or programming-adjacent materials in the past. Students need to know that programming is not a magical thing that some people just get and others cannot, but that programming is a skill that can be learned and honed. As with many other human endeavors and activities, some people are naturally more adept at programming, but that does not mean that others are incapable of learning this skill. The presence of in-class support provides students with the opportunity to talk through some difficulties, particularly for those students who are hesitant to talk to the class as a whole. Peer tutoring programs have demonstrated success, particularly in Computer Science learning contexts and can improve retention and student achievement [16].

There were 3 students who served as peer tutors for the course. They were selected such that they would be from different levels of proficiency. One student had recently completed Programming I (Programming with Python), one student had recently completed Programming II (Data Structures and Algorithms with Python), and the other student would be graduating with a Bachelor's Degree in Computer Science in 6 months. Thus, one student was at a first year level, the second student was 2nd year level, and the final student was at a senior (4th) year level. The different levels are beneficial in peer tutors to express to participants that learning programming is a journey and people at all levels have something to share. Peer tutors at different levels also bring different benefits with those fresh out of an introductory programming course having had recently experienced their own journey from insecure to self-reliant confident when it comes to programming, at the other end the senior level student brings the benefit of having more experience and knowledge yet being closer to the students than the instructors. Peer tutors always have the benefit of allowing more informal exchanges between students and peer tutors. The tutors for this class were also relatively diverse with 1 woman and 2 individuals from minority groups.

Support in this course involved both during class session support (peer tutors in groups) as well as peer-tutor led sessions during the project phase. About 8 students took advantage of sessions to work on the project during an open lab session with a peer tutor. In the week preceding the final presentations (week 3 of the course) these sessions were held. Instructors also held support sessions during this period with an additional 2 students attending. Altogether a total of 10 students used the support sessions. The availability of multiple sessions at different times of day and with different individuals surely contributed to many students taking advantage of the sessions. Slack was also used as an online forum space for communication, community, and support and students took advantage of Slack to troubleshoot code snippets, and importantly, share their work with all other students at the end. Further along in the CS program students commonly

submit work to GitHub and so this kind of shar-ing is possible. But starting this kind of sharing enables students to learn from each other, and in the instructors’ experience get impressed and inspired by each other. No amount of code samples provided by the instructors can match the value of seeing peer work.

5 Methodology and Results

Grading and assessment were done completely separately in this course. For grading, we used a growth mentality. As this course was an elective class that did not count for any major field of study or the core curriculum of the university, and only carried a minuscule credit value (less than 0.7% of the total credits required for graduation), we felt comfortable assigning grades based on the improvements our students made in the small amount of time available in the class. Our key assessment focused on the student experience in the course and the change of attitude we were hoping to achieve. To measure this, we created a pre-class and a post-class questionnaire for the students, focusing on their experience with and attitudes towards programming. Two key questions are worth discussing in more detail: the perception of difficulty and the perceived utility of programming. The following tables will show information from the 15 of the 23 students who filled out both surveys. It is interesting to note, that close to 90% of the female students filled out both the pre- and post-class questionnaires, while only 50% of male students did so.

5.1 Perceptions of Difficulty

To test the perceptions of difficulty of programming, in the pre-test we asked the students how hard the class will be for them. Not surprisingly, none of them said they expected the course to be too hard for them, as it was an elective course working with a positively biased audience. Most of the students, however, chose the tentative “I can probably do it” option, with only 3 (2 males and 1 female) selecting the assertive “I can for sure do it”.

At the end of the class, we were happy to find that close to half of the respondents found programming at least somewhat easier than they expected. Most of the rest found that programming was about as hard as they were expecting, with 2 students (a male and a female) feeling that programming was a bit harder than they thought initially.

While the above results are already exciting, they become even more so if we look at the gender breakdown of the responders. Table 1 shows the breakdown of the responses: the rows contain the questions from the pre-class questionnaire, while the columns the options from the post-class questionnaire. In the cells the values are in the form of [male : female] students giving that combination of answers. In alignment with the established literature, we found that six of the eight students who found programming to be easier than expected were females, while only four of the six who found it as hard as they expected were males. This means that the majority of the perception gain happened to the female students in the class, empowering and encouraging them to pursue programming further.

Table 1. Perceptions of programming difficulty before and after the course. Rows contain the post-course feedback, columns the pre-course feedback. Results are reported as [males : females]

	Too hard for me	I can probably do it	I can for sure do it
A bit harder than expected	0:0	1:1	0:0
About as hard as I expected	0:0	3:2	1:0
Easier than expected	0:0	1:4	0:1
A lot easier than expected	0:0	0:0	1:0

5.2 Perceived Utility

While it is exciting to see that the class increased the confidence and reduced the worry of female students, that alone is not going to improve participation if they find programming to be unimportant. To measure the perceived importance of programming, the pre-class questionnaire asked the students' estimation of the likelihood that programming will be useful for them personally. We had two students (both female) say that they did not expect to use programming at all, five students (4 males, 1 female) say that they will definitely use programming, the rest fell into the more tentative maybe category.

After the class, we were excited to find that six out of the eight females thought that programming will be more useful to them than they thought before the class, and one of the remaining two already thought it was going to be useful and found confirmation in her experience. The male students had a lot more varied journey: one student said that programming will be less useful than they initially expected, and all three of the students who were expecting programming to be a lot more useful to them were also males. It was also exciting to us that the class seems to have reinforced initial positive expectations: all 5 students who were certain of the usefulness of programming experienced increased valuation of it.

Similar to above, Table 2 showcases the breakdown of [male:female] respondents based on their preclass and post-class responses.

Table 2. Perceptions of the utility of programming before and after the course. Rows contain the post-course feedback, columns the pre-course feedback. Results are reported as [males : females]

	I won't use it	I might use it	I will use it
Less useful than expected	0:0	1:0	0:0
About as useful as expected	0:1	0:1	0:0
Somewhat more useful than expected	0:1	1:4	2:1
A lot more useful than expected	0:0	1:0	2:0

5.3 Overall Results

Like the above cited literature, we have found no difference in the quality of work done by male and female students. Indeed, the most complex project was created by a female student, and every female student turned in a project that greatly exceeded the expectations – and the material covered in the class. This is an excellent indicator of increased self-efficacy, that was demonstrated by nearly all students. We hope that this skill will prove to be transferable to other areas as well.

Computational thinking, especially decomposition, was clearly demonstrated in most projects. This will reinforce and expand the students’ critical thinking skills. We have seen both explicit and implicit use of design thinking. The most impressive ex-ample was one student who worked together with her younger brother to create a game with him for him to enjoy. This was both a touching moment – siblings using a class exercise to socialize remotely over Zoom – and also a great application of design thinking principles, working with and for a “client” to create a desired product. A final indicator of the success of the class, that out of the 15 responders, 13 re-ported desire to continue learning programming, either on their own, or in some kind of structured manner.

6 Next Steps

Due to the great success of the class, it will live on in the January of 2024, with some modifications. The greatest one of these will be an expansion to a full class, increasing the meeting times from four to 15. The increase in contact hours can provide an opportunity for introducing pair programming in the class. Pair programming and peer programming are also known to be effective learning techniques within software engineering and are a significant element of Agile Software Development practices. Talking through the thought process behind code development greatly strengthens students’ overall programming skills. Pair programming has been demonstrated to improve retention and student confidence [17].

A change in recruiting efforts will attempt to address the needs of incoming fresh-men. The class will continue to target students with no programming background, but it could serve as an ideal first experience not only for students who are programming curious, but also for those who know they want to pursue a career in computer or data science and want a more gradual on-ramp to programming prior to taking their introductory course.

The course will also aim for a more layered outcome regarding further study for the students: aside from continuing to study on their own or just taking an introductory class, they might be able to pursue a certificate in programming or website development, a minor in programming or data science, or even a major in computer science or/and data science. The majors and minors at our institutions are created in a way that they scale easily. We consider computer and programming abilities necessary for the enlightened citizens of 2023, and this

course might provide a gateway for students of all backgrounds and interests to expand their education with some 21st century skills.

7 Conclusion

Optional pre-introductory experiences implement separate introductory tracks for students with different backgrounds. Those who enter university with some back-ground in programming can enroll directly into an introductory programming class. Those, however, who are worried about their own skills or even underestimate their own abilities, as many female students seem to, can participate in a pre-introductory course to alleviate their concerns and improve their self-confidence. A student-centric, real-life grounded pre-introductory class can showcase the usefulness and fun of programming, creating and reinforcing interest. For example, a student who enjoys drawing can use programming to create animations. A student interested in child psychology can use programming to create research tools to engage children – or analyze the results of the engagement. A biochemist can use programming to model molecules or analyze experimental data, an economist can model the success of a product or the trajectory of a nation. By being able to focus on inspiring students, these classes can serve as the affective counterparts to the cognitive focused introductory courses[18]. And by focusing on increasing student interest, breaking down barriers, improving self-efficacy and confidence, pre-introductory courses can help support female students exactly in the ways they need support to be able to start a career in the IT field – or to expand their interest with technology.

References

1. Rubio, M.A., Romero-Zaliz, R., Mañoso, C., de Madrid, A.P.: Closing the gender gap in an introductory programming course. *Computers & Education*. 82, 409–420 (2015). <https://doi.org/https://doi.org/10.1016/j.compedu.2014.12.003>
2. NSF - National Science Foundation: Women, Minorities, and Persons with Disabilities in Science and Engineering: 2019, <https://ncses.nsf.gov/pubs/nsf19304/digest>. Last accessed 2023/06/02.
3. Connolly, C., Murphy, E., Moore, S.: Programming Anxiety Amongst Computing Students—A Key in the Retention Debate? *IEEE Trans. Educ.* 52, 52–56 (2009). <https://doi.org/https://doi.org/10.1109/TE.2008.917193>.
4. Höök, L.J., Eckerdal, A.: On the Bimodality in an Introductory Programming Course: An Analysis of Student Performance Factors. In: 2015 International Conference on Learning and Teaching in Computing and Engineering (2015).
5. Wyeld, T., Nakayama, M.: Visualising the Code-in-Action Helps Students Learn Programming Skills. In: 2018 22nd International Conference Information Visualisation (IV). pp. 182–187 (2018). <https://doi.org/https://doi.org/10.1109/iV.2018.00040>.
6. Alford, L., Dorf, M.L., Bertacco, V.: Student Perceptions of Their Abilities and Learning Environment in Large Introductory Computer Programming Courses. In: 2017 ASEE Annual Conference & Exposition Proceedings. p. 28867. ASEE

- Conferences, Columbus, Ohio (2017). <https://doi.org/https://doi.org/10.18260/1-2-28867>.
7. LaBouliere, J.J., Pelloth, A., Lu, C.-L., Ng, J.: An exploration of the attitudes of young girls toward the field of computer science. In: 2015 IEEE Frontiers in Education Conference (FIE). pp. 1–6 (2015). <https://doi.org/https://doi.org/10.1109/FIE.2015.7344265>.
 8. Schindler, C., Müller, M.: Gender gap? a snapshot of a bachelor computer science course at Graz University of Technology. In: Proceedings of the 13th European Conference on Soft-ware Architecture - Volume 2. pp. 100–104. Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/https://doi.org/10.1145/3344948.3344969>.
 9. Margitay-Becht, A., Das, U.: Enhancing student learning through hidden motivational learn-ing outcomes. In: Enomoto, K., Wagner, R., and Nygaard, C. (eds.) Enhancing student learn-ing outcomes in higher education. Libri Publishing Ltd. (2023).
 10. Akinola, S.O.: Computer programming skill and gender difference: An empirical study. *American journal of scientific and industrial research* **7**(1), 1–9 (2015).
 11. Kallia, M., Sentance, S.: Are boys more confident than girls? the role of calibration and students’ self-efficacy in programming tasks and computer science. In: Proceedings of the 13th Workshop in Primary and Secondary Computing Education. pp. 1–4. Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/https://doi.org/10.1145/3265757.3265773>.
 12. Pau, R., Hall, W., Grace, M., Woollard, J.: Female students’ experiences of programming: it’s not all bad! In: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. pp. 323–327. Association for Computing Machinery, New York, NY, USA (2011). <https://doi.org/https://doi.org/10.1145/1999747.1999837>.
 13. Funke, A., Berges, M., Mühling, A., Hubwieser, P.: Gender differences in programming: re-search results and teachers’ perception. In: Proceedings of the 15th Koli Calling Conference on Computing Education Research. pp. 161–162. Association for Computing Machinery, New York, NY, USA (2015). <https://doi.org/https://doi.org/10.1145/2828959.2828982>.
 14. Master, A., Cheryan, S., Moscatelli, A., Meltzoff, A.N.: Programming experience promotes higher STEM motivation among first-grade girls. *Journal of Experimental Child Psychology*. 160, 92–106 (2017). <https://doi.org/https://doi.org/10.1016/j.jecp.2017.03.013>.
 15. Braga, C., Mochetti, K.: Programming teaching tools and the gender gap in the Information Technology field. In: Anais do Workshop de Informática na Escola. pp. 70–79. SBC (2018). <https://doi.org/https://doi.org/10.5753/cbie.wie.2018.70>.
 16. Servin, C., Pagel, M., Webb, E.: An Authentic Peer-Led Team Learning Program for Community Colleges: A Recruitment, Retention, and Completion Instrument for Face-to-Face and Online Modality. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. pp. 736–742. Association for Computing Machinery, New York, NY, USA (2023). <https://doi.org/https://doi.org/10.1145/3545945.3569851>.
 17. McDowell, C., Werner, L., Bullock, H.E., Fernald, J.: Pair programming improves student retention, confidence, and program quality. *Commun. ACM*. 49, 90–95 (2006). <https://doi.org/https://doi.org/10.1145/1145287.1145293>.
 18. Bloom, B.S., Krathwohl, D.R.: Taxonomy of educational objectives: The classification of educational goals. Book 1, Cognitive domain. longman (1956).

Investigating Code Smells in K-12 Students' Programming Projects: Impact on Comprehensibility and Modifiability

Verena Gutmann^[0009-0007-2892-9160], Elena Starke^[0009-0004-5100-368X], and
Tilman Michaeli^[0000-0002-5453-8581]

Technical University of Munich, Arcisstraße 21, 80333 Munich, Germany
{verena.gutmann,elena.starke,tilman.michaeli}@tum.com

Abstract. Teaching students to code goes beyond focusing on the correct implementation of features. It is also about emphasizing the importance of comprehensible and modifiable code. Addressing these characteristics is already crucial for novice programmers in K-12 computing education, as fostering code quality can support their learning process especially when working collaboratively in group projects. This study aims to investigate the extent to which code smells are problematic in K-12 students' code and understand the impact of different code smells on comprehensibility and modifiability. We initially selected relevant code smells to address this research objective and then conducted a qualitative analysis of 12 student projects. The results show a differentiated picture for different types of code smells. While *Duplicated Code* and *Class Data Should Be Private* may not be critical issues. However, in our data, *Long Functions*, *Speculative Generality*, *Comments*, *Mysterious Names*, and bad *Code Formatting* negatively affected the comprehensibility and modifiability.

Keywords: novices · K-12 · computing education · code smells

1 Introduction

Learning to program is one major challenge in the K-12 computing education classroom. While teaching novices the fundamentals of programming syntax and implementing desired functionality is essential, it is equally crucial to recognize the significance of fostering code quality to enhance the learning process, particularly in group work projects.

Looking into professional software development, besides correct functionality, it is equally important to consider other quality aspects, such as maintainability, testability, comprehensibility, or modifiability. These factors are crucial in facilitating collaboration among developers and ensuring code's long-term viability and maintainability throughout the development process [15].

One key factor that can harm code quality is the presence of code smells [4]. Code smells are not related to functionality or syntax but serve as indicators of potential issues within the code. They act as warning signs, highlighting areas

of code that may require improvement. Comprehensibility and modifiability are particularly susceptible to the harmful effects of code smells and are consequently relevant to consider in collaborative work.

Preparing students to become future professional developers is no goal of K-12 programming education. Nevertheless, it remains crucial to emphasize the significance of writing comprehensible and modifiable code, particularly in collaborative settings. Code smells, which impact comprehensibility and modifiability, can challenge effective collaboration among students. Therefore, addressing code quality and mitigating the presence of code smells can enhance the collaborative learning experience in K-12 programming education. Furthermore, however, it is crucial to enable students to understand how and why a program works or does not work and to be able to comprehend their code. Additionally, students often need help to solve issues independently and sometimes even make it more complicated than it has to be [7].

While existing research has extensively explored code smells in various programming contexts, the specific domain of K-12 students' programming projects still needs to be explored. Therefore, in this paper, we investigate the impact of code smells on comprehensibility and modifiability to address them adequately in the classroom setting.

2 Theoretical background and related work

It is challenging to define comprehensibility in the context of source code due to its subjective nature. Code comprehensibility is closely related to code readability – “a human judgment of how easy a text is to understand” – which can be considered a prerequisite for comprehension [3]. The aim of comprehensibility should be to make it as effortless as possible for readers and future editors to familiarize themselves with the code. A clear structure that includes a meaningful arrangement of instructions and method calls, and uniform code formatting can be helpful [8]. Additionally, self-explaining variable names, simple control structures, and good documentation can support code maintenance, resulting in improved modifiability [3].

In contrast, code smells are recognized patterns in source code that indicate potential areas for improvement rather than bugs or errors. They act as indicators within object-oriented code, drawing attention to areas of weakness that could benefit from closer examination and refactoring. By identifying code smells, developers can proactively address issues and enhance the overall quality of their code. Various taxonomies have been proposed to classify these smells, including inappropriate comments, excessively long functions, or an over-reliance on primitive data types [4]. In addition to code smells, Stegeman et al. propose a rubric for educational settings to assess code quality and promote effective coding practices in novice programming courses. Considering aspects relevant for beginners, this rubric encompasses ten categories grouped into four overarching categories: *Documentation*, *Presentation*, *Algorithms*, and *Structure* [19, 20].

Previous research on quality issues in K-12 education has predominantly focused on block-based programming and tools such as Scratch [6]. In contrast, text-based programming has mainly been studied in university settings.

Looking into block-based programming, there are various findings regarding common issues. The source code of Scratch programs created by high school students is frequently characterized by multiple code smells, such as problematic variable names and duplicated code, which can negatively affect both correctness and readability [5]. Furthermore, students often acquire certain programming habits that can result in code smells. The prevalence of extensive bottom-up programming and extremely fine-grained programming approaches can result in code smells like dead code and an overwhelming number of scripts. Consequently, students may face significant challenges when debugging and maintaining their projects, as the complexity and scale make these tasks practically impossible [14]. Even when the students' projects were not particularly complex, those code smells arise [1]. Additionally, code smells such as too-long methods and code duplications impacted students' performance negatively. The *long method* smell hindered their understanding, while duplication decreased the modifiability of their code [9]. Even if comprehensibility and modifiability seem rather abstract, K-12 students can understand aspects of software quality just as well as general programming concepts [10]. In addition to examining poor programming habits, there is a potential approach to address code quality in K-12 classrooms by emphasizing "code perfumes" that represent good programming practices. Specific structures like *parallelism*, *nested loops*, and *nested conditional checks* are identified as beneficial patterns that are often found in functionally correct code [17].

Considering text-based programming in university settings, evaluating student projects led to identifying several quality issues. Using Java, the most common code smells were missing blank lines and a quirky usage or omission of parentheses. Considering Python programs, spaces were often omitted after the comment character or lines contained too many characters [11]. Overall, the correction rate for quality issues among students is low, with many topics going unnoticed or unaddressed even when using tools designed to detect them [12]. The comparison of static quality properties between first and second-year college students revealed that second-year students exhibited improvements in certain aspects, such as using shorter functions and fewer very short variable names, compared to their first-year counterparts. However, they also tended to write more complex code and incorporate a higher level of statement nesting within methods. Despite these differences, there was no significant improvement in the overall code quality of second-year students compared to first-year students. This indicates that code quality skills only improve by explicitly addressing them [2].

In summary, considerable research has been conducted on code quality in novice programming education, primarily focusing on university-level settings. Considering K-12, existing research in this domain predominantly centers around block-based programming languages such as Scratch. As a result, there is a need

for further investigation and exploration of code quality aspects, specifically in text-based programming within K-12 educational contexts.

3 Methodology

To address the research gap described above, we aim to investigate the source code of upper secondary students in this study. To gain a comprehensive understanding of the issues posed by code smells in schools, our approach diverges from a mere examination of their frequency. Rather, we strive to delve deeper, investigating the extent of the impact caused by these code smells. To this end, we address the following research question: *How do code smells impact the comprehensibility and modifiability of source code of programming novices in upper secondary school?*

3.1 Sample

To answer our research question, we analyzed a total of $N = 12$ source codes created by students as part of group work projects towards the end of the school year. The projects were collected from different German High Schools: four were created by year 11 students (P1 to P4) and eight by year 10 students (P5 to P12). At this point, students have one or two years respectively of programming experience in a text-based language. All projects implement small games programmed in Java using BlueJ, a widely utilized development environment designed for novices. The source codes from the 11th-grade class represent intermediate versions of their respective projects. However, it is noteworthy that all projects can be executed without errors. The length of the source code ranges from 161 to 735 non-empty lines. The mean is 325.5 lines of code, with a median of 292 lines.

3.2 Data Analysis

We conducted a structured qualitative content analysis according to [13] to analyze the data. We used a deductive category system based on various catalogs for code smells. However, due to the relatively small and less intricate nature of projects at this level, it is worth noting that several of the defined code smells aimed at professional development are not applicable in the school context. Based on these considerations as well as findings in related work, we have chosen the following set of code smells from [4]: *Long Function*, *Duplicated Code*, *Comments*, *Mysterious Name* and *Speculative Generality*. Furthermore, we included the code smells *Class Data Should Be Private* [18] and *Code Formatting* [20] in our analysis. Table 1 lists the complete category system.

To detect these code smells, we have established specific thresholds that we consider meaningful for school projects with limited scope and complexity. A function, therefore, is a *Long Function* if it exceeds 20 lines of code. In the case

of *Duplicated Code*, at least three lines must be duplicated, even if the variable names differ. Regarding *Comments*, they are flagged as code smells if they merely describe the code without providing any additional information, in line with the definition by [4]. However, commented-out lines of code are not considered code smells. The evaluation of the *Mysterious Name* code smell is based on the Oracle Naming Conventions¹. However, the distinction between uppercase and lowercase letters is not taken into account, as it does not significantly impact the code's comprehensibility or modifiability. Some analyzed projects are incomplete, so the code smell *Speculative Generality* refers only to unused attributes, variables, and parameters. Guidance on capturing the code smell *Code Formatting* is provided by the criteria for Java from Checkstyle².

Table 1. Final Category System

name	description
<i>Long Function (LF)</i>	A method has huge size.
<i>Duplicated Code (DC)</i>	A code section is included multiple times.
<i>Comments (C)</i>	A comment is superfluous.
<i>Mysterious Name (MN)</i>	The name of a variable, class, or method is not self-explanatory.
<i>Speculative Generality (SG)</i>	A part of the code is not called.
<i>Class Data Should Be Private (CDSP)</i>	A class exposes its attributes.
<i>Code Formatting (CF)</i>	The formatting of the code is not clear.

The detection of code smells was automatically done using the IDE IntelliJ³. This involved using the integrated code inspection and analysis, as well as the Checkstyle⁴ and Statistic⁵ plugins. If necessary, manual evaluation was performed when automated detection was not feasible. For all individual code smells, we evaluated their actual impact on the modifiability and comprehensibility of the source code individually based on a qualitative interpretation in the context of the related project. Given the difficulty and subjective nature of measuring these aspects, in the following, we describe in detail our interpretation and reasoning resulting from discussions in our research group.

4 Results

To first of all provide an overview of the smells in all projects, see Table 2. We employed two metrics to measure the number of detected code smells. For

¹ <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>

² <https://checkstyle.sourceforge.io/checks.html>

³ <https://www.jetbrains.com/de-de/idea/>

⁴ <https://plugins.jetbrains.com/plugin/1065-checkstyle-idea>

⁵ <https://plugins.jetbrains.com/plugin/4509-statistic>

code smells that tend to increase with code length, we express the frequency as occurrences per 100 lines (1). For code smells specific to a particular program element, we present the proportion of code smell occurrences relative to the total number of program elements of the corresponding type (2) [21].

Table 2. Frequencies of code smells by projects studied. “-” marks that no comments exist in the code, while 0 means that none of the dedicated program elements is considered a code smell.

	metric	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
<i>LF</i>	(1)	18	9	27	23	41	17	11	41	0	24	35	40
<i>DC</i>	(1)	0	3	3	2	5	22	3	0	0	8	23	0
<i>C</i>	(2)	72%	82%	30%	90%	-	100%	-	100%	100%	-	100%	100%
<i>MN</i>	(2)	8%	30%	23%	13%	29%	41%	3%	50%	28%	8%	29%	36%
<i>SG</i>	(1)	8	10	5	8	22	5	7	3	8	10	5	0
<i>CDSP</i>	(2)	40%	75%	40%	33%	0%	58%	10%	72%	100%	0%	0%	0%
<i>CF</i>	(1)	60	54	72	76	40	189	21	11	25	42	47	40

Long Function 22 methods were identified as *Long Function*. Half are slightly above the threshold and, therefore, less critical concerning comprehensibility and modifiability than the others. However, one method in particular (P8) spans 120 lines, accounting for 34% of the total code. This code lacks structure and clarity, making it prone to complications during modification. Notably, many very long functions are responsible for updating the user interface. Particularly conspicuous is such an expansion of functions in the projects of year ten students. Additionally, constructors (P2, P4, P10, P11) are particularly susceptible to becoming lengthy. Another factor contributing to long functions is the presence of if-statements (P11) which have empty branches. Furthermore, extensive if-statements with multiple cases can also contribute to the length of functions, but they remain readable due to their structured nature (P7).

Finding: Functions only slightly above the threshold do not pose significant limitations on comprehensibility and modifiability. However, in our data, some other functions lack a clear structure due to their excessive length and consequently impact comprehensibility and modifiability.

Duplicated Code Overall, a relatively low number of *Duplicated Code* instances was detected. The longest duplicate consists of only 11 lines (P11), and many duplicates are slightly above the threshold (P3, P4, P7, P10), indicating no significant issue regarding comprehensibility. The duplicates could be easily refactored in several cases by extracting them into separate functions (P3, P4, P5, P6, P11) to enhance the code’s comprehensibility and structure. Additionally, adding parameters could combine some methods (P2, P6). In one outstanding example, three functions perform the same task but have different names (P6). Modifying one function requires changing the other two as well. Another critical

instance of duplication arises from identical function calls within an if-statement under different conditions (P10), with the only difference being the passed argument (in this case, the color of an object as a String). Changing the method name would force multiple changes in the condition branches. One *Duplicated Code* smell requires advanced knowledge for refactoring. It involves code sections in four different classes (P6), where the constructors share identical code and contain another method with the same code. As already mentioned above, one change would entail several modifications.

Finding: *Duplicated Code* is relatively scarce overall, and when it is, it only slightly complicates comprehensibility. Concerning modifiability, only some critical code smells could mostly be addressed by simple refactoring.

Comments In projects P5 to P11, only a few comments are used. The existing comments are obviously from code templates that have not been customized and, therefore, cannot be considered beneficial and thus count as a code smell. However, it is worth appreciating the inclusion of author information and the providing short and concise descriptions of classes and functions, which helps in collaboration (P3). Comments that solely describe the functionality of a method, such as “closes the database connection” preceding a function named *ConnectionClose* (P1), do not offer significant additional support for comprehension and are consequently considered as a code smell. In some projects, every line of code was provided with a comment explaining the corresponding line (P2). This leads to a rather unattractive and difficult-to-survey picture of the code. Another issue that could be observed is that *Comment* code smells were also apparently created in the comments by copying them, causing incorrect and useless information (P2). Furthermore, in an if-statement, comments repeating the conditions were identified (P4). This information is redundant for the reader and makes it difficult to read.

Finding: Comment usage among students varies greatly. While some students refrain from using comments, others use them excessively without adding much value for comprehensibility.

Mysterious Name This code smell is present in every project. Generally, a mixture of German and English is commonly used, which only somewhat impacts comprehensibility. However, poor naming can negatively affect comprehensibility, even though its impact on modifiability is not crucial. Overall, the methods are mainly named understandably. It should be noted that we have not extensively investigated the functionality of the methods, and therefore may be discrepancies between the naming and the actual functionality. However, method names *niceMethod* (P10) and *addtt* (P2) do not provide any indication of their functionality. Regarding class names, the only related smells found in the data are the designations *LE* and *MyKeyAdapter* (P2). The variable naming reveals a contrasting scenario as it often lacks meaningful names. Despite being short, these names only sometimes convey the variable's functionality effectively to observers. Frequent usage of single-letter designations can lead to confusion, mainly

when numbering is utilized for differentiation (P1, P2, P3). Numbering is predominantly done for naming user interface objects (P5, P8, P11, P12) obscuring the association between specific objects and their corresponding graphical task. Additionally, certain abbreviations, which should ideally be avoided according to naming conventions, are frequently employed in numerous projects. Some of them, such as *koord* (P3), *min* (P6), or *xpos* (P2), can be derived by a reader and can be considered less critical. In contrast, there are also less comprehensible namings such as *sadpicm*, *sadpicq* (P6), *tt* (P2), or *LHor* (P9).

Finding: Most functions and classes have expressive names, enhancing code comprehensibility. However, variables often need more precise naming, relying on abbreviations or numbering that hinder code comprehensibility and impair collaboration. Nevertheless, this does not directly impact modifiability.

Speculative Generality In general, it is worth mentioning that the projects examined had relatively few unused components. As a result, these unused components only have a minor impact on the code comprehensibility and modifiability. Among these, unused parameters and attributes are the least critical, as they do not disrupt the overall flow of the program. However, they can introduce confusion when unused attributes result from the presence of local variables of the same type within class methods (P2, P6). Likewise, initializing unused local variables (P3, P6, P9, P10) can hinder comprehensibility. In one project (P10), an integer variable is declared before a for-loop, seemingly meant to serve as a constraint for the loop's iteration. However, the hard-coded value is used within the loop instead of the variable. This particular example may not be of utmost importance, but it has the potential to cause some irritation.

Finding: The code smell *Speculative Generality* is generally a minor issue that has a limited impact on code comprehensibility and modifiability. Solely, unused local variables are more likely to affect code comprehensibility negatively.

Class Data Should Be Private Four of the examined source codes do not exhibit inappropriate data encapsulation. However, in other projects (P1, P6, P8, P9), more than half of the attributes raised concerns as code smells. Some classes contain either only *public* attributes (P8) or do not include any attributes marked with the *private* access modifier (P9). In most cases, the attributes lacked an explicit access modifier, making them accessible to other classes within the same package. It is important to note that the investigated projects did not utilize packages, resulting in all classes having access to these attributes. The code smell *Class Data Should Be Private* does not directly contribute to poor code comprehensibility or pose challenges for modifiability.

Finding: The improper usage of access modifiers is typically a minor and isolated issue that does not significantly impact the comprehensibility or modifiability of the code itself.

Code Formatting Significant instances of poor *Code Formatting* were detected in all the analyzed source codes. Remarkably, in one particular project, there were approximately 189 violations of Checkstyle rules per 100 lines of code. The

most common violations in all projects involve missing spaces around operators, although these have a relatively minor impact on the overall code comprehensibility. Violations such as missing spaces after a comma or between the method name and parameter list are also relatively unproblematic. However, instances of curly braces placed on the wrong line are more significant for maintaining a clear code structure, frequently occurring in the analyzed source codes. Although consistent misplacement may not severely impact comprehensibility, it can lead to challenges when making modifications. Similarly, the absence of optional curly braces, although less frequent, is critical for code modification. Omissions and misplacement of braces can result in errors when inserting code in the wrong location, consuming substantial time to identify.

Finding: Cluttered and inconsistent code formatting reduces comprehensibility and poses a risk for future modifications.

5 Discussion

In this study, we qualitatively analyzed students' source code to investigate how selected code smells influence the comprehensibility and modifiability of code written by novice programmers. To this end, we used a selection of typical code smells from professional software development. Although we have great differences in the scope and aims of programming projects in K-12, using those professional patterns is a typical approach for analyzing students' projects. In line with this, our results indicate that certain smells, such as *Mysterious Name* and *Code Formatting*, are severe issues in the students' code. However, others, such as *Class Data Should Be Private*, do not seriously affect the comprehensibility and modifiability of the high-school students' projects.

Consequently, we consider addressing the problem of *Mysterious Names* in the classroom as highly relevant, mainly when students work collaboratively in teams. Notably, the prevalence of poorly named variables is not limited to specific programming languages, as similar problems can be observed in Scratch projects [16].

Comments can help in keeping an overview, especially when working on a project over a longer period of time. In addition, in team settings, good comments are crucial for collaboration. Interestingly, our analysis revealed that several comments in the code lack meaningful or new information, appearing to be included solely so that the code contains comments, possibly to comply with specifications. Following the agile principle using code as documentation, excessive comments could be reduced, thus well-chosen variable and method names, thereby addressing the *Mysterious Name* code smell.

We were surprised to find a high occurrence of cluttered *Code Formatting* in all the projects because a Checkstyle-plugin for BlueJ is available. BlueJ also provides automatic formatting features for consistent and clean code. Although missing blank lines and parentheses were frequently observed in other studies [11], we found missing parentheses were less common in our results. Instead,

missing spaces and the position of curly braces were more prominent issues affecting code quality.

Concerning the code smell *Speculative Generality*, we hypothesize that in many cases, it can be attributed to oversight or carelessness, where elements were not removed when they became unnecessary. Interestingly, contrary to findings in Scratch projects [1], the results of this study suggest that unused code may be considered less problematic.

Our findings regarding the code smell *Duplicated Code* also differ from previous results, specifically with studies focusing on frequently occurring code duplication in Scratch projects [1, 16]. Several factors, including the age of the students, the distinction between text-based and block-based languages, and the presence of an online repository for Scratch projects, could potentially account for these disparities. However, despite the ease of resolving the detected duplicates and their minimal impact on comprehensibility and modifiability within our data, addressing the issue of redundant code in school can still be valuable, particularly in collaborative group projects. Looking at the individual smells, it becomes apparent that the sense of certain smells, such as *Speculative Generality* and *Duplicated Code*, might be difficult for students to grasp, particularly in understanding their impact on code modifiability. This difficulty can be attributed to the relatively short and limited scope of school projects, which may not provide a broad context for students to grasp the significance of modifiability.

We want to emphasize that a high frequency of a code smell does not always indicate a high impact in our specific context. For example, the occurrence of *Mysterious Names* is less frequent than misplaced *Comments*, but it has a higher impact on comprehensibility. However, the effect of inappropriate comments may vary depending on the specific case.

5.1 Limitations

We used a qualitative approach to gain deep insights into selected code smells and assessed their impact individually. Consequently, our data and results are not necessarily representative for the target group. Additionally, it is worth noting that assessing the impact on comprehensibility and modifiability is inherently subjective. Furthermore, we lack direct insight into the programming classes and have no access to information regarding the guidance teachers provide. Therefore, we have limited information regarding conventions and evaluation principles presented to the students, making it challenging to draw conclusions about the reasons for occurring code smells.

6 Conclusion

In this study, we investigated code quality in novice programmers' projects, specifically focusing on how code smells impact code comprehensibility and modifiability. To this end, we conducted a qualitative content analysis of group projects of novice programmers. Our results show that several code smells occur

in students' source code. For example, *Mysterious Name* and *Code Formatting* are regarded as severe issues in the students' code, especially when working together on group projects. In our data, other code smells, like *Class Data Should Be Private* and *Duplicated Code*, do not seriously affect comprehensibility and modifiability.

The findings of this study contribute to our understanding of the impact of code smells on collaborative programming projects in K-12 computing education classes. To broaden our subjective view in further research, it could be interesting to evaluate the students' perspective on the code quality of their projects and investigate whether and how code quality depends on the group size. Furthermore, our results raise questions about how to effectively address them in the classroom and communicate their importance to K-12 students. One approach might be to incorporate activities that promote code quality, such as refactoring, using a linter, or employing an appropriate code-evaluation rubric. By integrating such activities into teaching, e.g., a specific phase for refactoring within project-based learning, students can gain hands-on experience in improving code quality that supports them, particularly in collaborative settings. Additionally, these activities might provide a valuable opportunity for novice programmers to enhance their programming competencies as they delve into the code and actively work towards improving its quality.

References

1. Aivaloglou, E., Hermans, F.: How kids code and how we know: An exploratory study on the scratch repository. In: Proceedings of the 2016 ACM conference on international computing education research. pp. 53–61 (2016)
2. Breuker, D.M., Derriks, J., Brunekreef, J.: Measuring static quality of student code. In: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. pp. 13–17 (2011)
3. Buse, R.P., Weimer, W.R.: Learning a metric for code readability. *IEEE Transactions on software engineering* **36**(4), 546–558 (2009)
4. Fowler, M.: *Refactoring, Improving the Design of Existing Code*. Addison-Wesley, 2 edn. (2019)
5. Frädriich, C., Obermüller, F., Körber, N., Heuer, U., Fraser, G.: Common Bugs in Scratch Programs. In: Giannakos, M., Sindre, G., Luxton-Reilly, A., Divitini, M. (eds.) Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. pp. 89–95. ACM, New York, NY, USA (2020)
6. Fraser, G., Heuer, U., Körber, N., Obermüller, F., Wasmeier, E.: Litterbox: A linter for scratch programs. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). pp. 183–188. IEEE (2021)
7. Gugerty, L., Olson, G.: Debugging by skilled and novice programmers. *SIGCHI Bulletin* **17**(4), 171–174 (apr 1986)
8. Hansen, M., Goldstone, R.L., Lumsdaine, A.: What makes code hard to understand? arXiv preprint arXiv:1304.5257 (2013)
9. Hermans, F., Aivaloglou, E.: Do code smells hamper novice programming? a controlled experiment on scratch programs. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC). pp. 1–10. IEEE (2016)

10. Hermans, F., Aivaloglou, E.: Teaching software engineering principles to k-12 students: a mooc on scratch. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET). pp. 13–22. IEEE (2017)
11. Karnalim, O., Chivers, W., et al.: Work-in-progress: Code quality issues of computing undergraduates. In: 2022 IEEE Global Engineering Education Conference (EDUCON). pp. 1734–1736. IEEE (2022)
12. Keuning, H., Heeren, B., Jeurig, J.: Code quality issues in student programs. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education. pp. 110–115 (2017)
13. Mayring, P.: Qualitative content analysis: A step-by-step guide. Sage (2022)
14. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Habits of programming in scratch. In: Proceedings of the 16th ACM conference on Innovation and technology in computer science education. p. 168–172. Association for Computing Machinery, New York, NY, USA (2011)
15. Mistrik, I., Soley, R.M., Ali, N., Grundy, J., Tekinerdogan, B.: Software quality assurance in large scale and complex software-intensive systems. Morgan Kaufmann (2016)
16. Moreno, J., Robles, G.: Automatic detection of bad programming habits in scratch: A preliminary study. In: 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. pp. 1–4. IEEE (2014)
17. Obermüller, F., Bloch, L., Greifenstein, L., Heuer, U., Fraser, G.: Code Perfumes: Reporting Good Code to Encourage Learners. In: Berges, M., Mühling, A., Armoni, M. (eds.) The 16th Workshop in Primary and Secondary Computing Education. pp. 1–10. ACM, New York, NY, USA (2021)
18. Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A.: Do they really smell bad? a study on developers’ perception of bad code smells. In: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 101–110. IEEE (2014)
19. Stegeman, M., Barendsen, E., Smetsers, S.: Towards an empirically validated model for assessment of code quality. In: Simon, Kinnunen, P. (eds.) Proceedings of the 14th Koli Calling International Conference on Computing Education Research. pp. 99–108. ACM, New York, NY, USA (2014)
20. Stegeman, M., Barendsen, E., Smetsers, S.: Designing a rubric for feedback on code quality in programming courses. In: Proceedings of the 16th Koli Calling International Conference on Computing Education Research. pp. 160–164 (2016)
21. Techapalokul, P., Tilevich, E.: Understanding recurring quality problems and their impact on code sharing in block-based software. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 43–51. IEEE (2017)

Supporting Non-CS Teachers with Programming Lessons

Svetlana Unkovic^[0009–0004–5538–2971] and
Martina Landman^[0000–0002–0274–4172]

TU Wien, Karlsplatz 13, Vienna, Austria

Abstract. The introduction of a new compulsory subject in secondary schools and the publication of the syllabus in Austria poses great challenges for many teachers. A crucial point is the fact that the selected teachers often do not have an adequate degree in computer science or similar training. Consequently, the lack of their CS competencies often leads to an insufficient teaching, as there were not specifically trained enough for this subject before its introduction. Our outreach programme “eduLAB” offers an initiative where teachers in schools are supported by student staff in teaching programming. To evaluate this initiative, 28 teachers were asked how our programming courses and offerings could be adapted and expanded to support teachers and also students in the new compulsory subject. This paper analyses the survey results and ideas of what we can do to further support teachers in this situation.

Keywords: programming course · Programming with Processing · outreach to school teachers · CS school curriculum · teacher training.

1 Introduction

In Austria, a rethinking of digital education and computer science education resulted in the replacement of the pre-scheduled exercise by the compulsory subject “Basic Digital Education” (BDE) in 2022, which combines aspects of media education and computer science. One hour per week is dedicated to this subject from the 5th to the 8th grade.

The number of teachers trained in the field is by far not sufficient to fill the open positions, which means that many teachers have to fill in without the proper training and background. We are interested in a first assessment of the teachers’ own view on the matter. Based on existing literature [7] we hypothesise that teachers feel insecure about teaching the subject area. Moreover, we aim to answer the following two research questions:

RQ1. How do teachers who teach the compulsory subject “Basic Digital Education” in secondary schools rate their prior programming knowledge?

RQ2. How can our programming course for 9th grade be adapted to make it useful for the compulsory subject and thus support teachers?

2 Related Work

The importance of teaching computer science in lower secondary education is being recognized worldwide and included in existing curricula or new school subjects, as well as the early implementation of this curriculum [12]. Introducing programming into curricula is also advocated by official guidelines such as the Informatics Curriculum or the new subject “Basic Digital Education” [2].

One example of a programming initiative is Scratch¹, which provides visual block programming as a learning tool and guided projects as teaching materials. Another example is Code.org², offering coding resources, curriculum materials, and professional development programs for teachers. There are many more online initiatives that offer teaching materials as well as teacher training courses (e.g. [coderdojo.com](https://www.coderdojo.com), codeclub.org, codeweek.eu, etc.). In contrast, our approach is to support teachers directly in their classroom and to provide additional teaching materials. A good example of a successful introductory programming course is shown by Porter and Simon at UCSD, which focused on the three aspects pair programming, peer instruction and media computation[9].

However, new content in the curriculum leads to new teacher responsibilities. Especially if they lack adequate training for new curriculum content, they may struggle to support their students and feel overwhelmed [7]. A study, which compared teachers’ attitudes towards CS skills, discovered a shortage of adequately trained teachers who require assistance in teaching programming skills [13]. The need of better training is also highlighted in [11]. Additionally, the new curriculum seems cryptic according to teachers without proper CS education (see 3.2). The importance of including computer science in national secondary school curricula underline the need of our approach to support untrained teachers in teaching programming.

Furthermore, we used Processing [10], a programming language known for its extensive visual representation capabilities, into our course to enhance the visualization of programming concepts. Starting from our Processing course, originally designed for university students, we adapted and tailored it for upper secondary school classes [5]. Based on teachers’ support, we expect to see greater interest in computer science and a better understanding of programming, which can be reinforced through such outreach activities [4].

3 Structure and Content of the Compulsory Subject

The aim of the introduced subject is to provide future adults with early educational and professional opportunities as well as private advantages over so-called “digital illiterates” [6].

¹ <https://scratch.mit.edu/>

² <https://code.org/>

3.1 Syllabus

A look at the syllabus reveals that the compulsory subject is built on the five competence areas “orientation”, “information”, “communication”, “production” and “action” [2].

- **Orientation:** “... analysing and reflecting on social aspects of media change and digitalization.”
- **Information:** “... dealing responsibly with data, information and information systems.”
- **Communication:** “... communicating and cooperating by using information and media systems.”
- **Production:** “... creating and publishing digital content, designing algorithms and programming: Decomposing problems, recognising patterns, generalising/abstracting and designing algorithms.”
- **Action:** “... assessing offers and possibilities for action in a world shaped by digitalisation and using them responsibly.”

In addition, the content is categorized in technological, social, and interactional domains. These perspectives are based on the “Frankfurt Triangle” [1] which purpose is to interdisciplinarily guide and structure educational processes in digital transformation, involving all relevant disciplines.

For the sake of simplicity, they can be described by three questions (**T**) “*How do digital technologies work?*”, (**G**) “*What are the social interactions that result through the use of digital technology?*” and (**I**) “*What are the options for interaction and action for pupils?*”.

In order to close the circle between the introduced model and the five competence areas, we want to show their connection: Each competence area is subordinated to the three presented perspectives *T*, *G* and *I*. It must be added that a competence area does not have to be limited to only one description of a perspective. On the contrary, several descriptions from one perspectives can be found in a competence area. In addition, some competence areas contain areas of application that can be used for teaching, but by far not all areas are supported with these suggestions.

3.2 Remarks and Criticism on the New Syllabus and Realisation

By studying the syllabus closely, including individual discussions with teachers during our (programming) workshops and their feedback on teaching at school, it becomes apparent that the strong generalisation and cryptic description of certain sub-areas *T*, *G* and *I* can quickly lead to perplexity and confusion in the preparation of the lessons. The abstract explanations and lack of examples and descriptions in the fields of application, if they even exist, make designing the teaching units an obstacle.

In addition, the quick introduction of the new compulsory subject leads to open teaching positions which need to be filled. Due to the shortage of teachers non-specialist teachers without training or background in computer science are

therefore obliged to teach this subject. Particularly young teachers are affected, as we have learned from informal conversations among young teachers that they are often presumed by older colleagues having the know-how and experience. Still the repertoire of knowledge in teaching is missing and by many older colleagues not considered. From this, the affected teachers face the problem of not knowing which and how they can adequately convey certain contents in class. Additionally, it has to be taken in account that they often have to become familiar with the contents themselves. This problem is also pointed out in other literature [14]. To support teachers, Austria’s Federal Ministry provides offers at the University College of Education and a MOOC “Basic Digital Education” [3] as part of a continuing and further education programme. The question has to be raised whether it is possible to pack an entire university course or a teacher training programme into such a framework. There is as well an offer to an existing university course “Teacher of Basic Digital Education”³ which corresponds to a duration of two years.

This gives us hope and future outlook that one day qualified persons will fill these places through this university course. Unfortunately, due to shortage of study places in this course and teachers in general, this initiative and approach could turn into a rather unrealistic concept. Nevertheless, our research group wants to support, reach out to those teachers who are lacking in programming, and also provide remedial support in the future.

4 Setting

We are offering a short online programming course for teachers and young students alike, originally targeted at 9th grade students. The course is based on and uses Processing⁴ as a programming language. The Austrian curricular guidelines demand, however, that programming be taught in 8th grade (the guidelines regarding programming are shown in table 1).

Level	Excerpts from the syllabus
8th grade	(T) “Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.”
	(I) “cooperate with provided media and software applications in a purposeful and creative way
	(I) “create simple programs or web applications using appropriate tools to solve a specific problem or accomplish a specific task.”

Table 1: Overview of the competence area “Production” from the curriculum of the subject “Basic Digital Education”.

³ <https://www.ph-noe.ac.at/de/weiterbildung/hochschullehrgaenge/>

⁴ <https://processing.org/>

We have therefore begun to modify our original course for 9th graders to better fit the target age group of 8th graders. Additionally, there is a number of teachers who took the course with or booked it for their students. The course’s contents are briefly discussed in the following subsections.

4.1 Programming Crash Course as a Workshop in Schools

Our free programming course focused on “Computer Science”, aligning with the syllabus objectives from 9th grade onwards. A year ago, we restructured our previous workshop into a four hour programming “crash course”. In this programming workshop, pupils have the opportunity to get a taste of the Java-like programming language Processing. Since the language allows graphic output right from the start, the students can quickly see results and experience their first successes in programming.

The workshop is run by our student staff and tutors in schools and, in exceptional cases, online and is designed for two double sessions⁵. The flexibility makes the workshop very popular, depending on the choice only one double lesson can be used or both. The workshop implementation follows a team-teaching format with the teacher. There is always a short introduction to the programming concepts, followed by a free working session. However, the hands-on session is always longer than the theoretical and practical introduction by the workshop leaders. In the practical section, the students have the opportunity to work in groups and support each other.

Our tasks and materials are accessed via a website link provided by our staff during the first session. In addition to the tasks, the students receive a “cheat sheet” that lists the most important Processing commands, and a student-friendly and adapted script with all important explanations from our MOOC course created for first-year university students [5]. Moreover, we provide solutions on request with uniform commentary notation, as well as a detailed description of the program sequence.

Looking back to the periods between March 2022 and May 2023, our offer was used by five schools, including eight groups with 187 pupils. Seven of these groups were in the 9th grade and one group in the 8th grade, of which five groups of the 9th grade opted for the four hour programming block. In the 8th grade, the workshop leaders noticed that the students needed more time to understand certain programming concepts compared to the 9th graders. Therefore, only the first double session was conducted. Nevertheless, it must be said that starting from one supervised group, it is not possible to draw conclusions about all 8th graders. However, since the subject BDE only lasts until the 8th grade and programming is part of the syllabus, we consider offering a slimmed-down version of the course for the 8th grade or possibly introduce another environment, such as microworlds [8], which would then be prepared on the basis of our programming

⁵ The duration of a session corresponds to 50 minutes. CS classes are usually held in double sessions.

crash course from the 9th grade onwards. For this purpose, we conducted a questionnaire to find out the needs of the teachers who teach BDE.

Between December 2022 and May 2023, five teachers, including one outside the country, requested access to the website and to the Moodle course which contains more content, in total 11 lessons [5].

4.2 Comparison of the previous and new version of the course

We noticed that certain contents were not well received by the students, so we skipped contents such as “defining and working with own variables”, “compound comparisons in branches” or “arithmetic operations” from our workshop program. Instead, we have included or adapted materials and tasks that focus on understanding the concepts, such as “calling methods”, “predefined variables”, “loops” and “branches”. The changes in the course can be seen in table 2 and 3.

	OLD VERSION	NEW VERSION
access	<ul style="list-style-type: none"> ▪ moodle course with multiple duplicated course ▪ login data needed 	<ul style="list-style-type: none"> ▪ website ▪ without login data
	block 1 (two sessions)	
	“basic shapes”	“basic shapes and pre-defined variables”
content	<ul style="list-style-type: none"> ▪ Programming environment <ul style="list-style-type: none"> - Processing surface - orientation in the sketch window: coordinate system - comments and indentations of the code - difference between <code>setup()</code> and <code>draw()</code> - first commands with <code>rect()</code>, <code>ellipse()</code> and <code>triangle()</code> ▪ Pre-defined variables: <ul style="list-style-type: none"> - <code>height</code>, <code>width</code> - <code>mouseX</code>, <code>mouseY</code> ▪ Own variables <ul style="list-style-type: none"> - <code>color</code> 	

Table 2: Comparison of the old and new version of the course - Part 1.

5 Data collection and methodology

We have conducted a survey among the first group of teachers who worked with our programming course and teachers who are teaching BDE. Some of

block 2 (two sessions)		
“variables and branches”		“pre-defined variables and branches”
content	<ul style="list-style-type: none"> ▪ own variables <ul style="list-style-type: none"> - color - boolean ▪ simple branches and compounded branches ▪ functions 	<ul style="list-style-type: none"> ▪ pre-defined variables ▪ simple branches
	changes	
		<ul style="list-style-type: none"> ▪ deletions in the content ▪ simple explanations and connecting to previous knowledge of the students ▪ from three to two basic tasks in each content block ▪ more creative tasks in the additional task section ▪ adjustments to the scripts ▪ no introductory videos ▪ removed sections like “own variables” and related tasks

Table 3: Comparison of the old and new version of the course - Part 2.

the teachers who participated do not have a formal training in computing as such. In total, we received answers from 28 participants (11 female and 17 male teachers), all of whom teach grades 5 to 12. The survey was mostly multiple-choice-questions (MCQ) or Likert-scale questions (from 1 to 5). The survey’s structure is shown in table 4.

Similar research in this area shows that the lack of inclusion of certain CT perspectives in computer science classes is due to a lack of prior programming knowledge and non-existent CT knowledge [14]. Learning motivation and fear of failure are cited as the main reasons and the biggest challenge. With our questionnaire we want to check whether teachers really need support, especially in the area of programming, and if so, which measures need to be set.

The online questionnaire was carried out in spring 2023. We used various channels for dissemination, such as our newsletter or contacting schools that attended our workshops. The questionnaire contains 26 questions that were grouped in five categories. Bifurcations were built into the questionnaire. Some sections could be skipped, depending on whether someone had taken the programming crash course before or not. This way, BDE teachers could be filtered out from CS teachers. However, the focus is on BDE teachers and how they can be guided by our research group in terms of programming. The remaining categories serve the general goal of improving our existing programming course.

Category 1	Demographic data of the participants
The gender, professional experience and teaching subjects of the participants were recorded at the beginning of the questionnaire.	
Category 2	Digital education
<ul style="list-style-type: none"> – assessment of the infrastructure at the school – access to learning materials and further and in-service training – reasoning for teaching the subject "Digital Basic" 	
Category 3a	Our Outreach programme
– participation in the offers of the programme (e.g. unplugged workshops)	
Category 4	Own programming experience
Objective 1	Prior programming knowledge of the teachers.
<ul style="list-style-type: none"> – previous knowledge and self-assessment in programming – repertoire of programming languages – motivation for participating in the programming 	
Category 5	Programming course Processing
Objective	Feedback and attempt at ongoing improvement of materials and implementation.
<ul style="list-style-type: none"> – preparation of the contents in the first and second programming block – difficulties encountered in programming and programming concepts by learners – suggestions for improvement – continuation with the programming language in the classroom and use of the Moodle course 	
Category 3b	Appeal of the programming Crash Course
Objective 2	Support BDE teachers through existing programming course (with customisation) or other programming opportunities.
<ul style="list-style-type: none"> – naming the reasons for participation – desires for other programming languages or microworlds – interest in further education and training 	

Table 4: Overview of the categories and the associated content of the questionnaire.

6 Results

In total, 28 teachers answered the survey, of which 19 teach BDE. Of these 19 participants ten answered to have a degree in computer science or similar training. The most common motivations for teaching BDE were *related subject*, *professional aptitude* and/or *assignment by others* (e.g. principal).

For *related subjects*⁶, the combinations were often mathematics, followed by other subjects like English or history. *Voluntary reporting* was predominantly coupled with *professional aptitude*, among those volunteers only two teachers did

⁶ In Austria, the teacher training programme is linked to the choice of two (or more) subjects, which enables teaching in Austrian schools. The choice of subjects can be made independently, i.e. they do not have to be related at all.

not have proper training and came forward due to strong interest in imparting knowledge to students. Those who were assigned by others to teach BDE, did not have more than 10 years of professional experience in teaching which supports the assumption that young teachers are assigned more commonly. But this small data set cannot be used to draw conclusions about the general public. Nevertheless, the result is a good indication of an existing problem in the school setting.

More than half of the participants, which corresponds to almost three quarters of the BDE teachers, did not take part in any of our offers, but heard about our initiative through colleagues or self-referrals. In terms of previous programming experience and knowledge, one third of BDE teachers claim to have good to excellent skills. (1) Python, (2) Java and (3) C#/C++ were the most used programming languages among all our participants and are ranked by frequency. 57% of all participants state to have little to none experience at all. Slightly more than half of them are BDE teachers and the rest correspond to CS teachers. On closer inspection, these CS teachers have more than 10 years of professional experience and probably do not have a computer science degree, but attended only a university course, see figure 1.

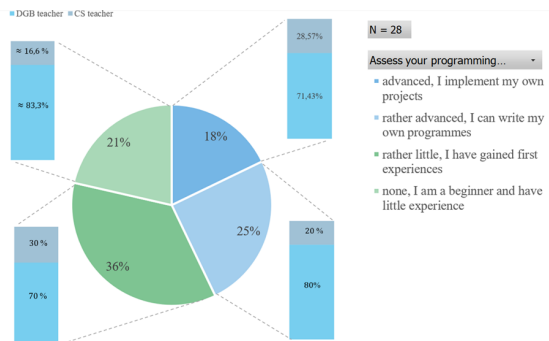


Fig. 1: Overview of programming experience of all participants.

16 out of 28 participants have little to none experience in programming, who we want to pay more attention to and reach with our programming course. For this purpose, six teachers, who already attended our course, were filtered during the questionnaire via branches. Two of these six teachers chose and completed the entire programme of our course with their students. The remaining teachers based it on the two sessions (= block 1). In the first block section the opinions on the categories *introduction to the environment* and *method selection*, e.g. `rect()` or `triangle()`, were split, with the first half describing as *suitable* and by the other as *very suitable*. Nearly 67% of the participants rated the subsection "predefined variables" as *suitable*, but no further comments for improvement were made. (1) *Problems with syntax*, (2) *Understanding error messages and finding errors*, (3) *Saving and finding files* and (4) *Lack of understanding the programme and programme process* were identified as difficult for the students

from the teachers' perspective and are ordered by occurrence. To all the concepts presented, *loops* and *compound comparisons in branches* were described as most difficult. Again only two groups have done the whole programme. A general suggestion for improvement was *interactive videos on the website* for the students to better understand the tasks and *unification and simplification of the provided cheat sheets*.

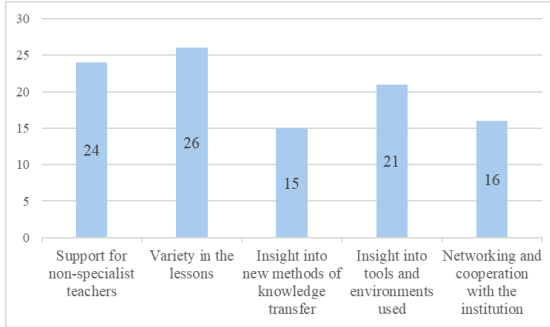


Fig. 2: Overview of programming experience of all participants.

The following items would speak for the attractiveness of the course on the part of the teachers: *Variety in the lessons*, *Support for non-specialist teachers* and *Insight into tools and environments used* (figure 2). The majority of participants would be satisfied with the existing offer and programming language, still 39% suggest a microworld, use of robots or another text-based programming language, such as Python, Rust or C#. 75% would like to receive further education or training from our institution. This can be explained by the insufficient offers, as 26.3% described the category "learning materials" as insufficient and 52.6% as neutral.

7 Discussion

This evaluation aimed to investigate the programming knowledge and needs of teachers who teach the compulsory subject BDE in secondary schools (RQ1) and to explore how we can adapt the programming course, which was originally designed from the 9th grade onwards, to support these teachers and their students better (RQ2).

Regarding the first research question (RQ1), the questionnaire revealed that a significant number of teachers (57%) reported having little to no programming experience. This outcome supports our assumption that there is a potential gap in programming skills among the participants and that those untrained often face the problem in teaching programming in their classes. Furthermore, among those who claimed to have programming knowledge, the most popular

used programming languages were Python, Java and C#/C++, which is an indicator of the general popularity of the languages in today's community and underscores the choice of Processing in our course, as it is Java-like. Regarding RQ2, the questionnaire results shed light on the challenges faced by teachers and their suggestions for improvement in the new version of the programming course.

Difficulties for students included *syntax problems*, *understanding error messages*, *saving and finding files*, and a *lack of understanding of the program and its process*. This results from oral comments made by students and their teachers. Teachers observed that loops and compound comparisons in branches were particularly challenging for the students. Feedback from teachers who had already participated in the course highlighted the need for *interactive videos* on the course website and *simplified cheat sheets*. These insights can guide future improvements to the programming course, addressing the identified challenges and suggested improvements. There were also suggestions for additional enhancements, such as including microworld, using robots, or incorporating alternative text-based programming languages like Python, Rust or C#. These suggestions indicate a desire to further diversify and customise the course content to teachers' and students' specific needs.

The questionnaire also revealed positive aspects of the course that were attractive to teachers, such as the *variety of lessons* or *support for non-specialist teachers*. A very satisfactory result on the course, since it was the goal to reach the teachers that needed it and give students a good first programming experience. However, it is crucial to note that the study's limitations, particularly the small sample size of only 28 teachers, strongly restrict the generalisability of the findings. A larger sample would provide a more representative picture and offer opportunities for future research.

8 Conclusion

In conclusion, this work gave an overview of the structure and content of the newly introduced subject "Basic Digital Education" (BDE) in secondary schools and the associated difficulties faced in teaching, especially in the area of programming. In addition, this paper presented an already existing programming course from 9th grade onwards, which shows great potential to be applied in lower levels. The survey results highlighted a programming skill gap among teachers responsible for teaching BDE classes and identified certain concepts as difficult for students.

The feedback from teachers who had already attended the programming course provided valuable suggestions for improvement. However, the study's limitations, particularly the small sample size, call for further research to validate and extend these findings. Future investigations should aim to include a more extensive and more diverse sample of teachers to obtain a more comprehensive understanding of the programming knowledge and needs of teachers in secondary schools. In the future, we would like to adapt the suggestions for improvement

in the current course and adapt it for the 8th grade, hold further training for CS and BDE teachers with a focus on “programming” and offer programming courses from 5th grade.

References

1. Brinda, T., Brüggem, N., Diethelm, I., Knaus, T., Kommer, S., Kopf, C., Mis-somelius, P., Leschke, R., Tilemann, F., Weich, A.: Frankfurt-Dreieck zur Bildung in der digital vernetzten Welt. Ein interdisziplinäres Modell (2020)
2. Bundesministerium für Bildung, Wissenschaft und Forschung: Änderung der Verordnung über die Lehrpläne der Mittelschulen sowie der Verordnung über die Lehrpläne der allgemeinbildenden höheren Schulen . <https://www.ris.bka.gv.at/eli/bgb1/II/2022/267/20220706> (2022), [Online; accessed 25-May-2023]
3. Bundesministerium für Bildung, Wissenschaft und Forschung: Mini-mooc „digitale grundbildung“. https://www.bmbwf.gv.at/Themen/schule/zrp/dibi/paed/mooc_dgb.html (2022)
4. Lakanen, A.J.: On the impact of computer science outreach events on k-12 students. *Jyväskylä studies in computing* (236) (2016), <https://jyx.jyu.fi/handle/123456789/49729>
5. Landman, M., Futschek, G., Unkovic, S., Voboril, F.: Initial learning of textual programming at school: Evolution of outreach activities. *Olympiads in Informatics* pp. 43–53 (2022). <https://doi.org/10.15388/oi.2022.05>
6. Moritz, T.: Bildung und medienpädagogik im zeitalter der digitalen medien. Probleme, Herausforderungen und Perspektiven für Pädagogik, Bildung und Schule in Zeiten von Internet und Telekommunikation. *Medien Impulse* pp. 51–60 (2001)
7. OECD: Curriculum overload: A way forward. OECD Publishing, Paris (2020). <https://doi.org/10.1787/3081ceca-en>
8. Papert, S.: *Microworlds: Incubators for knowledge*. Mindstorms-Children, Computers and Powerful Ideas pp. 120–134 (1980)
9. Porter, L., Guzdial, M., McDowell, C., Simon, B.: Success in introductory programming. *Communications of the ACM* **56**(8), 34–36 (2013). <https://doi.org/10.1145/2492007.2492020>
10. Reas, C., Fry, B.: *Processing: A programming handbook for visual designers and artists*. MIT Press, Cambridge, Massachusetts (2007), <https://ieeexplore.ieee.org/book/7008153>
11. Sentance, S., Csizmadia, A.: Computing in the curriculum: Challenges and strategies from a teacher’s perspective. *Education and Information Technologies* **22**(2), 469–495 (2017). <https://doi.org/10.1007/s10639-016-9482-0>, <https://link.springer.com/article/10.1007/s10639-016-9482-0#citeas>
12. Vegas, E., Hansen, M., Fowler, B.: *Building skills for life: How to expand and improve computer science education around the world*. Brookings (2021)
13. Wu, L., Looi, C.K., Multisilta, J., How, M.L., Choi, H., Hsu, T.C., Tuomi, P.: Teacher’s perceptions and readiness to teach coding skills: A comparative study between finland, mainland china, singapore, taiwan, and south korea. *The Asia-Pacific Education Researcher* **29**(1), 21–34 (2020). <https://doi.org/10.1007/s40299-019-00485-x>, <https://link.springer.com/article/10.1007/s40299-019-00485-x>
14. Zhang, L., Nouri, J., Rolandsson, L.: Progression of computational thinking skills in swedish compulsory schools with block-based programming. In: *Proceedings of the Twenty-Second Australasian Computing Education Conference*. pp. 66–75 (2020)

A Appendix: Excerpt from the questionnaire

1. Indicate how long you have been teaching:
 - (a) 0 - 1 years
 - (b) 1 year
 - (c) 2 years
 - (d) 3 years
 - (e) 4 years
 - (f) 5 years
 - (g) 5 - 10 years
 - (h) 10 - 20 years
 - (i) 20 - 30 years
2. Please state which subjects you teach:
 - (a) geometry
 - (b) mathematics
 - (c) English
 - (d) German
 - (e) other living foreign language
 - (f) Latin/ancient Greek
 - (g) Computer Science
 - (h) history
 - (i) geography
 - (j) Basic Digital Education
 - (k) other:
3. Why did you decide to teach “Digital Basic Education”? (*Multiple selection possible*)
 - (a) Voluntary reporting
 - (b) Professional aptitude
 - (c) Related subject
 - (d) Classification by other persons (e.g. principal, etc.)
 - (e) other:
4. Assess your programming skills:
 - (a) none, I am a beginner and have little experience
 - (b) rather little, I have gained first experiences
 - (c) rather advanced, I can write my own programs
 - (d) advanced, I implement my own projects
5. If applicable: Which programming languages have you already worked with? (*Multiple selection possible*)
 - (a) Java
 - (b) C#/C++
 - (c) Python
 - (d) Processing
 - (e) PHP
 - (f) other:
6. Please indicate why you would/have chosen the programming crash course (an introduction to programming with a Java-like programming language and comprises 1-2 double lessons):(*Multiple selection possible*)

- (a) Students should get to know a new programming language
 - (b) Processing as an introductory language to programming for students
 - (c) I have little programming experience myself and would like to see how it is implemented.
 - (d) Future cooperation with the institution
 - (e) Part of the syllabus
 - (f) other:
7. Please indicate if you have already booked a Processing course:
- (a) 2 hour session
 - (b) 4 hour session
 - (c) No
8. I found the content in the first 2 hour session suitably prepared:
- (a) Introduction in the environment
 - (b) Method selection (e.g. `rect()`, `triangle()`)
 - (c) Pre-defined variables
9. I found the content in the second 2-hour block suitably prepared:
- (a) Branches - simple comparisons
 - (b) Loops
 - (c) Branches - compound comparisons
10. Indicate whether difficulties were noticeable in your class: (*Multiple selection possible*)
- (a) Dealing with the environment
 - (b) Problems with the syntax
 - (c) Understanding and finding error messages
 - (d) Difficulties in creating programs to solve the tasks
 - (e) Saving and finding files
 - (f) Overstretched by the resources
 - (g) Lack of understanding the program structure
 - (h) other:
11. Please mark which concepts have been noticeably not easy for the students: (*Multiple selection possible*)
- (a) Branches - simple comparisons
 - (b) Loops
 - (c) Branches - compound comparisons
 - (d) Variables
 - (e) Methods
 - (f) other:
12. Rate this programming crash course. This offer would be good because (*Multiple selection possible*)
- (a) Support for non-specialist teachers
 - (b) Variety in the classroom
 - (c) Insight into new methods of knowledge transfer
 - (d) Insight into tools and environments in use
 - (e) Networking and cooperation with the institution
 - (f) other:

MazeMastery – A Python Framework for Teaching Maze-Traversal in High School

Raphaël Baur², Jens Hartmann¹, and Jacqueline Staub¹

¹ Faculty 4, Computer Science, University of Trier
Behringstraße 21, 54296 Trier, Germany

² Department of Computer Science, ETH Zürich
Universitätstrasse 6, 8092 Zürich, Switzerland

`rabaur@student.ethz.ch`

`{s4jehart, staub}@uni-trier.de`

Abstract Programming is an activity that is strongly based on abstraction as many solutions can be generalized to cover a wide range of applications. For students who are still in the process of developing their abstraction skills, learning to write code for the general case can be a daunting experience. We present the Python framework MazeMastery which offers a didactic tool for teaching graph exploration strategies through maze-based challenges at high school. Students can verify their algorithms against randomized test cases that currently span six levels of complexity as maze structures continuously increase in their structural complexity. The tool offers an adaptable platform for examining students' learning while challenging their conceptual understanding. MazeMastery is an open-source community project for scientists and educators.

1 Graph Theory and the Long Way to High School

School systems around the globe are slowly adapting to the changing demands in the job market by enforcing algorithmic problem-solving competencies in their school syllabi. The UK, a country once said to be lagging behind Germany in terms of CS education [15], has swiftly reacted by introducing computer science as a compulsory subject across all grades of schooling. Meanwhile, most districts of Germany still teach computer science as an optional subject which is offered at the earliest from lower secondary school [14].

Addressing algorithmic concepts late requires that the focus is all the clearer. The central objective of computer science in school is to foster computational thinking skills. This term summarizes a variety of skills that encompass problem decomposition, pattern recognition, generalization, and abstraction, alongside learning to develop algorithms and formalizing them by programming. A wealth of problems can be tackled using these skills, e.g., from modeling problems using the formal notations of mathematics all the way to advanced programming concepts such as recursion.

One domain that has already been shown to be accessible to students of all ages is graph theory [7, 16]. As a fundamental and powerful data structure, graphs

can be used for modeling and analyzing multi-entity relationships, such as social networks or transport systems. The corresponding algorithms are universally applicable to all graphs including grid graphs and mazes. Since the 1980s, graph theory has been suggested as a topic for computer science education in Germany [2] and has not disappeared since. In dealing with graph problems, students need to think systematically, experiment, and test their strategies.

Programmers typically explore a range of possible solutions – rarely is there only one correct solution for a given problem. Depending on their skill level, learners may find solutions between two ends of a spectrum: at one end they come up with conceptually simple solutions that are highly specialized to a specific problem instance, while at the other end, students find generalized and complex solutions for larger problem classes. The journey along this spectrum can be modeled as an adapted semantic wave. Maton’s *semantic wave theory* [18] distinguishes between semantic density/gravity to describe the different levels of human ingenuity. *Semantic density* conceptualizes the degree of condensation of meaning (perceived as complexity). Basic programming commands, for instance, contain less semantic meaning (i.e., they are semantically less “dense”) than the concept of recursion. In contrast, *semantic gravity* conceptualizes the degree of abstraction of meaning (perceived as abstraction). Tailored solutions to a given problem are less abstract (and thus contain less semantic gravity) than fully generalized solutions. Teachers can model a student’s learning path via their teaching materials according to the concept of a semantic wave.

Existing materials are available for generating and visualizing mazes [12], for illustrating search algorithms [3,5,17,1], for interacting with mazes in mixed reality [9], using robots [6] or via classical programming [4,13]. While these existing projects feature various maze-related topics in isolation, we present a tool that combines all parts in a single didactic framework for Python programming classes. Moreover, we incorporate techniques from the semantic wave theory into a single open-source tool. The following section describes how this tool works and how it can be used for modeling mazes as graphs and for subsequently traversing a maze in a visual environment.

2 A Framework for Exploring DFS via Maze Traversal

Graph traversal algorithms such as depth-first-search (DFS) are topics in computer science programming curricula at high school. To implement a DFS graph traversal algorithm from scratch, several preconditions must be met: Students require familiarity with both non-linear data structures like graphs for modeling purposes and with programming constructs such as sequences, loops, variables, branching, lists, matrices, stacks, and recursion.

In the following, we present a didactic tool for teaching *maze traversal* in Python based on the above-mentioned prerequisites. We first discuss how mazes can be modeled as graphs and then dive into the technical details by presenting a tool that generates custom maze instances to challenge students’

conceptual understanding. Finally, we discuss the generation of mazes with dedicated attributes.

2.1 Modeling Mazes as Graphs

Students encounter mazes in their everyday lives, from architecture to entertainment; it is thus an intuitively known object of study that can be modeled as a graph. Specifically, grid mazes can be represented as planar graphs whose nodes correspond to individual maze cells. A cell permits access to its spatially adjacent cells if and only if there is no wall between them. We call such cells *neighbors*. Three features characterize grid mazes:

1. There is a distinctive *start* and *end point*. These nodes can be marked, e.g., using dedicated markers. In our case, the start point corresponds to the initial position of the agent, whereas the endpoint is the location of the **Minotaur**.
2. A grid maze consists of individual cells arranged in a *grid structure*. Each cell has at most four neighbors; one in each compass direction.
3. Whether or not two adjacent grid cells are neighbors (which allow passing through in both directions) is determined by the presence or absence of separating walls.

Consider a rectangular maze of m rows and n columns exhibiting the attributes above. A natural approach to model such a maze as a graph consists in representing each cell as a node, arranging them in a grid of the same dimension as the maze, and connecting two adjacent nodes by an edge if their corresponding maze cells have no walls in between. With this representation, we obtain a grid graph of size $m \times n$. Each node has a unique coordinate (i, j) with $i \in \{0, 1, \dots, m - 1\}$ and $j \in \{0, 1, \dots, n - 1\}$ that can be used as a two-tuple to determine the cell. [Figure 1](#) illustrates such a maze of size 4×5 alongside the corresponding grid graph. In this case, maze traversal starts on the cell with coordinates $(0, 0)$, and the target is on coordinates $(2, 3)$.

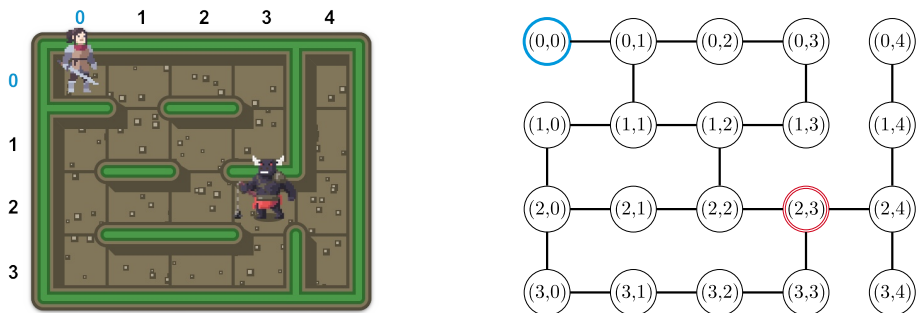


Figure 1: The same 4×5 grid structure once represented as a maze and once as a graph

Note that this naming convention for nodes not only allows for assigning each cell a unique name, but also allows for deriving from a coordinate (i, j) its adjacent cells to the north, south, west, and east ($(i - 1, j)$, $(i + 1, j)$, $(i, j - 1)$ and $(i, j + 1)$ respectively). This implicit information is the first objective that students get to explore in our task series.

Although nodes in grid graphs typically have 4 neighbors (ignoring nodes at the maze border), this is generally not the case in grid graphs representing mazes. In grid graphs for mazes, nodes tend to have fewer neighbors with as few as 0 to 4 edges per node. For example, in [Figure 1](#), cell (1,3) is connected only to (0,3) and (1,2). Although (2,3) and (1,4) are also spatially adjacent, they do not allow direct access due to the prevailing topology with walls. It is thus impossible to directly visit the Minotaur on cell (2,3) from (1,3).

2.2 MazeMastery – A Tool for Teaching Maze Traversal

We developed a Python library named *MazeMastery* to facilitate students' explorative, programmatic, and in-depth exposure to the graph model presented in the previous subsection. We provide MazeMastery as a stand-alone Python package, which allows it to be easily integrated into any development environment supporting a Python interpreter. We provide an open-source implementation at [GitHub](#), and the package can be installed using the `pip` package manager using `pip install mazemastery`.

The library provides a student-friendly user interface created with `tkinter` that offers randomized exercises and tests for a diverse learning experience. A minimal vocabulary of six basic commands, shown in [Figure 2](#), is sufficient in combination with classic Python constructs to use the library in high school programming classes.

Students take control of an agent and are challenged to guide it from the starting point to the end point (where the Minotaur awaits, drawing inspiration from mythological narratives). Commands like `get_pos()` and `set_pos(c)` are provided to determine the agent's position and facilitate its traversal to neighboring cells. Using `put_[blue|red]_gem(c)` and `has_[blue|red]_gem(c)`, markers can be positioned on the current cell for state management, and by utilizing `has_minotaur()` and `get_neighbors()`, students gain the means to examine the local surroundings of the agent, even when dealing with randomized and unknown grids. MazeMastery uses a global Cartesian coordinate system where the agent is, however, only able to move within his local neighborhood.

2.3 Level Description

MazeMastery currently provides six progressive levels, each curated to systematically increase the complexity and abstraction of the algorithmic implementation required. Students have the flexibility to approach the levels with whatever methods seem suitable. Each level comprises mazes with a specific characteristic that poses requirements on the generalizability of the students' solutions.

get_pos()		set_pos(coordinates)	
Description	Return current position	Description	Move to <code>coordinates</code> if neighboring
Parameters	–	Parameters	Integer tuple <code>(i, j)</code>
Returns	Integer tuple <code>(i, j)</code> , corresponding to the agent's current position	Returns	<code>None</code>
put_[blue red]_gem()		has_[blue red]_gem(coordinates)	
Description	Mark current position with a blue (red) gem, overwriting previous gems	Description	Check if the <code>coordinates</code> are marked with a blue (red) gem
Parameters	–	Parameters	Integer tuple <code>(i, j)</code>
Returns	<code>None</code>	Returns	<code>True</code> , if the node is marked with a blue (red) gem, <code>False</code> otherwise
has_minotaur()		get_neighbors()	
Description	Check whether the Minotaur is at the agent's current position	Description	Return list of nodes neighboring to the agent's current position
Parameters	–	Parameters	–
Returns	<code>True</code> , if the Minotaur is at current position, <code>False</code> otherwise	Returns	List of integer tuples corresponding to neighboring nodes

Figure 2: Basic Commands used within MazeMastery

An overview of all levels along with an exemplary and modular codebase showcasing key insights, can be found in [Figure 10](#). These insights serve as guiding principles, regardless of the specific implementation chosen by students. We designed levels such that students can build upon their previous code and add new functionality as the complexity of the mazes increases, thus increasing semantic density and decreasing semantic gravity over time.

Level 1 introduces the agent starting on the left of an extended pathway with the Minotaur at the wall to the right (see [Figure 3](#)). Navigating through this maze requires students to visit several consecutive nodes. Students can achieve this using a single coordinate instance whose column index is continuously updated and used in the `set_pos(c)` command.

Level 2 shifts the positioning of the Minotaur at the beginning of each level (see [Figure 4](#)). Without adapting the solution found for level 1, the agent likely steps past the Minotaur due to this change. Consequently, the search should terminate upon encountering the Minotaur. To this end, our library exposes the `has_minotaur()` function, which checks whether the Minotaur is located at the agent's coordinate.

Level 3 deviates from the linear progression of the previous two levels by presenting an *unicursal* maze structure that does not contain any dead ends or loops, i.e., each node still has only two neighboring nodes but is not arranged in a straight line (see [Figure 5](#)). In previous levels, it was sufficient to only adjust the

column index of the agent’s position, but now this method will prove ineffective if the path changes direction. Keeping track of where the agent is coming from and where it is going to be is the core challenge of this stage. The users will most likely find that they need to mark previously visited nodes, which can be achieved using the `put_blue_gem()` command, which places a blue jewel on the agent’s coordinates, and the `has_blue_gem(c)` command, which enables users to check if node `c` was previously marked.

Level 4 presents a *perfect* maze structure exclusively comprising dead ends, devoid of any cycles (see [Figure 6](#)). It introduces nodes with more than two neighbors, thus confronting students with the challenge to effectively address dead ends and navigate despite them to advance further in the maze. A possible strategy involves the utilization of blue gems to mark the nodes that have been visited once and red gems, which can be placed and checked for with the commands `put_red_gem()` and `has_red_gem(c)` respectively, for nodes that have been visited twice (i.e., on the return path from a dead end). When students find themselves at a dead end, they need to employ a mechanism to exclude nodes already marked from their path options.

Level 5 presents *multiply connected* mazes with both dead ends and cycles (see [Figure 7](#)). To address this challenge, students need to recognize the importance of making previously visited nodes retrievable, a requirement not present in previous levels. In this context, students are encouraged to contemplate adopting either an iterative or recursive approach. In the iterative method, students employ a stack data structure, which allows them to store the visited nodes in a last-in-first-out manner. As the agent naturally progresses through the maze, visited nodes can be pushed onto the stack, facilitating easy retrieval when backtracking is necessary.

Level 6 serves as the most advanced stage in the current structure. After locating the Minotaur, the students’ objective changes to guiding the agent back to the entrance ([Figure 8](#)). This can be achieved via recursion; the recursive descent enables traversal into the maze, while the recursive ascent allows for an effective return path. Level 6 serves as a culmination of the student’s learning journey, consolidating their understanding of recursive algorithms and providing a platform to showcase algorithmic skills in the context of maze traversal.



Figure 3: Sample Maze for Level 1



Figure 4: Sample Maze for Level 2

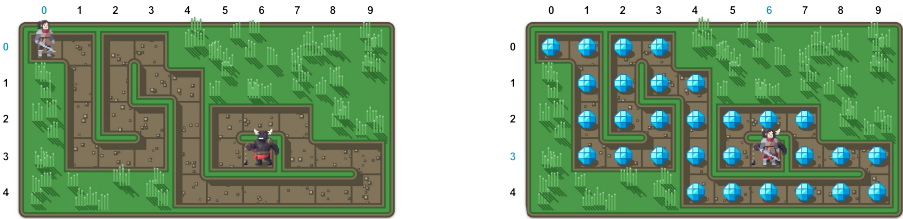


Figure 5: Unicursal Maze with One Single Path From Start to End (Level 3)



Figure 6: Perfect Maze without Loops but with Dead Ends (Level 4)

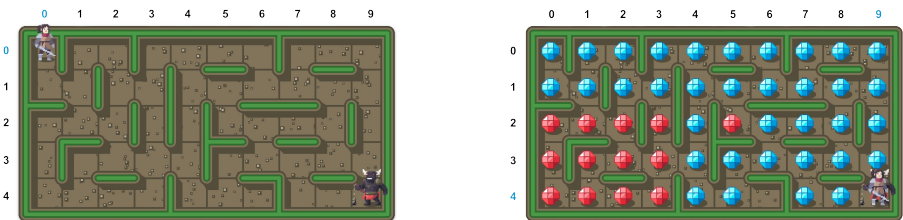


Figure 7: Multiply connected Maze with Loops and Dead Ends (Level 5)

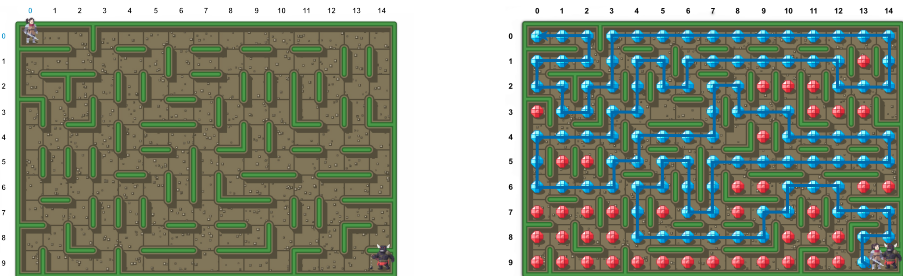


Figure 8: Multiply connected Maze with Backtracking (Level 6)

LEVEL 1	LEVEL 2	LEVEL 3
Straight path, agent starts left, Minotaur at path end	Straight path, agent starts left, Minotaur at arbitrary position	Uncursal maze, Minotaur at arbitrary position
Goal: Reach the Minotaur	Goal: Reach the Minotaur	Goal: Find the Minotaur
Insight 1 (I_1)	Insight 2 (I_2)	Insight 3 (I_3)
Movement through direct manipulation of the coordinates	Once the Minotaur was found, there is no further need to search	Visit only neighbors that have not yet been visited $I_{3,1}$ Mark visited node with blue gem $I_{3,2}$ Check marked status, when considering next node
<pre>def level1(): while True: i, j = get_pos() new_pos = (i, j + 1) set_pos(new_pos)</pre>	<pre>def level2(): while not has_minotaur(): i, j = get_pos() new_pos = (i, j + 1) set_pos(new_pos)</pre>	<pre>def level3(): while not has_minotaur(): put_blue_gem() for neighbor in get_neighbors(): if not has_blue_gem(neighbor): new_pos = neighbor break set_pos(new_pos)</pre>
LEVEL 4	LEVEL 5	LEVEL 6
Perfect maze, Minotaur at arbitrary position	Multiply connected maze, Minotaur at arbitrary position	Multiply connected maze, Minotaur at arbitrary position
Goal: Find the Minotaur	Goal: Find the Minotaur	Goal: Find the Minotaur and backtrack
Insight 4 (I_4)	Insight 5 (I_5)	Insight 6 (I_6)
If all neighbors are marked as visited (i.e., dead end), mark node with red gems and backtrack	Once reaching blue or red gems (i.e., circles) mark node with red gems and backtrack	Encourage short, recursive approach and return to the starting node
<pre>def level4(): while not has_minotaur(): put_blue_gem() found_neighbor = False for neighbor in get_neighbors(): if not has_blue_gem(neighbor): found_neighbor = True new_pos = neighbor break if not found_neighbor: put_red_gem() for neigh in get_neighbors(): if not has_red_gem(neigh): new_pos = neigh break set_pos(new_pos)</pre>	<pre>def level5(): stack = [get_pos()] while not has_minotaur(): put_blue_gem() found_neighbor = False for neighbor in get_neighbors(): if not has_blue_gem(neighbor): found_neighbor = True new_pos = neighbor break if not found_neighbor: put_red_gem() new_pos = stack.pop() else: stack.append(get_pos()) set_pos(new_pos)</pre>	<pre>found_minotaur = False def level6(): global found_minotaur put_blue_gem() for neighbor in get_neighbors(): if not has_blue_gem(neighbor): new_pos = neighbor old_pos = get_pos() if has_minotaur() or \ found_minotaur: found_minotaur = True return set_pos(new_pos) level6() put_red_gem() set_pos(old_pos)</pre>

Figure 9: Level structure with possible implementation

2.4 Our Contribution to Teaching Recursion

The outlined level hierarchy is designed to provide students with a structured and progressive introduction to algorithmic concepts related to maze traversal. The challenges emphasize the importance of iterative and recursive methods, conditional statements, and data structures for graph traversal. While some of these concepts can be challenging to teach, recursion is known as a concept that is especially hard to grasp [10,19] and for which several incorrect mental models have been found [8,11]. We address two such flawed mental models:

1. The *looping model* manifests as an erroneous identification of recursion and loops, where the recursive process is perceived as a unified entity rather than a sequence of successive instantiations. This misinterpretation blurs the distinction between recursion and traditional looping constructs. It is crucial to note that until level 4, the maze challenges can indeed be effectively solved using loops. However, although loops can be replaced with tail recursion, this substitution is not universally applicable to all forms of recursion. This limitation becomes evident in levels 5 and 6, where more complex tasks such as identifying cycles and dead ends demand a more profound understanding of the backtracking process. As students grapple with the intricacies of backtracking in these scenarios, they might recognize the constraints inherent in relying solely on an iterative approach, especially without an explicit stack.
2. The *recursive descent model* manifests as an initial assumption that recursion terminates solely upon reaching the base case. This model is challenged in levels 5 and 6 as students realize the necessity of the recursive ascent to continue the traversal process by backtracking and overcoming impediments. They realize that instructions following the recursive call are required to return to the correct path using different-colored node markings.

Students who develop mental models that do not fully capture the essence of recursion might potentially encounter challenges in their conceptual understanding, which could affect their problem-solving abilities. Addressing these misconceptions is, therefore, a pertinent aspect of teaching programming. Our library addresses this point using tailor-made maze instances for each of the six levels.

2.5 Maze Generation

To accommodate different levels of complexity, MazeMastery generates maze instances with specific topological and geometrical properties. We discuss how mazes for each level are generated:

- Levels 1 and 2 present mazes that represent straight corridors that only differ in the positioning of the Minotaur. The corresponding graphs consist of a single chain of nodes that differ only by their column coordinates.
- Level 3 involves *unicursal* mazes constituted by a single corridor with randomized turns. This type of maze is generated by sampling self-avoiding walks, i.e., from an initial node, neighbors are added by sampling randomly

until the path self-intersects. To prevent early intersections, we use a heuristic by increasing the probability of choosing neighbors in less-traversed directions. For each direction, we count the number of nodes that lie opposite the direction at hand, yielding values z_d where d indicates the direction *up*, *down*, *left*, and *right*. To compute the probability of selecting the neighbor in a specific direction, we use a *tempered softmax* function, defined as $\exp(-\beta z_d) / \sum_{d'} \exp(-\beta z_{d'})$. The tempered softmax serves two purposes. First, it compresses the counts we obtain into a range between 0 and 1. This compression allows us to interpret the results as probabilities. Second, it ensures that the sum of these probabilities across all directions equals 1, creating a valid probability distribution. The parameter β influences the shape of the resulting distribution. When β is small, the distribution approaches uniformity, meaning each direction is chosen with roughly equal probability. Larger values of β emphasize the differences between the counts, increasing the chances of exploring less frequently chosen directions. However, deterministically extending the path towards a less-explored direction will not yield trajectories that cover the whole grid, but will tend to the center of the grid and then end due to self-collisions. Choosing $\beta = 0.01$ seems to strike a good balance between uniform distribution, which is prone to early self-intersections, and a strategy that strives towards unexplored areas too greedily, leading to paths that converge to the middle of the maze too quickly.

- Level 4 involves mazes with junctions and dead-ends, but no cycles, so-called *perfect mazes* that correspond to trees. To generate these mazes, we create a spanning tree using randomized depth-first search. In that process, potential neighbors are chosen uniformly at random. To create the final maze, we start with a fully disconnected grid graph and connect two nodes if and only if they are connected in the previously generated spanning tree. With that, we ensure that each cell is reachable and no loops exist, following the properties of a spanning tree.
- Levels 5 and 6 involve mazes with dead ends and loops. We first generate the maze analogously to level 4. Then, we remove walls with probability p if their removal does not create 2×2 grids within the maze that do not contain a wall, retaining the maze structure while avoiding large areas with no walls.

3 Evaluation

We conducted a qualitative think-aloud study with five male participants aged 19 to 23 years. They had all programming experience of at least 3 years but varied knowledge of Python syntax.

3.1 Study Setup

The study aimed to assess (i) whether the tool allows heterogeneous learning, and (ii) whether there are specific problems all participants encounter. After a brief introduction to the six commands (see [Figure 2](#)), participants started progressing from level 1 to 6 on their own. All actions were recorded for subsequent analysis.

3.2 Findings

A short summary of experiences:

- (i) Personalized learning seems possible. Solutions for level 1 sometimes foreshadowed later concepts. Participants progressed at their individual pace, some reaching level 3 in just five, others taking more than 20 minutes.
- (i) Six levels seem too few. Intermediate levels between 2 and 3, and between 3 and 4 are advisable for a smoother learning experience. Within 60 minutes, no participant reached beyond level 4, one only reached level 3.
- (ii) Confusion arose from associating commands with the agent's actions. Initial parameterized commands like `put_red_gem()` allowing off-agent gem placement caused confusion, leading to the command set presented before.

More detailed information including solutions is provided [here](#).

4 Conclusion

Programming is an activity that requires abstraction as many solutions can be generalized to cover a wide range of applications. For students who are still in the process of developing their abstraction skills, this can be a daunting experience. This work proposes an approach to address this hurdle in the context of graph traversal using two-dimensional mazes. MazeMastery, our Python framework, currently provides six levels of increasing semantic complexity: students first infer global attributes of a given problem class by analyzing concrete instances, they then develop a generalized algorithm and finally verify their algorithm against concrete but unknown test cases. The intended learning path varies in semantic complexity both throughout a specific level but also across all levels.

Ongoing research evaluates the framework's effectiveness in teaching algorithmic problem-solving and analyzing learning paths. The modular and adaptable nature of the framework allows for customization and integration with other teaching resources in the context of programming. Limitations include the frameworks' early development stage which did not yet involve tutorials and a quantitative evaluation. However, we hope that this work inspires the scientific community to become an active partner in researching the topic of graph theory in education and promote future research on or with our open-source Python framework.

Acknowledgment We gratefully acknowledge the generous support of the Carl-Zeiss-Foundation for funding our research. Moreover, we extend our heartfelt appreciation to Dr. Martin Löhnertz for his assistance in proofreading and providing input to this work.

References

1. Alan Blair, David Collien, Dwayne Ripley, and Selena Griffith. *Constructivist Simulations for Path Search Algorithms*, pages 990–998. Australasian Association for Engineering Education, Sydney, 2017.

2. R. Bodendiek, H. Schumacher, K. Wagner, and G. Walther. *Graphen in Forschung und Unterricht: Festschrift K. Wagner*. B. Franzbecker, 1985.
3. Alberte Emilie Christensen, Cecilie Jegind Christensen, Johannes Louis Geishauser Hald, and Nicolai Otto. Generating and solving mazes, 2019.
4. Xiaozhou Deng, Danli Wang, Qiao Jin, and Fang Sun. Arcat: A tangible programming tool for dfs algorithm teaching. In *Proceedings of the 18th ACM International Conference on Interaction Design and Children, IDC '19*, page 533–537, New York, NY, USA, 2019. Association for Computing Machinery.
5. Clemens Bandrock et al. Projekt 4d labyrinth. https://www.mintgruen.tu-berlin.de/mathesisWiki/doku.php?id=ss2021:project6:4d_labyrinth retrieved 23-4-10, 2021.
6. Sergey Filippov, Natalia Ten, Ilya Shirokolobov, and Alexander Fradkov. Teaching robotics in secondary school. *IFAC-PapersOnLine*, 50(1):12155–12160, 2017.
7. J. Paul Gibson. Teaching graph algorithms to children of all ages. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 34–39, 2012.
8. Tina Götschi, Ian Sanders, and Vashti Galpin. Mental models of recursion. *ACM SIGCSE Bulletin*, 35(1):346–350, 2003.
9. Lorenz Klopfenstein, Saverio Delpriori, Brendan Paolini, and Alessandro Bogliolo. Codymaze: The hour of code in a mixed-reality maze. In *INTED2018 proceedings*, pages 4878–4884. IATED, 2018.
10. Dalit Levy and Tami Lapidot. Recursively speaking: analyzing students' discourse of recursive phenomena. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 315–319, Austin Texas USA, 2000. ACM.
11. Claudio Mirolo. Is iteration really easier to learn than recursion for CS1 students? In *Proceedings of the ninth annual international conference on International computing education research*, ICER '12, pages 99–104, New York, 2012. Association for Computing Machinery.
12. Muhammad Ahsan Naeem. pyamaze. <https://github.com/MAN1986/pyamaze> retrieved 2023-4-10, 2021.
13. Richard Rasala, Jeff Raab, and Viera K. Proulx. The sigcse 2001 maze demonstration program. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 287–291, 2002.
14. Richard Schwarz, Lutz Hellmig, and Steffen Friedrich. Informatikunterricht in Deutschland – eine Übersicht. *Informatik Spektrum*, 44:95–103, 2021.
15. Sue Sentance and Neena Thota. A comparison of current trends within computer science teaching in school in germany and the uk. In *Informatics in schools: local proceedings of the 6th International Conference ISSEP 2013; selected papers; Oldenburg, Germany, February 26–March 2, 2013*, volume 6, pages 63–75. Universitätsverlag Potsdam, 2013.
16. Robert R. Snapp. Teaching graph algorithms in a corn maze. *ACM SIGCSE Bulletin*, 38(3):347–347, 2006.
17. MakeSchool Team. Solving the maze: Trees and mazes. <https://makeschool.org/mediabook/oa/tutorials/trees-and-mazes/solving-the-maze/> retrieved 23-4-10.
18. Jane Waite, Karl Maton, Paul Curzon, and Lucinda Tuttiett. Unplugged computing and semantic waves: Analysing crazy characters. In *Proceedings of the 2019 Conference on United Kingdom & Ireland Computing Education Research*, pages 1–7, 2019.
19. Susan Wiedenbeck. Learning recursion as a concept and as a programming technique. *ACM SIGCSE Bulletin*, 20(1):275–278, 1988.

A Appendix

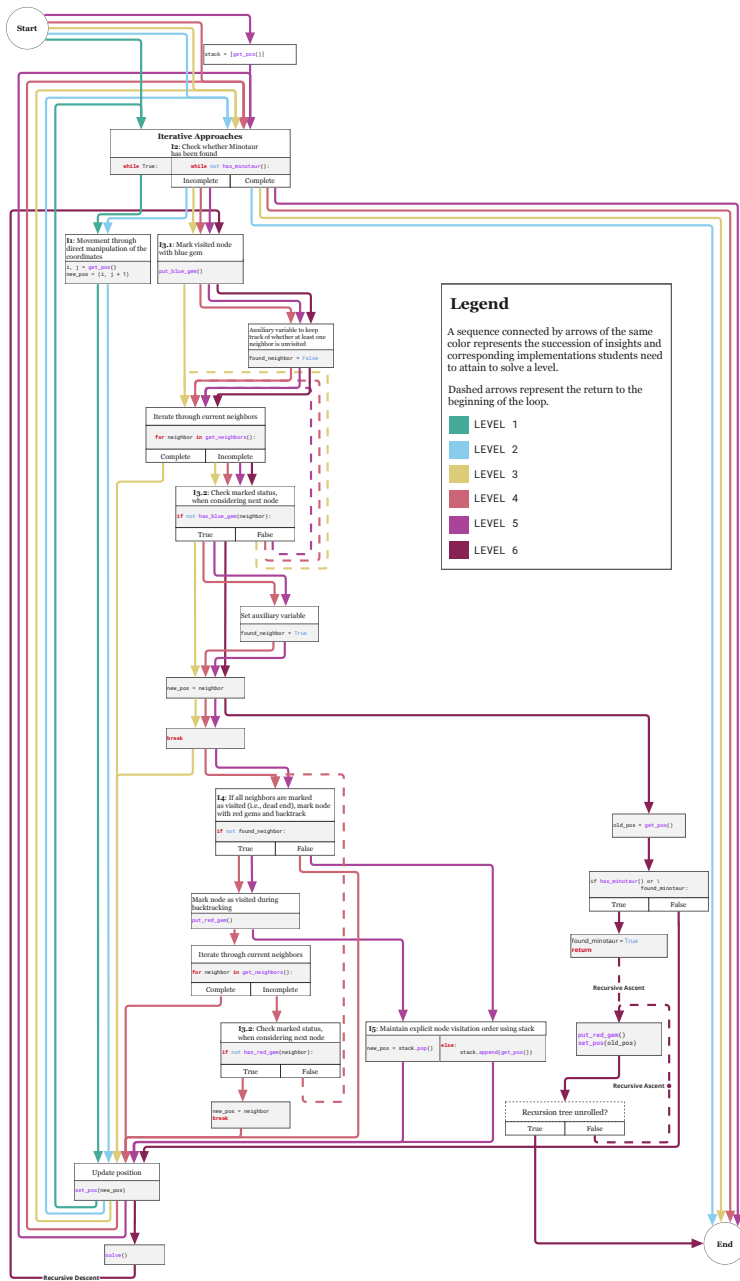


Figure 10: Flowchart illustrating the insights gained in the course of programming

Computer Science Education with a Computer in the Background

Maciej M. Sysło¹[0000-0002-2940-8400]

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

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

Keywords: Unplugged, Computational Thinking, Curriculum

1 Introduction

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

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

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

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

2 CS education in Poland

2.1 Early History of CS education in Poland

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

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

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

2.2 Computational thinking in the CS curriculum of 1997

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

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

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

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

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

2.3 The new CS curriculum

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

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

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

2.4 Computational thinking

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

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

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

3 CS education with a computer in the background

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

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

Looking back at the history of CS education in Poland, briefly described in Sec.2 (Sysło, 2014a), algorithmics plus programming and problem solving using algorithmic thinking were closely related in the 1965 and 1985 curricula for schools as well as in all national core curricula after 1997. In 1980’ when regular CS lessons entered

schools, students had a long way (in distance and time) to a computer, therefore they had to spend a lot of time writing their programs on paper before the programs reached a computer. Also teachers were explaining CS concepts and algorithms using traditional tools. It was time of unplugged introduction to CS and preparation for programming. I remember HS classes coming with their teachers to our Institute for CS lessons (the Institute was quite well equipped in computers) – the students spent first hour in a classroom developing their algorithms and programs and then spent one hour in a computer lab uploading, running, testing, and debugging their programs.

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

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

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

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

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

One may think also about other types of environments which are combination of different tools, mechanical and electronic calculating machines, games, computer games etc. which can be used to introduce students to fundamental concepts of computing. We use such environments at a children's university (Sysło, Kwiatkowska, 2014b). Our approach contributes to constructionist learning, to learning by doing and making

meaningful objects in the real world, computational models of real-world situations. Our learning environments are extensions of classical unplugged ones by encouraging children to purposely and properly use computers for certain activities.

3.1 Classical unplugged

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

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



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

3.2 Bebras tasks

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

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

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

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

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

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

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

3.3 Educational robotics

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

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

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

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

3.4 Algorithmics and programming unplugged

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

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

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

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

Each problem situation in the *Pyramids* can be first discussed, analyzed and solved to some extent far from a computer. Popular examples are: social games, short codes, change making, etc. The book contains also a chapter on the stable marriage problem which has a much longer history in the author' teaching using unplugged approach. In a class on algorithmics in the early 1970', the author has decided to introduce the Gale and Shepley's algorithm for creating stable marriages to a group of students (the same number of boys and girls). First, the students created lists of preferences in the other sex group and then they started to perform the algorithm (which is a kind of greedy method) interchangeably choosing in the other group and revising their choic-

es when refused in the other group. Finally a class concluded writing a computer program which in that time was run in the batch mode. The author was able to see the benefits of the applied approach – unplugged – after 20 years, when he met one of the students and he remembered exactly how the algorithm “run” on the living organism of students – he was able to repeat it. I doubt whether he would be able to reproduce a program written for this algorithm. Today, when computers are at hand and everywhere and I still recommend this algorithm to be performed in a group of students before they start programming it.

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

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

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

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

3.5 Conclusions

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

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

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

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

All these four types of activities have one thing in common – they are addressed to all students, including also those who do not think about connecting their professional future with CS. Therefore, they are to bring them closer to CS using various methods and from different points of view, with or away from the computer, to varying de-

grees of depth. As a result, they are to tear off the secrecy from CS solutions and bring closer the laws and mechanisms of their functioning. Knowledge of these mechanisms can be useful even to a non-specialist to understand their operation, and sometimes even modify them for their own purposes.

References

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

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

Effects of the Use of Robots on Algorithmization, Decentration and Locating in the Plane Skills

Emma Schenkenberg van Mierop^[0009-0001-4502-7704],
Acsa-Loriane Schmidt, and Morgane Chevalier^[0000-0002-9115-1992]

University of Teacher Education, Lausanne, Switzerland
{emma.schenkenberg, acsa-loriane.schmidt, morgane.chevalier}@hepl.ch

Abstract. Now that computer science (CS) has entered the curriculum, two questions regularly emerge: on the one hand, teachers wonder in which time slot to teach CS and, on the other hand, they wonder about the equipment to be used and thus the methods of implementing the activities (plugged or unplugged). Without a time slot in the schedule, CS activities are widely used in mathematical activities, especially for planar orientation. In this context, we question the effect of the use of a robot by 10 to 12-year-old students on their abilities to create an algorithm, decenter, and coordinate in the plane. An experimental method was conducted with thirty-six students divided into two experimental groups (with and without robots). Four critical skills were assessed in a pre- and post-test: algorithmization, decentration, absolute location, and relative location abilities. The results show that the students who programmed the robots made more progress overall than those who performed paper-based tasks only. The algorithmization performance of both groups improved significantly, showing that this skill can be trained in both plugged and unplugged activities. The contribution of this research is twofold as it shows the effect of the use of educational robots on students' ability to decenter and, assures teachers that they can use both plugged and unplugged modalities for computer science activities during planar orientation activities in mathematics.

Keywords: teacher education · computer science education · disciplinary course

1 Introduction & Context

Computer Science (CS) entered the compulsory school curriculum in the French-speaking part of Switzerland in 2021. Since no time slot is dedicated to its teaching in the K-8 timetable, this new discipline has largely found its place in mathematical activities, particularly in those of locating in the plane. Indeed, the design of a route is a recurrent task in geometry and it can be materialized by educational robots (ER) which execute programs to move from point A to point B. As computer science is not a subject that is being thought on its own, it needs to be integrated into other subjects. Therefore we often see the use of robots in mathematics, but it is also used during language courses. The use

of ER then allows "placing the student at the heart of the conscious learning process, which sees him/her involved as an actor of his/her own learning tools" [7] (p. 35). Beyond this pedagogical added value, it remains relevant to study the contributions of ER to the construction of knowledge in both CS and mathematics. In the context of the route task, each of these two disciplines mobilizes the notion of algorithmization as knowledge to be constructed, while relying, as a prerequisite, on the students' ability to locate in the plane. However, the literature indicates that this prerequisite is often underestimated and that the ability to locate in the plane often poses difficulties for students [2] insofar as it implies, upstream, the ability to decenter [13] and thus to locate in space. Faced with such a nesting of abilities, it is clear that the route task is not so easy. However, the literature also shows that the use of ER, such as the Blue-bot robot (TTS Group Ltd, Hucknall, Nottinghamshire, United Kingdom), allows precisely the training of students' ability to decenter [8,3]. Therefore, our research question is the following: **if ER allows students to develop their ability to program and to decenter, does it have also a positive effect on their ability to locate themselves in the plane?** As a result, the objective of the present study is to measure the effect of the use of ER on formulating an algorithm, decentration, and locating in the plane abilities of 10 to 12-year-old students. Our research hypotheses are as follows: 1) Students who performed the route task by programming a robot (test group) will improve their ability to decenter (thus to locate in space) significantly, while this will not be the case for students who only performed the same task on paper (control group). 2) The students who performed the route task by programming a robot (test group) will significantly improve their ability to locate on the plane, as a result of the development of their decentration skills, while the students who only performed the same task on paper (control group) will not.

This paper has the following structure: in Section 2, we mention analyses of similar initiatives or approaches to CS education of future teachers. Subsequently, we talk about the source of the data we collected to answer our research question in Section 3 and analyze it in Section 4. We conclude finally in Section 6.

2 Related Work

2.1 Locating in the Plane

According to Piaget [12], knowledge related to locating in space develops in childhood when the child is in the operative stage (between the ages of 7 and 12). At this stage, children can adopt different points of view from their own, in particular, to distinguish between what is in front of them and what is behind them, but they still have some difficulty distinguishing between left and right. Moreover, knowledge related to planar location is developed around 11-12 years old, when the child moves into the formal operative stage. Euclidean space (coordinate space) then appears, which corresponds to "global knowledge of an environment independent of the individual's point of view" ([10], p. 25). Furthermore, three types of cues can be distinguished according to Charnay and Douaire [4]: 1) The subjective cue that takes into account the observer's point

of view i.e., the cue is placed on the subject, and the directions are determined according to the subject's frame of reference; 2) The objective reference point which is independent of the observer's point of view i.e., the chosen objects are used as temporary reference points and the directions are defined independently of the observer's point of view; 3) The absolute reference point i.e., reference points defined by a reference point (origin) and directions and orientations (graduated axes). Based on this state-of-the-art, in-plane locating tasks often consider, on one hand, the relative cues and, on the other hand, the absolute cues.

Moreover, while manipulating objects, 2 spaces should be considered: the micro-environment considers the space between the student and the plane while the meso-environment considers the space between the student and the robot which moves both in the plane and in the space. The use of robots thus adds complexity to the task of locating objects in the plane.

2.2 Decentration and Robots

Locating in space requires the ability to decenter, i.e. to adopt points of view other than one's own. According to Piaget[12], during the decentration process, the subject moves from the spatio-temporal stage to the logical-mathematical stage. As soon as students start using an object (such as a robot), they need to know how to decenter from it, so as to be able to locate in the plane and space. In this regard, a study [14] tested students' ability to distinguish left and right through a paper-based activity versus an activity with a robot. This study was carried out in Quebec with 22 students aged 6 and 7. The results were similar between the pre-test (60% success rate before the activity with the robot) and the post-test (62% success rate after the activity with the robot), which does not, at this stage, allow us to attribute an effect of the use of a robot on the decentration activity. Nevertheless, according to the researchers, we should continue to explore the effects of paper-based activities aimed at decentration awareness, and the effects of activities with educational robots that mobilize cognitive strategies on decentralized movement planning. Our study thus allows us to continue exploring this avenue, albeit with older children (aged 10 to 12).

2.3 The Route Task and the Skills of Algorithmization and Programming

While the route task is aimed at the ability to locate in the plane, it is also often implemented to develop the ability to anticipate what is needed for problem-solving. It is in this context that the notions of algorithmization and programming take their place in both mathematics and CS. Learning to program, using educational robots is a relatively recent pedagogical approach [11]. The task of moving from point A to point B (referred to here as the route task) offers an affordable programming opportunity, insofar as the problem-solving posed by this task is common. Programming involves writing computer code to create a program to be executed by a machine (in this case, a robot). This program tells the robot what to do and how to do it. As a result, the problem-solver (here, the student)

needs to think before programming in order to be able to solve the problem [6] behind the route task. As these previous authors demonstrate, educational robots are tangible, offering students the chance to put their thoughts into practice by manipulating them. Like puppets, students move them (for example) to embody the solution to the problem posed. It is at this point that students verbalize a solution, i.e. formulate in their own words the behavior of the robot to be programmed. The next step is to transpose this behavior, formulated in the student's language, into a programming language (the robot's). In the context of our study, it is pertinent to delineate between the proficiencies of algorithmization and programming, considering that not all students engage in robot programming. Algorithmization, herein, pertains to the procedure involving the conceptualization, formulation, and construction of algorithms. In our pedagogical approach, students are tasked with the creation of a comprehensive set of sequential directives, transcribed onto paper, aimed at strategizing a navigational course from point A to point B. The act of programming itself involves a whole range of actions and thoughts, and when it comes to programming robots, it also involves the physical dimension and spatial location (not just in the plane). In fact, according to [7], the specific features of robots compared to other digital tools, such as computers, are on the one hand that "the robot is distinguished by its nature as a real and systemic object", which contrasts with the virtual character of computer-based educational software [9], and secondly that the robot can "combine learning from robotics and learning by robotics".

3 Data Collection & Methodology

3.1 Population

In order to address our research question, we conducted a quantitative quasi-experiment involving two classes from primary schools in Switzerland's Canton of Vaud. A total of 36 primary students aged 10 to 12 participated in the study, which lasted three weeks. In order to create two equivalent groups, the overall score on a pre-test, which measured all four skills (decentration, algorithmization, relative location, and absolute location) was assessed (Table. 1). The research was conducted in accordance with the stipulations set forth in the 190 decision of the education department of the canton of Vaud (Switzerland), which required anonymization of data, consent from school principals, and voluntary participation with written consent from legal representatives of the participants. The study was conducted by two pre-service teachers from HEP Vaud with the assistance of two qualified primary teachers. One researcher was present per class to oversee the experimental groups and only intervened if there was an issue with the robots. The other qualified teacher oversaw the control group and intervened only to correct exercises when they were finished. Most students never programmed a robot before and never had a computer science course as it is not implemented in the curricula. However, all students knew how to use a tablet and received a short 30-minute introduction to the Blue-bot robot.

	Experimental group	Control group	Total
Number of girls	11	10	21
Number of boys	7	8	15
Total	18	18	36
Grades $x < 60$	7	8	15
Grades $60 < x < 80$	4	5	9
Grades $x > 80$	7	5	12

Table 1. Distribution of participants in both groups.

3.2 Design of the Study

The experimental group received a 90-minute math course dedicated to the coordinate plane and locating objects in space, during which they programmed a Blue-bot robot using the Blue-bot application on a tablet. Each group was provided with one robot, a gridded floor mat, a tablet, and an exercise worksheet (Appendix. 8). The control group performed a paper-based activity in pairs, completing traditional mathematical exercises (Appendix. 8). Both groups received similar feedback: the control group was informed if an exercise was correct or incorrect, while the experimental group received feedback from the tablet or the robot. All the exercises used in the study were sourced directly from Swiss exercise books or were modified to align with the robot-based activities. The 90-minute session required students to create an algorithm to guide a robot from point A to point B. While students working on paper wrote down the algorithm (which they embodied in a programming language made of arrows), students working with the robots programmed the path on a tablet (Table. 2). As the paper-based activity was routine, students were expected to complete it more quickly. On the other hand, students working with the robot were expected to take more time by programming the robot to accomplish the task.

Pre-test	Introduction to robotics	Robots and Paper activity	Post-test	Exchange groups
All students	All students	Gp 1 with robots Gp 2 on paper	All students	Gp 1 on paper Gp 2 with robots
30 minutes	60 minutes	90 minutes	30 minutes	60 minutes

Table 2. Design of the experiment

It was known beforehand that the precision of the movements of the Blue-bot would pose a challenge for students in the experimental group. Due to the robot's tendency to move 15cm and turn 90°, it was anticipated that the robot may not always end up in the precise location intended. The students were advised of this issue and were instructed to replace the robot when necessary. Furthermore, it was anticipated that programming exercises would be relatively more manageable for students working with the robot, while more conventional coordinate plane exercises would prove more difficult with the robot. Conversely, it was also

anticipated that students working on paper would experience greater difficulty with the algorithmization, but would face less difficulty with the coordinate plane exercises. While creating the exercises, we presumed that students might encounter challenges in transferring their learning from the robotics-based activities to the paper-based post-test.

3.3 Data Collection & Data Analysis

The experiment tested a two-modality experimental condition: with a robot and without a robot (on paper). To measure the learning of decentration and locating in the plane, we created a paper-based pre- and post-test, which measured four skills: decentration, algorithmization, relative location, and absolute location. These skills were broken down into measurable indicators (Appendix. 6). The pre- and post-test were given before and after the paper-based activity and the robot activity. The dataset utilized in this study consisted of points, each representing a continuous variable. The study quantitatively evaluates the dimensions of decentration and algorithmization. For each instance where the student is required to decenter themselves, one point is awarded if the spatial rotation is executed accurately. Similarly, one point is awarded for the correct execution of a series of steps to evaluate the dimension of algorithmization. The dimensions of absolute and relative locating are evaluated qualitatively based on four weighted criteria, each with a weightage of two points: coordinate order, coordinate sign, origin accuracy, and precision. Scores were assigned based on a total of 16 to 27 points and were weighted to reflect the students' competencies as closely as possible. The pre-test in decentering provided a baseline score and ensured that the control and experimental groups were equivalent. The present study employed a statistical analysis using the Wilcoxon signed-rank test. Excel and XLSTAT software were used for data analysis and visualization. Due to a limited sample size of 18 students per group, the Wilcoxon test was utilized as it is well-suited for assessing significant differences in small paired samples.

4 Findings

The mean scores of students' pre- and post-tests were compiled in a table that can be found in Appendix. 7. The mean scores for the experimental group and control group were compared to assess the effectiveness of the robot intervention. A statistically significant improvement ($p < 0.05$) in all mean scores was observed for both experimental and control groups, with the exception of the absolute locating skill within the control group.

4.1 General Achievement in the Tests

The test revealed a significant difference between the pre- and post-test scores for the experimental group, with a Wilcoxon signed-rank test statistic of $T = 2.50$ ($p = 0.000$). The test also revealed a significant difference between the pre- and

post-test scores for the control group, with $T = 31.50$ ($p = 0.009$). This indicates a statistically significant improvement in performance regardless of the group to which they were assigned. The findings of this study indicate that the choice between paper-based and robot-based instruction does not significantly influence the overall grades of the students. However, it is important to conduct a detailed analysis of each specific skill to determine if there are any significant differences between the two instructional methods.

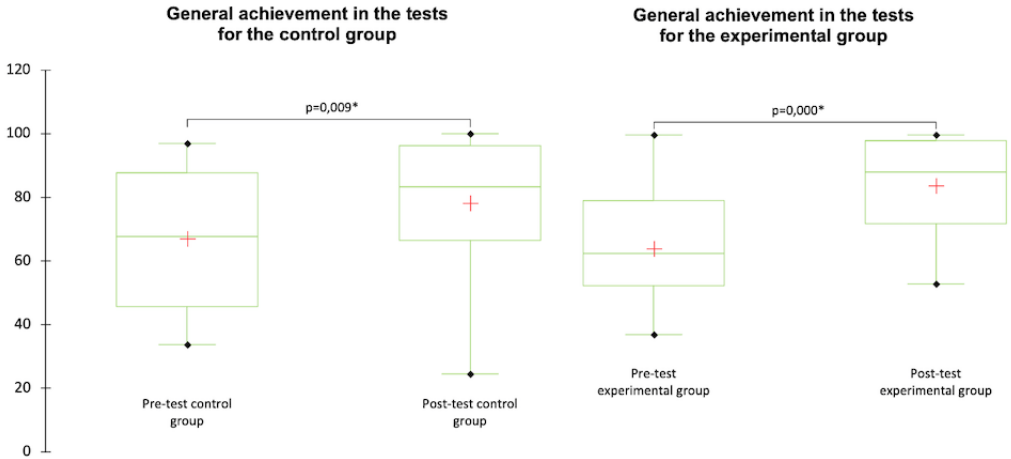


Fig. 1. Graph of general achievement of both groups in the tests.

4.2 Achievement in decentration

The results of the study revealed a significant difference between the pre- and post-test scores for the experimental group, with $T = 25.5$ ($p = 0.014$). Upon conducting a detailed analysis, it was found that the skill of decentring exhibited a significantly greater improvement in the group assigned to manipulate robots compared to the paper group.

4.3 Achievement in Algorithmization

Our study findings indicate that participants' algorithmization skills significantly improved, regardless of whether they completed the exercises on paper or by using a Blue-bot robot. Both the experimental group and control group demonstrated substantial performance gains in their algorithmization abilities. The Wilcoxon signed-rank test statistics were $T = 12$ ($p = 0.003$) for the control group and $T = 2$ ($p = 0.000$) for the experimental group, indicating that the intervention effectively enhanced this skill regardless of the mode of delivery.

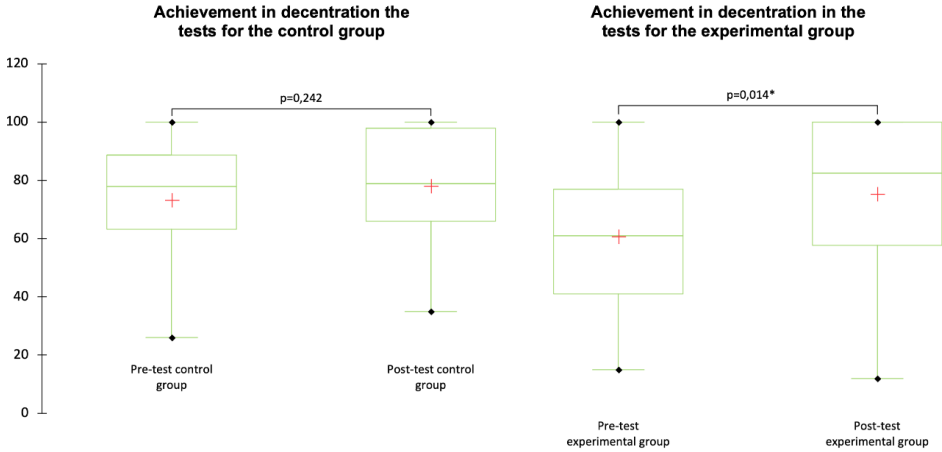


Fig. 2. Graph of achievement in decentration for both groups in the tests.

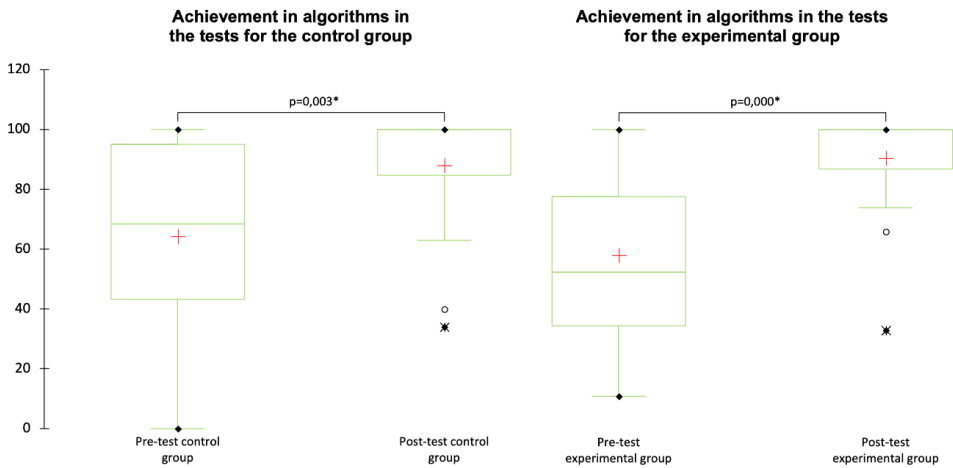


Fig. 3. Graph of achievement in algorithmization for both groups in the tests.

4.4 Achievement in Relative Location

The results of our study also demonstrate that all participants significantly improved their relative location skills, irrespective of their group allocation. The Wilcoxon signed-rank test statistics were $T = 9$ ($p = 0.020$) for the control group and $T = 13$ ($p = 0.026$) for the experimental group, indicating that the intervention effectively enhanced this skill regardless of the mode of delivery.

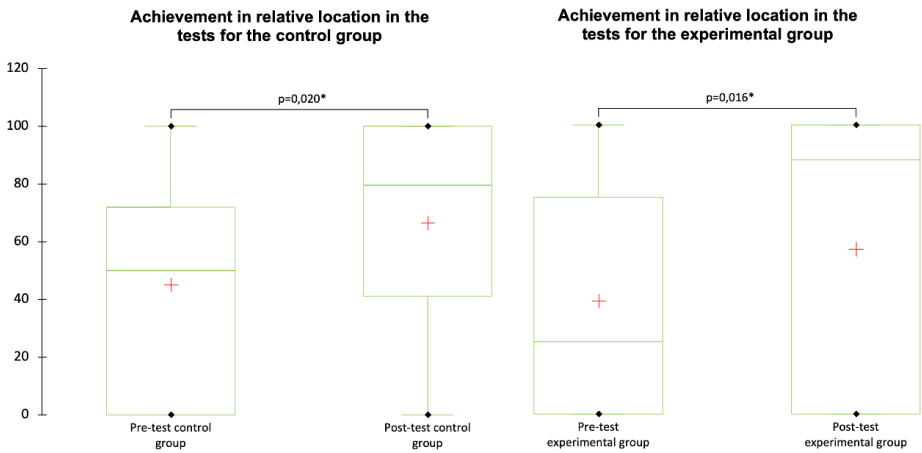


Fig. 4. Graph of achievement in relative location for both groups in the tests.

4.5 Achievement in Absolute Location

The results of the study revealed a significant difference between the pre- and post-test scores for the experimental group, with $T = 10.5$ ($p = 0.040$). The findings suggest that the robot-assisted intervention used in the study was particularly effective in enhancing the absolute location ability of the participants.

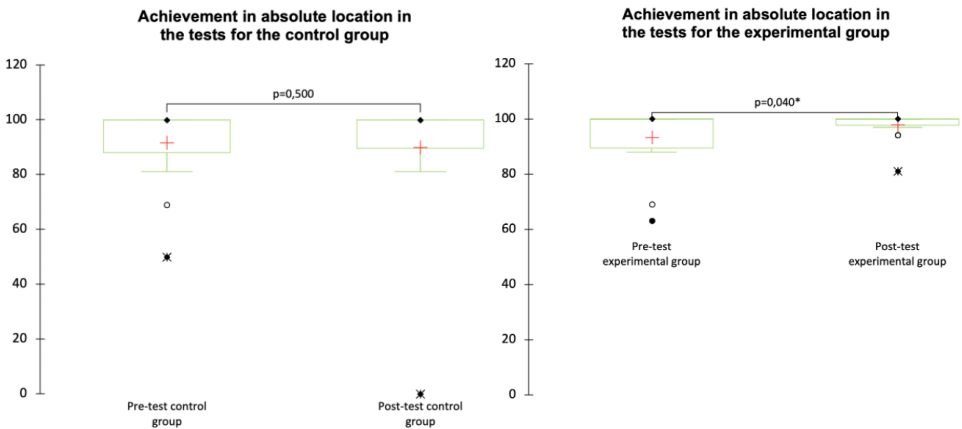


Fig. 5. Graph of achievement in absolute location for both groups in the tests.

5 Discussion

The purpose of this study was to explore the potential benefits of manipulating and programming robots in order to develop decentration and spatial location skills.

Our first hypothesis proposed that students who programmed a robot would significantly improve their ability to decenter. Our findings support this hypothesis. The engagement with robot programming seemed to facilitate a deeper understanding of spatial relationships and improved spatial navigation. Our research findings support existing literature, affirming that programming enhances decentration skills. The Blue-bot application facilitated decentration and anticipation, which are essential components of the learning process [8]. Furthermore, the provision of immediate feedback by the Blue-bot robot reinforced the positive effects on decentration. By promptly recognizing errors and limitations in decentration, students became aware of the challenges involved and demonstrated self-regulatory behaviors. Immediate feedback is particularly beneficial for procedural learning [15], and since decentration is a procedural skill, it requires timely feedback for effective acquisition. In contrast, the control group, which did not receive immediate feedback, experienced greater difficulty in recognizing their own limitations in decentration. Therefore, the disparity in results regarding decentration skills can be attributed to the more direct and tangible approach utilized in the experimental group, involving the robot, as well as the provision of immediate feedback, which facilitates the regulation and learning process associated with decentration.

Our second hypothesis suggested that the experimental group, who engaged in robot programming, would demonstrate a significant enhancement in their spatial skills compared to the control group. Our findings provide further support for this hypothesis, as they reveal that students who engaged in robot programming exhibited notable advancements in their absolute locating abilities. The development of decentration appeared to play a pivotal role in the participants' enhanced capacity to mentally manipulate objects within a plane and comprehend spatial relationships more proficiently. These results have been shared with the teachers of both classes and will be taken into account in further mathematics and computer science courses. Students who solely worked with paper materials remained confined within a limited micro-environment, lacking the stimulus to explore beyond their immediate paper sheet. In contrast, students manipulating robots transcended this micro-environment and transitioned to a broader meso-environment by constantly walking around the Blue-bot grid. We postulate that the size of the resources utilized may have influenced the students' engagement with the environment.

While the absolute locating skill was improved only by the experimental group, the relative locating skill was significantly improved by both groups. Decentration is useful for relative location [4], which explains why the robot group was able to improve this ability. However, a valid question arises as to why the control group also exhibited significant advancements (p -value = 0.003) despite the absence of notable improvements in decentration skills. We posit that this occurrence

may be attributed to the delayed feedback they received, which fostered a more comprehensive understanding of the concepts at hand [5].

Lastly, our findings in the algorithmization skill showed that both groups improved significantly. Research indicates that algorithmization can be effectively taught through both plugged and unplugged approaches [1]. Thus, both groups in our study were provided with opportunities to engage with this concept, either by directly programming robots or by manually constructing algorithms on paper. The cognitive processes involved in the creation of algorithms were highly similar across both groups. Notably, students in the experimental group received immediate feedback, enabling them to self-regulate. In contrast, the control group received delayed feedback, as their exercises were corrected by the teacher once all tasks were completed. Nonetheless, substantial progress in algorithmization skills was observed in both groups. It can be postulated that immediate feedback facilitates task completion, whereas delayed feedback encourages the development of problem-solving strategies. These findings align with previous research, which demonstrates that delayed feedback stimulates the cultivation of anticipation processes and the formulation of more refined behavioral instructions for programming the robot [5].

It is important to acknowledge that further research is needed to validate these findings across larger and more diverse samples, as well as explore the use of robots with more complex tasks and prevent students from getting immediate feedback. Such investigations would contribute to a more comprehensive understanding of the benefits and implications of integrating robot programming into educational practices.

6 Conclusion

This research aimed to investigate the impact of manipulating robots on decentration skills and performance in locating in the plane among 10 to 12-year-old students. The study included two experimental groups, one using robots and one without robots, with a total of 36 participants. Pre- and post-tests assessed four key skills: decentration, algorithmization, absolute locating, and relative locating. The results indicated that students who programmed the robots showed a greater overall improvement, particularly in the skill of decentering, which had a positive effect on location in the plane.

Students who did not manipulate robots demonstrated an understanding of algorithms, as a result of delayed feedback. Both groups made significant progress in algorithmization skills, highlighting the effectiveness of both plugged-in and unplugged approaches.

However, it is important to note that these results are limited by the small sample size of the study. Future research should aim to replicate these findings on a larger scale to further investigate the effects of educational robots on decentration and location skills. Overall, educational robotics has the potential to contribute to both mathematical and computer science education, which proves that ER has its place in primary schools. We strongly believe that computer

science supports mathematics and vice versa. A programmer needs to know basic mathematics to understand algorithmization, but computer science also supports mathematics by enabling students to manipulate abstract concepts.

References

1. Baron, G.L., Drot-Delange, B.: L'informatique comme objet d'enseignement à l'école primaire française ? mise en perspective historique. *Revue française de pédagogie* **195**, 51–62 (2016)
2. Berthelot, R., Salin, M.H.: L'enseignement de l'espace à l'école primaire. *Grand N* **65**, 37–59 (1999)
3. Béziat, J.: Les tic à l'école primaire en france: informatique et programmation. *Revue de* **38** (2012)
4. Charnay, R., Douaire, J.: Apprentissages géométriques et résolution de problèmes au cycle 3. Hatier (2006)
5. Chevalier, M., Giang, C., El-Hamamsy, L., Bonnet, E., Papaspyros, V., Pellet, J.P., Audrin, C., Romero, M., Baumberger, B., Mondada, F.: The role of feedback and guidance as intervention methods to foster computational thinking in educational robotics learning activities for primary school. *Computers & Education* **180**, 104431 (2022)
6. Chevalier, M., Giang, C., Piatti, A., Mondada, F.: Fostering computational thinking through educational robotics: A model for creative computational problem solving. *International Journal of STEM Education* **7**(1), 1–18 (2020)
7. Gaudiello, I., Zibetti, E.: La robotique éducationnelle: état des lieux et perspectives. *Psychologie française* **58**(1), 17–40 (2013)
8. Greff, É.: Le robot blue-bot et le renouveau de la robotique pédagogique. *La nouvelle revue de l'adaptation et de la scolarisation* (3), 319–335 (2016)
9. Hsu, S.H., Chou, C.Y., Chen, F.C., Wang, Y.K., Chan, T.W.: An investigation of the differences between robot and virtual learning companions' influences on students' engagement. In: 2007 First IEEE International Workshop on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL'07). pp. 41–48. IEEE (2007)
10. Nys, M.: Développement des représentations spatiales d'itinéraires virtuels: composantes cognitives et langagières. Ph.D. thesis, Université Paris Descartes Paris Sorbonne (2015)
11. Papert, S.A.: *Mindstorms: Children, computers, and powerful ideas*. Basic books (2020)
12. Piaget, J.: *La naissance de l'intelligence chez l'enfant*. 9e éd. Neuchâtel: Delachaux et Niestlé (1977)
13. Rigal, R.: Right-left orientation, mental rotation, and perspective-taking: When can children imagine what people see from their own viewpoint? *Perceptual and motor skills* **83**(3), 831–842 (1996)
14. Romero, M., Dupont, V., Pazgon, E.: À gauche ou à droite du robot? test de perspective décentrée gauche-droite par le biais d'une activité sur papier et d'une activité de robotique pédagogique. In: *Actes Du Colloque CIRTA*. pp. 52–53 (2016)
15. Shute, V.: Focus on formative feedback. *Review of Educational Research* **78**(1), 153–189 (2008)

Appendix

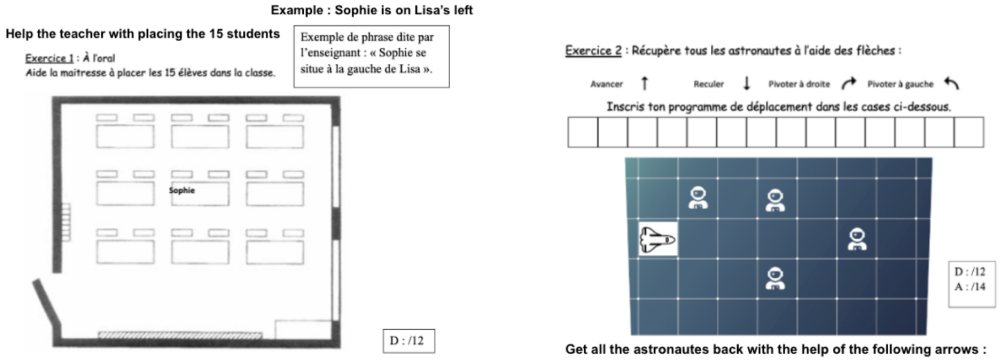


Fig. 6. Example of tasks in Pre- and post-tests. Original instructions in French with English translation.

General achievement	Observations	Minimum	Maximum	Median	Mean	SD
Pre-test control group	18	34	97	68	67	21
Post-test control group	18	25	100	83	78	22
Pre-test experimental group	18	37	100	63	64	18
Post-test experimental group	18	53	100	88	84	15
Decentration achievement	Observations	Minimum	Maximum	Median	Mean	SD
Pre-test control group	18	26	100	78	73	23
Post-test control group	18	35	100	79	78	19
Pre-test experimental group	18	15	100	61	61	25
Post-test experimental group	18	12	100	82	75	27
Algorithm achievement	Observations	Minimum	Maximum	Median	Mean	SD
Pre-test control group	18	0	100	69	64	33
Post-test control group	18	34	100	100	88	21
Pre-test experimental group	18	11	100	53	58	31
Post-test experimental group	18	33	100	100	91	18
Relative location achievement	Observations	Minimum	Maximum	Median	Mean	SD
Pre-test control group	18	0	100	25	39	42
Post-test control group	18	0	100	88	57	47
Pre-test experimental group	18	0	100	50	45	39
Post-test experimental group	18	0	100	80	67	41
Absolute location achievement	Observations	Minimum	Maximum	Median	Mean	SD
Pre-test control group	18	50	100	100	92	14
Post-test control group	18	0	100	100	90	23
Pre-test experimental group	18	63	100	100	93	11
Post-test experimental group	18	81	100	100	98	5

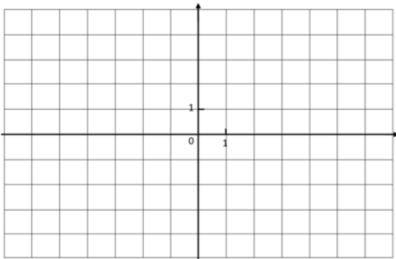
Fig. 7. Table of descriptive statistics

Exercise 1 : create a program so that the Blue-bot arrives at the finish (arrivée) square via the black squares. Use the following commands (forward – backward – turn left – turn right)

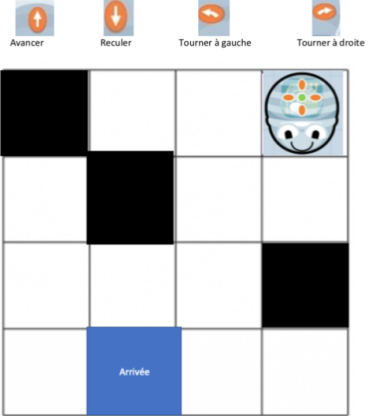


Exercise 2 : read and write the following coordinates and draw the quadrilateral ABCD and EFGH

A (-6 ; -1) B (-4 ; -3) C (0 ; 1) D (-2 ; 3)
 E (2 ; 0) F (4 ; -2) G (6 ; 0) H (4 ; 2)



Exercise 1 : create a program so that the Blue-bot arrives at the finish (arrivée) square via the black squares. Use the following commands (forward – backward – turn left – turn right)



Exercise 2 : read and write the following coordinates and draw the quadrilateral ABCD and EFGH. Create a program so that your robot draws the quadrilateral.

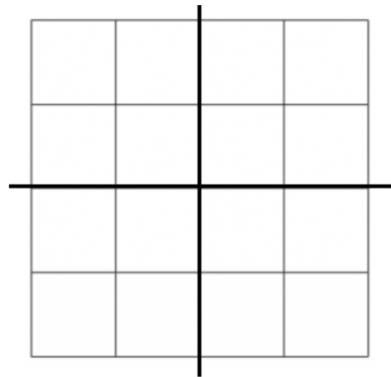


Fig. 8. Left : Paper tasks. Right : Robot tasks. Original instructions in French with English translation.

Teaching Quantum Informatics at School: Computer Science Principles and Standards

Giulia Paparo¹[0000–0002–4782–8337], Regina Finsterhoelzl²[0000–0002–0899–4957],
Bettina Waldvogel¹[0000–0002–0658–1032], and
Mareen Grillenberger^{1,3,4}[0000–0002–8477–1464]

¹ Schwyz University of Teacher Education, Goldau, Switzerland
{giulia.paparo,bettina.waldvogel,mareen.grillenberger}@phsz.ch

² University of Konstanz, Konstanz, Germany
regina.finsterhoelzl@uni-konstanz.de

³ Lucerne University of Teacher Education, Lucerne, Switzerland

⁴ Lucerne University of Applied Sciences and Arts, Rotkreuz, Switzerland

Abstract. The application of knowledge from quantum physics to computer science, which we call “quantum informatics”, is driving the development of new technologies, such as quantum computing and quantum key distribution. Researchers in physics education have recognized the promise and significance of teaching quantum informatics in schools, and various teaching methods are being developed, researched and applied. Although quantum informatics is equally relevant to computer science education, little research has been done on how to teach it with a focus on computer science concepts and knowledge. In this study, we position quantum informatics within Denning’s Great Principles of Computing and propose Quantum Informatics Standards for secondary schools.

Keywords: quantum computing · quantum information science and technology · K-12 education · great principles of computing

1 Introduction

The application of quantum physics to computer science has the potential to open new avenues of thought and endeavors within computer science. From quantum computers that could outperform classical supercomputers at solving computationally hard problems, to quantum key distribution that could make it physically impossible for a third party to eavesdrop unnoticed, quantum informatics has the potential to have a major impact on computer science and society as a whole [29]. Most of these technologies are still in development, and the timeframe and applications for their large-scale implementation are not yet predictable. However, this does not affect the theoretical change in thinking and skills that quantum technologies require. It is important to start thinking about how to introduce students to this new way of thinking at an early stage, for several reasons. First, teaching quantum informatics in schools means laying the foundation for an informed society that can consciously discuss the future

of these technologies and their applications. Second, learning about quantum information can help students develop new perspectives, challenge what they think they already know, and develop a curious approach to the nature of computation. Finally, from an educational perspective, it is a wonderful example to explore how to teach complex and inherently multidisciplinary topics in school.

The aim of this paper is to support the teaching of quantum informatics in lower and upper secondary schools by providing an overview of the subject and defining principles and standards, as an orientation in the development of educational material. To this end, we ask the following questions: *How can quantum informatics be positioned from the perspective of computer science education?* and *What are the most important learning outcomes for secondary school students learning about quantum informatics from a computer science education perspective?* To answer these questions, we first suggest how quantum informatics could be positioned within Peter Denning's Great Principles of Computing [5], and then propose Quantum Informatics Standards for secondary schools.

2 Related Works

In the past years, there has been a rapidly growing interest in the teaching of quantum informatics. This interest sparked especially from physics educators, who saw in quantum technologies the possibility to access quantum mechanics, a notorious hard to teach subject at school, in a more tangible and direct way [7]. Many interesting approaches for teaching quantum informatics at school were developed and evaluated, e. g. [23], [2], [9]. In addition, a variety of serious games [27], visualization tools [21], zines (i. e. self-published DIY magazines) [11], and role-playing [18] approaches were developed that could be used to engage high school students along with the broader public. We will present in more depth some of these approaches in section 7. It is important to note that these were mostly small-scale interventions carried out by a group of experts, and that more extensive research, as well as teaching guidelines, training, or evaluation criteria for teachers are still needed.

An important step towards establishing a common language as well as an orientation for educational programs was the formulation of the European Framework for Quantum Technologies [8]. However, this framework aims to map competencies and skills for quantum technologies, and therefore includes topics of little relevance from a computer science perspective, such as quantum sensing. At the same time, it does not give much space to relevant topics, such as quantum information theory, and focuses mainly on higher education. On the contrary, “Key Concepts for Future Quantum Information Science Learners” [1] and more directly its expansion for computer science [12] target more clearly quantum informatics from a computer science perspective and at high school level. The first [1] offers a good overview and a basis for educators, and it is expanded and evaluated further for Computer Science learning outcomes and activities [12]. The latter, however, is aimed at pre-college and beyond, and the level of knowledge and skills required is not suitable to younger students.

Seegerer et al. [26] discussed quantum computing as a topic in computer science education and made a proposal for its central concepts, ideas, and explanatory approaches. In our previous work, we systematically analyzed the key concepts of quantum informatics further and applied them for the formulation of competencies within guidelines of the German Informatics Society [22]. Although the results are in German, the presented mapping of the key concepts of quantum informatics should also be understandable for English speakers, as it consists mostly of language-independent technical terms.

The above-mentioned works constitute a first good step for supporting the teaching of quantum informatics. Standards for quantum informatics education at the lower and upper secondary level are still needed, and this work aims to fill this gap, building upon some of the previously defined key concepts and competencies ([12], [26], [22], [8]).

3 Terminological Remarks

Generally, it is distinguished between quantum technologies of the first generation and second generation. The first generation is defined by technologies that could be built thanks to our understanding of quantum physics. This definition includes technologies, such as lasers, transistors and global positioning systems (GPS). The second generation of quantum technologies builds on the capability to control and manipulate the properties of quantum systems, for example, to build quantum computers and quantum sensors.

Naturally, as computer science educators, we are looking further than the technology alone. As Michaeli et al. [20] pointed out, the best term to define the field of research describing the knowledge and applications related to these new technologies from the perspective of computer science is in German “Quanteninformatik”. We translated that as “quantum informatics” instead of quantum computer science to indicate a broader view on information processing not limited to computers alone. Other commonly used terms to describe the field are “Quantum Information Science (QIS)”, “Quantum Information Science and Technologies (QIST)”, and “Quantum Computing”. All of these terms intersect in some aspects and at the same time suggest a different focus (i. e. on information theory or computation). The term “quantum informatics”, although less commonly used, describes more directly what we are interested in: We want students to understand, use, and think about the changes in computer science (informatics) brought about by the application of our knowledge of quantum theory. For these reasons, we prefer the term “quantum informatics” in this article and hope it will be broadly used in the future.

4 Methodological approach

In order to position quantum informatics from the perspective of computer science, we used the Great Principles framework of Peter Denning [5] as a way to categorize a new technology within overarching principles that could then

serve as a guide for educators and curriculum developers. As described in more detail in section 5, we first categorized the key concepts of quantum informatics [22] within the Great Principles. The categorization was carried out by the first author, while the second and third authors, with expertise in quantum information theory and computer science education respectively, provided critical review. Re-categorization possibilities were discussed, as well as the addition of further concepts. The categorization was then reviewed by an external computer science expert.

The newly formulated orientation within the Great Principles, as well as relevant aspects of the previously described frameworks ([12], [8], [26]), then served as a basis for the further formulation of the standards. The previously defined learning goals [22] were analyzed within the framework of the CSTA K-12 Computer Science Standards [4]. These describe a general core set of learning objectives for the K-12 computer science curriculum and are widely used for curriculum development and assessment in the USA and around the world, and thus provided an excellent orientation for an initial formulation of international quantum informatics standards. The formulated standards were iterated following the principles of the Standards Developer’s Guide [16]. Details are explained in section 6. Finally, to make the learning outcomes more tangible to the reader, exemplary teaching approaches from the literature were assigned to the standards. The teaching approaches were chosen based on whether they were considered to fit the newly defined Quantum Informatics Standards, specifically whether they were designed for high school students, address one or more of the defined standards, and were suitable from a computer science perspective (see section 7 for more details).

5 Great Principles of Quantum Informatics

Peter Denning defined in a compact and coherent way, the overarching, fundamental principles of computer science, what he called “Great Principles of Computing” [5]. The Great Principles framework aimed to stimulate deep structural thinking about computer science as a discipline and to provide a common language that encourages connections within computer science and across disciplines, providing existing stories while inspiring and structuring new didactic approaches. To do so, he defined seven categories (“windows”) based on the fundamental questions underlying computing technologies. For each window, Denning proposed “principal stories” to depict the richness of each principle, make it more tangible, and to indicate examples of its historical development.

We carried out a categorization of quantum informatics within the Great Principles framework, in order to provide an orientation of quantum informatics within computer science, i. e. to help in structuring and seeing the principles of computer science within quantum informatics. To do so, quantum informatics concepts [22] were first categorized within this framework. The categorization was then critically reviewed, and re-categorization and the addition of new concepts were discussed. There was almost no need for re-categorization. As in Den-

ning’s framework, some concepts could be assigned to more than one window, as these are regarded as overlapping rather than exclusive. However, in order to achieve a comparable balance with the principal stories of classical computing, it was necessary to add several concepts that were not originally part of our selection. This was especially the case for the windows “Design” and “Evaluation”, which were also added later to the Great Principles by Peter Denning [6]. As Denning states, this framework is evolving with time and is not an exhaustive representation of the field, which is even more true for a relatively young discipline such as quantum informatics.

Our goal was to use the framework as a didactic tool to provide orientation, not to revise the Great Principles. While it has been possible to position the main concepts of quantum informatics within this framework, this should not be seen as an exhaustive or conclusive statement, but rather as an exploratory approach to provide guidance to computer science teachers and educators. The results are illustrated in Table 1.

Table 1. Great Principles of Computing and Principal-Stories of Quantum Informatics

Window	Central Concern	Principal Stories (Classical Computing)	Principal Stories (Quantum Informatics)
Computation	What can be computed; limits of computing	Algorithm, control structures, data structures, automata, languages, Turing machines, universal computers, Turing complexity, Chaitin complexity, self-reference, predicate logic, approximations, heuristics, non-computability, translations, physical realizations.	Quantum algorithms (Shor, Deutsch, Grover, aso), quantum simulation, physical and logical control structures, programming languages, universal quantum computing (on different hardware platforms), quantum complexity theory, fault-tolerant quantum computation.
Communication	Sending messages from one point to another	Data transmission, Shannon entropy, encoding to medium, channel capacity, noise suppression, file compression, cryptography, reconfigurable packet networks, end-to-end error checking.	Quantum cryptography, quantum key distribution, quantum networks (quantum channels, trusted nodes), quantum error correction, von Neumann entropy, quantum Shannon theory, multi-party computation with entangled pairs.
Coordination	Multiple entities cooperating toward a single result	Human-to-human (action loops, workflows as supported by communicating computers), human-computer (interface, input, output, response time); computer-computer (synchronizations races, deadlock, serializability, atomic actions).	Classical-to-quantum (hybrid algorithms), human-classical computer (quantum circuit code), human/machines-quantum computer (pulse execution, control of operating conditions).
Automation	Performing cognitive tasks by computer	Simulation of cognitive tasks, philosophical distinctions about automation, expertise and expert systems enhancement of intelligence, Turing tests, machine learning and recognition, bionics.	Quantum machine learning.
Recollection	Storing and retrieving information	Hierarchies of storage, locality of reference, caching, address space and mapping, naming, sharing, thrashing, searching, retrieval by name, retrieval by content.	Quantum measurement, error mitigation techniques, encoding and decoding.
Design	Performance prediction and capacity planning	Hierarchical aggregation, design principles (abstraction, divide-and-conquer, virtual machines, layering).	Fault tolerant design, hybrid quantum-classical architectures, circuit design and transpilation.
Evaluation	Building reliable software systems	Network of servers, Resource sharing, modeling, simulation, experiment, and statistical analysis of data.	Benchmarking, i.e. gate set tomography, quantum state tomography, randomized benchmarking, direct fidelity estimation.

6 Learning Standards for Quantum Informatics

The K-12 Computer Science Framework [4] is divided into practices and concepts that form the basis of our Quantum Informatics Standards. We left the practices unchanged because they are overarching practices that apply to computing in general as well as to its quantum applications. The standards were then formulated following the principles of the Standards Developer’s Guide [16] highlighted in italics. Each standard was formulated by *integrating one or more practices* with a quantum concept and one or more CS concepts. We formulated the standards with a high degree of *rigor*, meaning that the standards should

represent an appropriate cognitive and content demand for students. However, these are only theoretically formulated standards that need to be tested empirically. In light of future empirical evaluation, when unsure, we rather included a topic or chose a higher standard. For reasons of *manageability*, we focused on fundamental concepts like quantum information and quantum algorithms, while left out topics such as quantum simulation and hardware for quantum information processing, as these were either less relevant for computer science standards or too complex for secondary schools. We wrote the standards to be *specific* enough to convey and *measure* the knowledge expected of students, and to be as jargon-free and accessible as possible for a topic as new and complex as quantum informatics (for the sake of clarity and consistency, we could not avoid using some technical terms in the standards definitions, but we have accompanied them with comprehensive explanations). *Equity and diversity* is a particularly important principle for our topic, since what we want to contribute to by teaching quantum informatics early is indeed a more diverse and inclusive science. In formulating the standards, we have been careful to ensure that they can be learned and demonstrated in a variety of ways, using different visualizations and representations, allowing learners to use and further develop individual learning strategies and approaches. However, this is an aspect that needs further and deeper attention in the future as more materials and approaches are developed. Last, the *connection to other disciplines* is intrinsic to physics and mathematics, but has not been made explicit in this first formulation of the standards. Overall, it should be considered that these standards are a first theoretical formulation, and we wish to see them used, tested and critically evaluated in order to formulate more accurate, inclusive and adequate descriptions in the future.

The resulting standards are summarized in Table 2. For each quantum concept, we have indicated the corresponding CSTA concept. We did not establish a one-to-one correspondence with the CSTA concepts, as topics such as quantum error correction and quantum cryptography are very relevant within quantum informatics, while quantum networks and quantum internet are not yet developed enough to be suitable for secondary schools. In the following, each quantum informatics content area is described in more detail, based largely on some of the authors' previous work [22] and on the description of the key concepts in the literature (e. g. [1], [26]).

Quantum information: A qubit is the basic unit of quantum information. Qubits, like bits, always have a well-defined value when measured, but unlike bits, they also exhibit quantum mechanical properties such as superposition and entanglement. Superposition means that a qubit can be in a state that is a combination of 1 and 0, and only when it is measured will it be either 0 or 1 with a probability determined by its previous state of superposition. Qubits can also be entangled, that is, connected in such a way that none of the entangled qubits can be described independently of the others. If one qubit (in a maximally entangled pure two-qubit state) is measured, the state of the other entangled qubit is also instantaneously determined, no matter how far apart the qubits

are. These special properties can provide computational advantages, since it is possible, for example, to influence the state of two or more different, physically distant qubits by a single operation. This understanding of qubits and the laws that they follow is the basis for further understanding of quantum informatics. In the classroom, students should learn what a qubit is and what makes it similar to bits and what makes it different. A deeper understanding of the special properties of qubits would allow students to grasp the possibilities as well as the limitations of quantum information and computation. Students should also be able to read and evaluate the accuracy of a popular article on quantum computing.

Quantum error correction: The current phase of quantum computer development is called the Noisy Intermediate-Scale Quantum (NISQ) era. These processors are very error-prone and are yet too small for implementing large-scale quantum error correction codes on them. Quantum error correction and fault-tolerant quantum computation are currently a very actively researched area and fundamental for the development of large scale quantum computers. Although classical error detection and correction requires students to apply important computer science concepts such as data decomposition and representation, it is not traditionally taught in school. By learning about quantum error correction, students will learn what error correction is, why it is relevant to computer science, and why classical error correction is not applicable to quantum computing.

Quantum computing systems: Different physical methods of building quantum computers are currently being explored, such as superconducting qubits or ion traps. Just as there are different ways of developing quantum computers, there are also different languages for programming them. By using one of these platforms, students can experience the principles of quantum computing and interact with a quantum computer themselves.

Quantum algorithms: Calculations on quantum computers are described by quantum algorithms, and one model to describe them is that of quantum circuits. In a quantum circuit, the steps of the algorithm are described by quantum gates performed on one or more qubits and a measurement operation for read-out. So far, there are only a limited number of useful algorithms that could give quantum computers a significant speed advantage in basic computing problems such as factoring large numbers. However, for many other types of computation, there are no easy ways to implement them on a quantum computer, and there is no advantage to doing so. A current challenge in quantum computing is to develop efficient as well as useful algorithms for quantum computers. In fact, students should not be expected to develop new algorithms themselves. However, they can learn about the effects of gates on qubits and how they can be put together to create an algorithm. Students can replicate some well-known algorithms and reason about their logic and properties.

Quantum cryptography: Shor's algorithm showed that fully functional quantum computers (with a large enough number of qubits and a sufficiently small error rate) would be able to factor large numbers efficiently, threatening today's most widely used encryption methods. However, quantum effects can also make communication more secure based on physical principles.

Table 2. Learning Standards for quantum informatics at secondary school level

CSTA Concept	Quantum Concept	Standards	CSTA Practice
Data & Analysis	Quantum Information	By the end of grade 12 students should be able to: <ul style="list-style-type: none"> - point out the differences and similarities between qubits (quantum bits) and bits as units of information. - explain which new possibilities of information processing arise with qubits (thanks to superposition and entanglement). - explain why quantum error correction is fundamental in the development of quantum computers. - describe the concept of fault tolerance. 	abstraction, computational problems
	Quantum Error Correction	<ul style="list-style-type: none"> - explain why quantum error correction is fundamental in the development of quantum computers. - describe the concept of fault tolerance. 	computational problems
Computing Systems	Quantum Computing Systems	<ul style="list-style-type: none"> - describe the basics of quantum computer design and operation. - use a formal language to interact with a quantum computer. 	creating, testing and defining
Algorithms & Programming	Quantum Algorithms	<ul style="list-style-type: none"> - explain how quantum algorithms can help solve important problems in computer science. - rebuild given quantum algorithms and analyze their advantages over classical algorithms. 	testing and defining, computational problems
Networks & the Internet	Quantum Cryptography	<ul style="list-style-type: none"> - describe why today's most commonly used encryption method is threatened by future quantum computing. - test a quantum key distribution protocol (e.g., BB84) - explain the advantages of quantum encryption and how it differs from other encryption methods. 	collaborating, testing and refining
Impact of Computing	Impact of Quantum Informatics	<ul style="list-style-type: none"> - identify future implications between quantum informatics and its embedding in society. - discuss choices, norms, and behaviors that are possible in response to the opportunities and risks of quantum computing. 	communicating, inclusion

Quantum cryptography takes advantage of the fact that due to the principles of quantum mechanics, it is impossible for a third party to eavesdrop on the system without disrupting it and thus (probably) being detected. Cryptography is fundamental to today's digital society, and many frameworks and curricula have recognized it [17]. Students can learn about quantum cryptography in a course focused solely on cryptography, or while learning about quantum informatics. Students should learn why today's most widely used encryption method, RSA, is threatened by quantum computers. Also, by trying out a quantum key distribution protocol on their own, they can consider how it differs from classical cryptography and where its potential and risks lie.

Impact of quantum informatics: If fully functional quantum computers are realized, they could crack the most commonly used encryption methods. At the same time, they could be used to perform process optimizations that could lead to significant efficiency gains and bring energy savings. In addition, quantum simulation has the potential to contribute to a better understanding of quantum mechanical systems, which could allow us, for example, to develop new drugs and materials. Societal implications lend themselves well to discussion with students, and one could discuss, for example, how access to quantum computing, if limited to individual governments or a few private companies, could alter power relations in society (cf. [29]).

7 Exemplary Teaching approaches

Lastly, we provide concrete examples of how the standards could be implemented in practice. We selected teaching approaches that fit with the newly defined Quantum Informatics Standards (selecting for content, age group and computer science applicability). In order to show how it is possible to develop different approaches to the Quantum Standards, appealing to younger students and different levels of knowledge and abstract thinking, we then classified the selected approaches based on Bruner's modes of representation [3]. These approaches are meant to be examples, and as such we did not undertake a systematic review of all existing approaches, as it was not the focus of this work.

Action based approaches: The role-playing approach to introduce to qubits and their properties developed by López-Incera and Dür [18] is a good example for an action based introduction to *quantum information*. The authors developed a role-playing game where some students are the qubits and others are the scientists. The qubits have to follow rules about how to position arms and legs and what to do when the scientists throw a ball at them (measure them), while the scientist have to figure out these rules. This way it is possible to explain superposition and entanglement, as well as the complexity of scientific hypotheses, in a tangible and playful way. The authors developed a similar role playing approach also for *quantum cryptography* [19] and there are several other unplugged examples to teach to quantum cryptography. Perry et al. [23] developed a pen&paper way to experience the BB84 protocol, one of the main quantum key distribution protocols, which relies on qubit properties instead of

mathematical complexity. Another promising enactive way to learn about quantum key distribution is to build and use the Qeygen machine [28]. With this analog machine, students can exchange a key using the BB84 protocol, simulate an eavesdropper, and experience the possibilities and limitations of quantum key distribution.

Image based approaches: The states of a qubit are generally represented as vectors in a 2-dimensional complex space (Hilbert space), and most textbooks use the Bloch sphere (or a simplified unit circle) as a geometrical representation of qubits ([2], [23]). However, since most high school students lack knowledge of complex numbers and linear algebra, this might not be the most accessible visualization. Often metaphors are used, such as flipping coins, balls of different colors, or a couple in love ordering wine in a restaurant. Although the metaphors have the advantage of making a connection to the everyday life of the students, as also pointed out by Seegerer et al. [26], they run the risk of being taken too literally on the one hand, and of not making the peculiarity of quantum principles sufficiently obvious on the other. Therefore, also other approaches that are closer to quantum mechanical formalism have been developed, such as the QI4Q formalism [25]. Here black and white marbles are used to represent qubits and the boxes the marbles pass through represent *quantum gates*. This approach has the advantage of explaining all necessary quantum gates correctly and in an accessible, tangible way, without using any mathematics, but also the disadvantage of having to learn somewhat arbitrary rules and of being suitable only for relatively simple algorithms (cf. [9]). Economou et al. [9] showed how to use the QI4Q formalism to model the Deutsch algorithm, disguised as a game. This allows students not only to build a *simple quantum algorithm* using the properties of quantum gates, but also to observe the advantages of quantum over classical information processing. As with classical algorithms, it is valuable for students to discuss them with pseudocode or different approaches before being confronted with a formal language.

Language based approaches: Many platforms offer free access (after registration) to their quantum computing power via the cloud, such as Qutech's Quantum Inspire [24] or IBM Quantum [14]. This last one is widely used for educational purposes as it provides an attractive graphical interface where you can drag and drop to simulate (and execute) the effect of different gates on one or more qubits, as well as qiskit, a Python-based software development kit. Students can build their qubit circuits and run them on a real quantum computer, *experiencing quantum computing* and its limitations, i. e. they can compare the simulation with the real computer and understand the need for error mitigation as well as *error correcting techniques*. The IBM website offers a comprehensive textbook with integrated exercises [15], but it is aimed at university students with a high motivation and interest in mathematics. This could be reduced and simplified according to the students' knowledge, needs and the time available (as in [10]). On a symbolic, language-based level, students can also discuss *risks and potentials* in the development of quantum informatics. Although most learning resources mention this aspect, it is never the main focus of the teaching material

on quantum informatics. Interesting approaches to thematize this can however be borrowed by other disciplines such as future studies [13] and integrated in a quantum informatics curriculum.

The presented approaches show how an action based access to quantum information and quantum cryptography is possible for younger students (e.g. lower secondary school), while more complex and abstract topics such as quantum computer systems, quantum algorithms, and the implications of quantum informatics might be more suitable for higher secondary school. In general, we have provided one or more teaching examples for the defined quantum concepts, although not all defined learning outcomes, such as the ability to describe the concept of fault tolerance or the discussion of responses to the opportunities and risks of quantum computing, are directly addressed by existing approaches. We are convinced that it is possible to teach all the defined standards in a way that is appropriate for school, and intend to show this in the future.

8 Conclusion

With this paper, we offer a contribution to the introduction of quantum informatics in schools: Its technologies and practices have been mapped and viewed from the perspective of the Great Principles framework, and a first proposal for learning standards at secondary school level, have been presented. It is important to note that this is only preliminary theoretical work and that the standards need to be tested and evaluated empirically. We are in the process of developing teaching materials and approaches based on the proposed standards and the analysis of existing teaching approaches, and hope that others will do the same. It is our goal that the standards and analysis proposed here will be useful to computer science teachers and educators who are designing new materials, planning lessons, or seeking a first orientation for teaching quantum informatics.

References

1. Alpert, C.L., Edwards, E., Franklin, D., Freericks, J.: Key Concepts for Future QIS Learners, <https://qis-learners.research.illinois.edu/>, (accessed: 08.2023)
2. Billig, Y.: Quantum Computing for High School Students. Yuly Billig (Aug 2018)
3. Bruner, J.S.: *Toward a Theory of Instruction*. Belkapp Press, Cambridge (1966)
4. Computer Science Teacher Association(CSTA): K-12 Computer Science Standards (Revised 2017), <https://csteachers.org/page/standards>
5. Denning, P.J.: Great principles of computing. *Commun. ACM* **46**(11), 15–20 (2003)
6. Denning, P.J.: Computing is a natural science. *Communications of the ACM* **50**(7), 13–18 (Jul 2007)
7. Deutsch, D.: Physics, philosophy, and quantum technology. In: *Proceedings of Sixth International Conference on Quantum Communication, Measurement and Computing*. Princeton, NJ (2003)
8. Directorate-General for Communications Networks, Content and Technology (European Commission), Müller, R., Greinert, F.: *Competence framework for quantum technologies: methodology and version history*. Tech. rep., Publications Office of the European Union (2021)

9. Economou, S.E., Rudolph, T., Barnes, E.: Teaching quantum information science to high-school and early undergraduate students (Aug 2020)
10. EPIQC Education team: Introduction to quantum computing and qiskit (2023), <https://www.epiqc.cs.uchicago.edu/hs-qiskit>, (accessed: 08.2023)
11. Franklin, D., Palmer, J., Jang, W., Lehman, E.M., Marckwordt, J., Landsberg, R.H., Muller, A., Harlow, D.: Exploring Quantum Reversibility with Young Learners. In: Proceedings of the 2020 ACM Conference on International Computing Education Research. pp. 147–157. ACM, New York, NY, USA (Aug 2020)
12. Franklin, D., Rogers, M., Rozansk, D., Tabor, C., Wong, T.G., Yen, B.: QIS Key Concepts for Early Learners: K-12 Framework High School Computer Science (2021), <https://q12education.org/wp-content/uploads/2022/09/QISE-K-12-framework-HS-Computer-Science-1.pdf>
13. Horst, R., Gladwin, D.: Multiple futures literacies: An interdisciplinary review. *Journal of Curriculum and Pedagogy* **0**(0), 1–23 (2022)
14. IBM: Ibm quantum, <https://quantum-computing.ibm.com/>, (accessed: 08.2023)
15. IBM Quantum: Qiskit textbook, <https://qiskit.org/learn/>, (accessed: 08.2023)
16. K12 Computer Science Framework: Guidance for Standards Developers, <https://k12cs.org/guidance-for-standards-developers/>, (accessed: 08.2023)
17. Lodi, M., Sbaraglia, M., Martini, S.: Cryptography in Grade 10: Core Ideas with Snap! and Unplugged. In: Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1. ACM (Jul 2022)
18. López-Incera, A., Dür, W.: Entangle me! A game to demonstrate the principles of Quantum Mechanics. *American Journal of Physics* **87**(2), 95–101 (Feb 2019)
19. López-Incera, A., Hartmann, A., Dür, W.: Encrypt me! A game-based approach to Bell inequalities and quantum cryptography. *European Journal of Physics* **41**(6) (Nov 2020), <http://arxiv.org/abs/1910.07845>
20. Michaeli, T., Seegerer, S., Romeike, R.: Quanteninformatik als Thema und Aufgabengebiet informatischer Bildung. In: INFOS 2021. pp. 1–10 (2021)
21. Migdal, P., Jankiewicz, K., Grabarz, P., Decaroli, C., Cochin, P.: Visualizing quantum mechanics in an interactive simulation – Virtual Lab by Quantum Flytrap. *Optical Engineering* **61**(08) (Jun 2022)
22. Paparo, G., Waldvogel, B., Grillenberger, M.: Quanteninformatik: Schlüsselkonzepte und -kompetenzen. In: INFOS 2023. pp. 1–10 (in press)
23. Perry, A., Sun, R., Hughes, C., Isaacson, J., Turner, J.: Quantum Computing as a High School Module (Apr 2020), <http://arxiv.org/abs/1905.00282>
24. QuTech: Quantum inspire, <https://www.quantum-inspire.com>, (accessed: 08.2023)
25. Rudolph, T.: Q is for quantum, <https://www.qisforquantum.org>, (accessed: 08.2023)
26. Seegerer, S., Michaeli, T., Romeike, R.: Quantum Computing As a Topic in Computer Science Education. In: The 16th Workshop in Primary and Secondary Computing Education. pp. 1–6. ACM, Virtual Event Germany (Oct 2021)
27. Seskir, Z.C., Migdal, P., Weidner, C., Anupam, A., Case, N., Decaroli, C., Ercan, I., Foti, C., Gora, P., Parvin, N., Scafrimuto, F., Sherson, J.F., Surer, E., Wootton, J., Zabello, O., Chiofalo, M.: Quantum Games and Interactive Tools for Quantum Technologies Outreach and Education: A Review and Experiences from the Field. *Optical Engineering*, 61(8) (2022)
28. TueftelAkademie: Qey-gen, the quantum key generator, https://tueftelakademie.de/quantum1x1/quantum_cryptography/gey-gen, (accessed: 08.2023)
29. de Wolf, R.: The Potential Impact of Quantum Computers on Society (2017), <http://arxiv.org/abs/1712.05380>

Measuring Didactical Competencies for Informatics Education among Prospective Primary School Teachers

Christin Nenner¹[0000-0002-5230-4343] and Nadine Bergner²[0000-0003-3527-3204]

¹ TUD Dresden University of Technology, Germany

² RWTH Aachen University, Germany

Abstract. Even primary school children live in a world that is permeated by informatics. Therefore, they need informatics competencies in order to be able to understand and help shape this world. To enable the acquisition of informatics competencies in primary school, informatics-competent primary school teachers are essential. In addition to informatics competencies, they also need the didactical competencies to teach informatics in a way that is appropriate for the subject, the child, and the purpose. This article presents a way to measure didactical informatics competencies. Self-assessment items are used to measure participants' self-perceived didactical informatics competencies pre and post seminar participation. Reflection sheets are used to support reflection on the developed lesson plans, including assessment of target group orientation, informatics relevance, and professional correctness.

Keywords: Informatics education · Prospective primary school teachers · Didactical competencies

1 Motivation and Introduction

Both the Committee on European Computing Education [1] and the European Commission within the Digital Education Action Plan 2021-2027 [4] advocate continuous informatics education for all students, starting in primary school. For this to be implemented, informatics education needs to be integrated into primary school teacher training [13, 5]. Therefore, prospective primary school teachers not only need informatics competencies by themselves but also competencies in teaching them. According to [20], didactical competencies refer to the question of how learning processes can be promoted and supported. In particular, it is about good quality tasks, explanations, and representations. The model of didactic reconstruction for informatics described in [2] includes, among other components, the capture of teachers' perspectives.

In order to enable prospective primary school teachers to develop didactical informatics competencies, courses are designed and offered explicitly for this purpose at various university locations. In a seminar at the PH Schwyz for prospective primary school teachers focusing on data structures, algorithms

(incl. block-based programming) and informatics systems [3], the development of didactical informatics competencies is supported through the use of teaching examples. In a seminar on informatics for primary school at the University of Wuppertal offered for prospective primary school teachers [7], not only existing teaching materials are tested but the participants also develop their own informatics-specific lesson plans (LP). The participants test these with each other in the seminar and reflect on them together. In seminars of a collaborative project in North Rhine-Westphalia for prospective primary school teachers for science [11], the developed LP are implemented by the participants with primary school children. The experiences of the participants are reflected upon cooperatively. At TUD Dresden University of Technology the seminar “Informatics education in primary schools” [19] was designed that combines good approaches of the described courses. For this seminar, previously existing and in the seminar acquired didactical informatics competencies should be measured³. Until now, there is no informatics-specific measuring instrument for didactical competencies.

This article presents a possible instrument for measuring didactical informatics competencies. Chapter 2 addresses the research question and design before exemplary results are presented and discussed in chapter 3. In both cases, the self-assessment questionnaire and the reflection sheet on the developed LP are considered. Chapter 4 on limitations and outlook closes the article.

2 Research Question and Design

In order to address the research question “*How can didactical informatics competencies of prospective primary school teachers be assessed?*”, an instrument consisting of a self-assessment questionnaire and a reflection sheet for the developed LP was designed. It was used for the first time in a seminar on informatics education in primary schools, in which prospective primary school teachers acquire both professional and didactical informatics competencies (see Fig. 1).

2.1 Self-Assessment Questionnaire

The developed self-assessment questionnaire consists of eleven items (see Tab. 1 in the appendix), which are rated on a 5-point Likert scale (1: strongly disagree to 5: strongly agree) before the start of the seminar and after the discussion of the LP (see Fig. 1). Eight self-assessment items (SA1–SA8) were formulated based on competency formulations in the standards for teacher education of the Standing Conference of the Ministers of Education and Cultural Affairs [12, p. 9]. These were supplemented by three items on confidence in SA9: elaborating informatics content together with primary school students, SA10: the excitement of primary school students for informatics content, and SA11: the answering of individual questions about informatics content, taken from [9] with slight adaptations. In

³ In order to also measure subject-specific informatics competencies, another measurement instrument was developed, which is explained in more detail in [18].

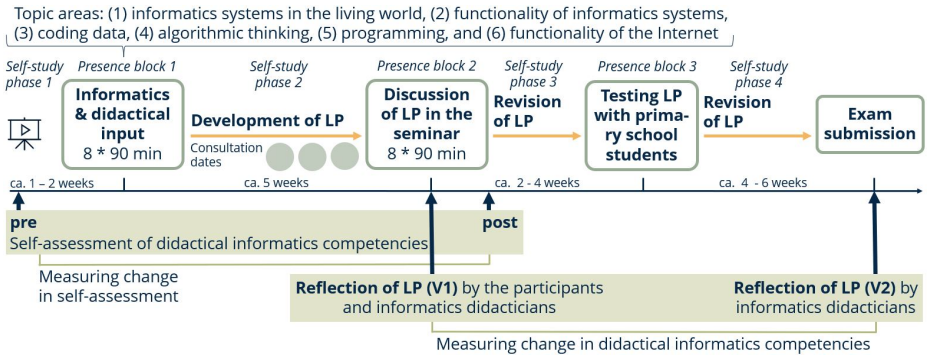


Fig. 1. Structure of the seminar on informatics education in primary school

order to establish a reference to concrete informatics contents, the self-assessment items SA9, SA10 and SA11 are queried for the six topic areas based on the recommendations of the German Informatics Society [6]: (1) informatics systems in the living world, (2) functionality of informatics systems, (3) coding data, (4) algorithmic thinking, (5) programming, and (6) functionality of the Internet. Here, the participants can only indicate whether they are confident in the self-assessment item related to the respective topic area.

2.2 Reflection Sheet for Developed LP

To improve one’s own teaching competencies, lesson observations with subsequent reflections are an important foundation [10]. Based on [8] and [17], a reflection sheet with eleven items was created to survey the didactical informatics competencies (see Tab. 2 in the appendix). In order to use the reflection sheet, the prospective primary school teachers have to create informatics-specific LP, which they present to one another, then test parts and reflect on them cooperatively in the seminar. To develop these eleven items, existing statements in literature were modified or supplemented with regard to the specifics of informatics. The reflection sheet is used by the prospective primary school teachers and the informatics didacticicians to rate the LP on a five-point Likert scale (1: strongly disagree to 5: strongly agree). The participants use the reflection sheet after the presentation and the testing of a part of the LP with the other participants for a peer feedback (example item: The teacher focused on informatics-specific content in the lesson.). The informatics didacticicians use the reflection sheet for the evaluation of the LP (tabular plan and first materials) without further explanation (example item: The LP focuses on informatics-specific content.). In the first step, each informatics didacticician comments on all eleven items for each LP, and rates them on the scale. After the didacticicians have done this independently, they discuss their assessments with each other and agree on a value on the scale. This applies to version 1 (V1) and the revised version 2 (V2). The assessment of the revised versions will take place approximately three months after rating V1.

There is no direct comparison with V1. The assessments of V1 and V2 are compared in order to detect possible changes in the students' individual didactical informatics competencies. Fig. 1 shows the measurement times.

3 Exemplary Results and Discussion

The instrument for measuring didactical informatics competencies has been used in the elective seminar since the winter semester 2021/22. A total of $N = 19$ data sets were collected. Prospective primary school teachers from the 3rd semester or higher participated. Four of these indicated that they had already participated in informatics education courses. Two further participants stated to have educated themselves individually through their own research. In the following, first, the results of the self-assessment questionnaire and then those of the reflection sheet for the developed LP are presented and discussed.

3.1 Exemplary Results on the Self-Assessment Questionnaire

In the pre-post comparison of the participants' self-assessment of didactical informatics competencies (see Fig. 2), significant changes are shown for all eleven items (exact significance (2-sided) ≤ 0.05). The largest changes are shown in SA5: knowledge of material ($\Delta A = 2.68$) and SA7: development of concepts to enable informatics education ($\Delta A = 2.53$). This can be explained by the fact that many already existing materials are presented and tested within the input phase of the seminar. Developing concepts for informatics education is also a focus of the seminar. With average values of 3.68, the participants already estimate before the seminar SA2: importance of informatics education as well as SA10: their confidence in relation to the excitement of students for informatics contents to be high. Thus, the changes are small with 0.53 and 0.84. The high pre value of the importance of informatics education in primary school can be explained by the fact that it is an elective seminar and the prospective primary school teachers freely chose from various elective options. Between participants without and with informatics education, the greatest differences are evident in the items SA3: connection to current curricula ($\Delta A = 1.17$), and SA5: knowledge of material ($\Delta A = 1.06$) prior to participation in the seminar, with the average values of those with prior knowledge being higher in each case. Participants' self-assessments after the seminar differed little between these two groups.

When participants assessed whether they were confident in SA9: elaborating the informatics content, SA10: exciting students, and SA11: answering individual questions (see Fig. 3), significant changes were found in the pre-post design for the topic areas of (2) functionality of informatics systems, (3) coding data, and (4) algorithmic thinking. For the topic area (1) informatics systems in the living world, significant changes were found for SA11: answering questions, and for the topic area (5) programming, significant changes were found for SA9: elaborating the informatics content and SA10: exciting primary school students. For the topic area (6) functionality of the Internet, participants' confidence of

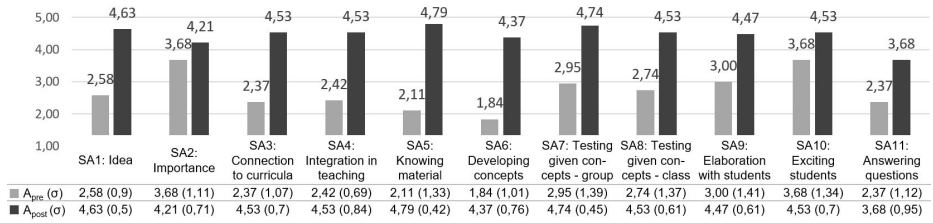


Fig. 2. Pre-post comparison of participants’ self-assessment of didactical informatics competencies. $N = 19$

all three items (SA9, SA10, SA11) actually decreased. This could indicate that, prior to participating in the seminar, the participants associated media literacy competencies such as using the Internet with this topic area, indicating a misconception. In this case, the acquired informatics competency could have led to the participants being less confident in this topic area after participating in the seminar.

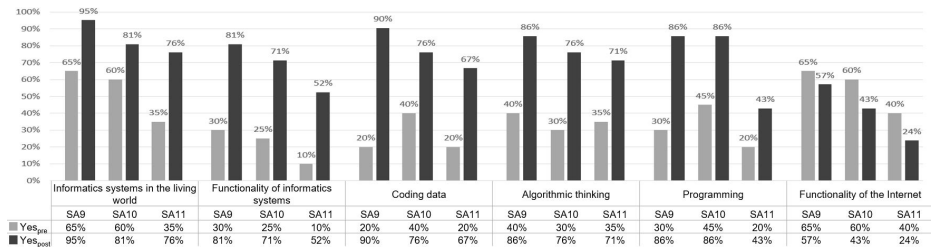


Fig. 3. Pre-post comparison of participants’ self-assessment of didactical informatics competencies SA9, SA10 and SA11 for six informatics topic areas. $N = 19$

Furthermore, it is remarkable that the participants already assess themselves with rather good values for SA9 ($A_{pre} = 3.00$) and SA10 ($A_{pre} = 3.68$) for informatics contents in general prior to the seminar. This could be related to the fact that participants are more likely to associate these items with talking about informatics and may also have misconceptions about informatics (e.g., equating it with media literacy). The participants are much less confident in answering individual questions about informatics content (SA11; $A_{pre} = 2.37$). This could indicate that the participants perceive more informatics competencies as necessary here than in SA9 and SA10. Looking at the confidence for the six topic areas prior to the seminar, less than 50% of the participants are confident in SA9, SA10 and SA11 for (2) functionality of informatics systems, (3) coding data, (4) algorithmic thinking, (5) programming. This could indicate, when asked about their confidence in more concrete informatics topic areas, doubts seem to arise.

3.2 Exemplary Results on the Reflection Sheet for Developed LP

The assessment of V1 of the LP by the informatics didacticians shows deficits especially for the items concerning RS2: prior knowledge of the primary school children, RS4: comprehensible work assignments, RS9: informatics correct explanation, and RS10: target group oriented explanation. Fig. 4 shows an example of the differences in the assessments of the informatics didacticians of the two versions (V2 – V1) of the LP by three participants. The difference of the evaluation on a 5-point Likert scale (1 - strongly disagree, 5 - strongly agree) is shown. A positive difference is to be interpreted as an increase in competency since the score of V1 is subtracted from that of V2.

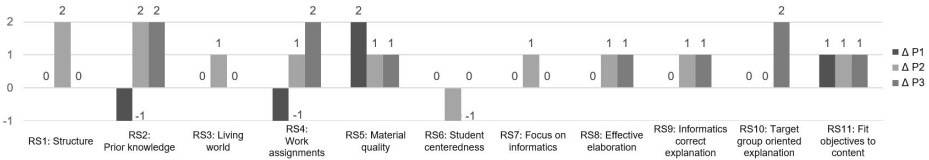


Fig. 4. Differences of the evaluation of the first and revised LP (points V2 - points V1) of three participants (P)

The assessment of the prospective primary school teachers of the LP using the reflection sheet reveals that they generally give rather positive ratings. The deficits of the LP in RS9: informatics correct and RS10: target group oriented explanation, which were pointed out by the informatics didacticians, are not visible in their assessments.

4 Limitations and Outlook

The self-assessment questionnaire of didactical informatics competencies was applied successfully. However, this could only be tested with prospective primary school teachers who voluntarily chose to attend the seminar. Therefore, it is not a representative sample of all prospective primary school teachers. Furthermore, misconceptions, as they have been shown in the experiences of the seminars at the PH Schwyz [3] and the University of Wuppertal [7] (see chapter 1), about the term informatics and the terms in the naming of the topic areas influence the self-assessment of the participants.

The presented reflection sheet can only partially be used to measure didactical informatics competencies. The peer feedback collected with the reflection sheet has no added value for the survey of the didactical informatics competencies because the participants hardly recognized the weaknesses in the LP of the others and generally evaluated them very positively. It cannot be used as a research tool, but it is didactically useful for application in teaching. In addition,

the peer feedback by the participants and the feedback by the informatics didacticians are not comparable, since the formats of the survey are very different from each other. The informatics didacticians provide their feedback only on the LP that are available in tabular form. The participants of the seminar also look into the tabular documents, can ask questions to the presenting participants and often have more or revised materials available. By asking questions or the expression of comments by individual participants in the presence of the whole group, all participants may be influenced in their evaluation of the LP. In the use of the reflection sheet by the informatics didacticians in the assessment of the V1, limitations arose from the fact that the participants did not always follow the instructions for the preparation of the LP. If, e. g., the work assignments and informatics explanations were not provided or specified, the LP could only be assessed inadequately or not at all with regard to the items on these topics. The same applies to the learning materials.

The items of the reflection sheet are partly difficult to evaluate on the five-point Likert scale. Here, item-specific formulations describing the respective levels could be helpful. The use of the reflection sheet enables the consideration of the same criteria for the assessment of the developed LP from the perspective of the informatics didacticians and the participants.

For testing the LP revised by the prospective primary school teachers with primary school students as a real teaching-learning scenario like in the seminars of a collaborative project in North Rhine-Westphalia [11] (see chapter 1), a feedback sheet has already been developed on which the primary school students can evaluate the lesson. For this, statements from [15] were selected, which are supplemented by one or two informatics learning objectives of the respective LP. The seminar leader participates in the tests and uses the reflection sheet to assess the observed lesson. Through the extension, it can be checked whether the informatics competencies can be elaborated by the participants together with the primary school students.

Acknowledgement "The projects underlying this article are part of the "Qualitätsoffensive Lehrerbildung", a joint initiative of the Federal Government and the Länder which aims to improve the quality of teacher training. The programme is funded by the Federal Ministry of Education and Research. The authors are responsible for the content of this publication."

References

1. Committee on European Computing Education: Informatics Education in Europe: Are We All In The Same Boat? Tech. rep., ACM (2017)
2. Diethelm, I., Hubwieser, P., Klaus, R.: Students, Teachers and Phenomena: Educational Reconstruction for Computer Science Education. In: Proceedings of the 12th Koli Calling International Conference on Computing Education Research. pp. 164–173. ACM, New York, NY, USA (2012)

3. Döbeli Honegger, B., Hielscher, M.: Vom Lehrplan zur LehrerInnenbildung - Erste Erfahrungen mit obligatorischer Informatikdidaktik für angehende Schweizer PrimarlehrerInnen. In: *Informatische Bildung zum Verstehen und Gestalten der digitalen Welt*. pp. 97–107. Gesellschaft für Informatik, Bonn (2017)
4. European Commission: *Digital Education Action Plan 2021-2027 – Resetting education and training for the digital age* (2020)
5. German Rector’s Conference: *Teacher education in a digital world. Resolution of the Senate of the HRK on 22 March 2022* (2022)
6. Gesellschaft für Informatik e. V. (GI): *Kompetenzen für informatische Bildung im Primarbereich. Empfehlungen der Gesellschaft für Informatik e. V. erarbeitet vom Arbeitskreis »Bildungsstandards Primarbereich« – Beschluss des GI-Präsidiums vom 31. Januar 2019 – veröffentlicht als Beilage zu LOG IN Heft 191/192.* (2019)
7. Haselmeier, K.: *Informatik in der Grundschule – Stellschraube Lehrerbildung*. In: Pasternak, A. (ed.) *Informatik für alle*. pp. 89–98. Gesellschaft für Informatik, Bonn (2019)
8. Helmke, A.: *Unterrichtsqualität und Lehrerprofessionalisierung. Diagnostik von Lehr-Lern-Prozessen und evidenzbasierte Unterrichtsentwicklung*. Klett-Kallmeyer, Hannover, 1. edn. (2022), Anhang: Einblicknahme in die Lehr- und Lernsituation, ABS, V 7.1
9. Hildebrandt, C.: *Skalenhandbuch Selbstwirksamkeitserwartung von Informatiklehrkräften* (2019)
10. Höppner, C., Dotzler, C., Körndle, H., Narciss, S.: *Training mit Microteaching zur Entwicklung und zum Einsatz formativer Feedbackstrategien in Lehr-Lernsituationen*. In: Uhde, G., Thies, B. (eds.) *Kompetenzentwicklung im Lehramtsstudium durch professionelles Training*. Universitätsbibliothek Braunschweig (2019)
11. Kuckuck, M., Best, A., Gryl, I., Grey, J., Brinda, T., Windt, A., Schreiber, N., Batur, F., Schmitz, D.: *Informatische Bildung in Praxisphasen des Sachunterrichts in NRW*. In: Humbert, L. (ed.) *INFOS 2021 – 19. GI-Fachtagung Informatik und Schule*. pp. 241–250. Gesellschaft für Informatik, Bonn (2021)
12. Kultusministerkonferenz: *Standards für die Lehrerbildung: Bildungswissenschaften* (2019), vom 16.12.2004 i. d. F. vom 16.05.2019
13. Kultusministerkonferenz: *Lehren und Lernen in der digitalen Welt. Die ergänzende Empfehlung zur Strategie »Bildung in der digitalen Welt«* (2021)
14. Lehner, M.: *Didaktische Reduktion*. Haupt, Bern, 2. edn. (2020)
15. Lenske, G.: *Schülerfeedback in der Grundschule: Untersuchungen zur Validität*. No. Bd. 92 in *Pädagogische Psychologie und Entwicklungspsychologie*, Waxmann, Münster New York (2016)
16. Maras, R., Ametsbichler, J., Ostermann, A.: *Unterrichtsgestaltung in der Grundschule - ein Handbuch*. Auer, Augsburg, 6. edn. (2019)
17. Meyer, H.: *Leitfaden Unterrichtsvorbereitung*. Cornelsen, Berlin, 1. edn. (2007)
18. Nenner, C., Bergner, N.: *Informatische Kompetenzen (angehender) Grundschullehrkräfte sichtbar machen: Ein Messinstrument mit selbsteinschätzungs- und aufgabenbasierter Komponente*. In: Hellmig, L., Hennecke, M. (eds.) *INFOS 2023 20. GI-Fachtagung Informatik und Schule*. Gesellschaft für Informatik, Bonn (2023), (in press)
19. Nenner, C., Bergner, N.: *Seminar zur informatischen Bildung in der Grundschule*. In: Desel, J., Opel, S. (eds.) *10. Fachtagung Hochschuldidaktik Informatik (HDI) 2023*. FernUniversität Hagen, Hagen (2023), (in press)
20. Schubert, S., Schwill, A.: *Didaktik der Informatik*. Spektrum, Akademischer Verlag, Heidelberg, 2. edn. (2011)

A Appendix

Table 1. Items for the self-assessment questionnaire of didactical informatics competencies

Self-assessment items SA1 – 8 (based on competency formulations in the standards for teacher education of the Standing Conference of the Ministers of Education and Cultural Affairs [12, p. 9])

SA1: I have an idea of what content and concepts informatics education encompasses.

SA2: The teaching of informatics competencies is of great importance to me.

SA3: I know points of contact for informatics content and competencies in current primary school curricula.

SA4: I have an idea of how to integrate informatics education into my teaching.

SA5: I can draw from a selection of existing materials for the implementation of informatics education in primary school.

SA6: I can develop teaching-learning concepts that enable informatics education for primary school students.

SA7: I have the confidence to test given teaching-learning concepts with a small group (max. ten participants) that enable informatics education for primary school students.

SA8: I have the confidence to test given teaching-learning concepts with a class that enable informatics education for primary school students.

Self-assessment items SA9 – 11 (slightly adapted taken from [9])

SA9: I am confident in my ability to work with primary school students to develop informatics content and skills.

SA10: I am confident in my ability to get primary school students excited about informatics content.

SA11: I have the confidence to answer primary school students' individual questions about informatics content.

Table 2. Items of the reflection sheet (RS) for developed lesson plans (LP) including the source (S) it is inspired by

Nr.	Item - informatics didacticians	Item - participants	S
RS1	The LP is clearly structured.	The teacher structured the lesson clearly.	[8]
RS2	The LP actively links to the students' previous experiences and knowledge.	–	[17]
RS3	The LP contains examples from the everyday life of the primary school students and/or ties in with the interests of the primary school students.	The teacher integrated examples from the everyday life of the primary school students and/or ties in with the interests of the primary school students.	[8]
RS4	In the LP, the work assignments are formulated in a way that is understandable for primary school students.	The teacher formulated the work assignments in a way that is understandable for primary school students.	[17]
RS5	The materials used in the LP are designed in a way that is appealing to and understandable for primary school students.	The teacher designed the materials in a way that is appealing to and understandable for primary school students.	[17]
RS6	The LP is student-centered and activating.	The teacher designed the lesson in a student-centered and activating way.	[17]
RS7	The LP focuses on informatics-specific content	The teacher focused on informatics-specific content in the lesson.	[16]
RS8	In the LP, the informatics-specific subject matter is effectively developed with the primary school students.	The teacher developed the informatics specific subject matter with the primary school students in an effective manner.	[14]
RS9	In the LP, the informatics-specific subject matter is explained correctly.	The teacher explained the subject matter correctly.	[14, 16]
RS10	The LP explains the informatics-specific subject matter in a way that is appropriate for primary school students.	The teacher explained the informatics-specific subject matter in a way that is appropriate for primary school students.	[14]
RS11	In the LP, the informatics-specific content is prepared according to the formulated learning objectives.	–	[14]

Computational Thinking from Preschool to University: The Versatility of UML Modeling in Education

Nina Lobnig^[0000-0002-7097-8317] and Corinna Mößlacher^[0009-0001-6223-406X]

Department of Informatics Didactics, University of Klagenfurt
Universitätsstraße 65-67, 9020 Klagenfurt
Nina.Lobnig@aau.at; Corinna.Moessler@aau.at

Abstract. Teaching computer science is seen as an important part of today’s society and many educational institutions, not only as a separate subject but also as part of cross-curricular and interdisciplinary teaching. Modeling with UML diagrams is a computer science topic that is excellent for this purpose since it offers wide applicability in computer science classes as well as other subjects. In computer science education, modeling is an important part of software development and the design of databases. It also has potential in other disciplines, including language classes, natural sciences, economics, and more, to structure and visualize interrelations and processes in different contexts.

This paper highlights the benefits of (UML) modeling in education and provides teaching examples and ideas. Specifically, this involves the use of activity diagrams, object diagrams, and class diagrams. Apart from computer science classes, one of those teaching examples is planning (stop-motion) videos as an interdisciplinary approach. Another example is using UML modeling in language classes for (better) visualization and understanding of the characters and their relations from a book or play. We use this approach in our workshops and can provide experience from teachers using our teaching material. Overall, this paper offers insight into the successful use of UML modeling in different educational settings and provides practical recommendations and further ideas for teachers as well as instructors seeking to incorporate UML modeling into curricula.

Keywords: computer science education · UML diagrams · interdisciplinary education · cross-curricular education

1 Introduction

In today’s educational landscape, there is increasing recognition of the need to think and act beyond traditional disciplinary boundaries. Efforts are being made in many schools and curricula to promote interdisciplinary and cross-curricular teaching, as evidenced by the multiple references to “interdisciplinary” in nearly all present curricula (e. g. in [2] or even in primary school with explicit interdisciplinary topics [1]). Beyond the well-known goals of deepening and connecting

knowledge, integrating tools and methods from other disciplines can bring several valuable advantages. An example approach is the adoption of modeling with UML (Unified Modeling Language) in educational contexts. Our objective is to highlight how this can enrich non-computer-science subjects and as well be used in interdisciplinary teaching.

UML facilitates the establishment of connections between various topics and concepts, enabling to grasp interrelationships better and enhance their ability to connect and synthesize knowledge and communicate about it. By introducing practical examples straight out of the classroom and workshops, we offer valuable insights and recommendations to educators on how to integrate modeling into their subjects. By doing so, teachers can expand their repertoire beyond traditional and often unstructured mind maps and instead use established and standardized forms of representations already prevalent in computer science.

Visualizing thoughts and ideas can help in understanding and solving problems - an important part of Computational Thinking. When learning this concept, it can be helpful to have methods that help to formulate (unplugged) algorithms and structured thoughts. Writing ideas and concepts in a more formal language, e.g. UML, can also be seen as part of Computational Thinking. Furthermore, one can perform pattern recognition on these structures and abstract thoughts into higher-level concepts. These are all operations that belong to Computational Thinking [9] and can be visualized with UML (object diagrams for structuring, class diagrams for abstraction, activity diagrams for algorithms).

Throughout this paper, we will showcase a range of diverse approaches to support teachers in integrating a structured form of modeling into their lessons. Additionally, we will address the challenges that may arise and provide best practices to overcome these hurdles. We firmly believe in the potential of modeling in the educational context for students of different ages and in different topics and subjects.

2 Modeling with UML diagrams: Why and what for?

2.1 UML as a Standardized Notation for Modeling in Computer Science

UML is a universal modeling language that provides various diagrams for visualizing the structure and behavior of systems. Although its original purpose is to visualize software in a standardized way, the features can be used for various other purposes. UML provides diagrams for various purposes, defined in such a way that they can be applied to different topics. Basically, you just choose the right diagram type for the given information (e.g. entities with properties and their relationships) and visualize the information. The standardized rules ensure that the information can be understood without much additional context by any person who knows UML. The process is no different than visualization in mathematics: you choose the right statistical diagram to represent the results, apply the rules, and any person with basic mathematical knowledge can read the representation without much effort.

This standardization is one of the main features that shows the strength of this method in education: it can be learned as a new method (like statistical diagrams in mathematics) that can be applied to various topics in other subjects. The rules remain the same once learned, only the context changes. In addition, standardization facilitates the design and exchange of teaching materials.

2.2 Beyond Mindmaps: About the Power of Formal Diagrams

A typical approach when trying to visualize information is to create a mind or concept map. This method is widely used - also in a school context - and has many advantages. The rules are easy to explain. Unnecessary filler words are omitted, the most important terms are arranged in a structured way, and so on.

Nevertheless, it is sometimes difficult to create correct mind maps. The line between the words represents a relationship (supercategory - subcategory starting from the middle of the diagram). Often, however, the connecting lines simply represent "somehow belongs to". In the worst case, this leads to circles in the diagram. The relationships can then no longer be interpreted and information is lost. If the diagram is drawn strictly according to the relationship of the items, it is structured in categories (e.g. as a preliminary stage of a table of contents).

The diagram is not suitable for more complex relationships or processes. Actually, there is only one rule that must be followed for creating a mind map: the relationship is represented by a line. Thus, from a computer science point of view, it can be said: mind-maps are always possible if the information can be represented as a tree (as there is a root, edges, and nodes). A mixture of different meanings for the edges is not possible, as the edges are not labeled and must therefore always represent the same relationship. Concept maps address this issue by incorporating labeled edges and enabling cross-links between nodes. Those are used instead of mind maps as an improved and established concept, which can also be used in interdisciplinary contexts (see [6]).

However, they also follow a hierarchical structure like mind maps, and in addition, the nodes are units with a label but without properties. The use cases are thus limited. To be able to represent these contents, more complex diagrams with further rules must be used. In an object diagram, for example, the relationships between the individual objects (nodes) can be defined with identifiers and each object has different properties with different characteristics.

2.3 Choosing the Right UML Diagram: Matching Diagram Types to Modeling Needs

UML includes many diagram types. For use in teaching, a selection must be made that on the one hand covers all important use cases (i.e. different types of information), but on the other hand, gets by with as few different diagrams as possible. A selection could be (according to [8]):

- Use Case Diagram
- Class Diagram (and Object Diagram)

- State Machine Diagram
- Sequence Diagram
- Activity Diagram

The Class and Object Diagram can be used to visualize entities and relationships. The Activity Diagram can be used to visualize every kind of algorithm/process. With these two diagrams, we can cover a wide range of contexts. Of course, there are also other variants to represent relationships and processes. However, UML offers the advantage of standardized representation and is widely used (outside the classroom). The Use Case Diagram, State Machine Diagram, and Sequence Diagram have a rather limited application scope (in our topics) and will not be discussed here.

We have also looked at other variants of representation with diagrams, but currently use mainly UML diagrams. Earlier variants of the material (details on our workshops and material are presented in the next section) also used the entity-relationship diagram. These represent the same content as the class diagram. The original idea was that this diagram would be more design friendly for students. But with this diagram, only classes can be represented - an abstract concept for students. This led to many incorrect diagrams, for example, the students were looking for a place to write the property values and added them wrong. The change to an object diagram (introduced as a fact sheet) for a specific object was easier to understand and use.

When further analyzing the object diagrams, students can more easily recognize the abstract structures and form a class diagram on the object diagram (e.g. class Person from several individuals).

2.4 UML Modeling in Computer Science Education

The use of graphical notations for the structured representation of facts, mostly in the form of UML diagrams, is particularly common in computer science education (not surprisingly, as it is a computer science technique). However, the extent to which it is used in the classroom varies greatly between countries and types of schools. In general, however, it is very similar to its use in computer science. Modeling is most commonly used in software development to describe, analyze and present algorithms and programs. UML diagrams are also used in database design.

For example, in the Austrian curriculum, at the end of primary school, the defined competencies for interdisciplinary computer science education encompass comprehension, execution, and formulation of instructions [1]. This could be done through modeling, although hardly is. In the basic curriculum for digital education [3] the term modeling is mentioned directly, but only in the general section. In the concrete objectives, only “(graphical) notations” within the topic of algorithms are mentioned. Experience (also) shows that modeling is hardly dealt with in lower secondary education. However, in secondary school curricula (like the AHS curriculum [2]), modeling can be found in the area of software development (and algorithms) and especially in the design of databases.

3 Methodology

In our lab (“Informatik-Werkstatt”) we develop teaching material for different (computer science) topics. We try to offer material about important concepts for different age groups (kindergarten to university level). To test and develop our material, we organize workshops. The development of the material, presented later on, follows a multi-step process established at our computer science lab within our university and includes not only the initial development phase but also quality assurance in the form of internal reviews and evaluations and ongoing improvements.

In the initial phase, we break down the important concepts of a computer science topic according to the circumstances (regarding the age group, available time, and others) and create a first draft of the materials. This is refined and integrated into playful, open learning using the “COOL Informatics” concept (Cooperative Open Learning, see [7]). It is then handed over to colleges or field experts to provide constructive feedback and suggestions for improvement. After rework, the materials undergo practical testing in small-group workshops in different age groups, ranging from kindergarten to primary and secondary schools. These workshops are controlled environments for gathering empirical data, insights, and experiences from facilitators and participants. Additionally, collaborative partnerships with schools enable the materials to be sometimes piloted in specific classroom settings, further enriching the empirical basis for refinement and serving as pre-studies.

Upon another revision process, the materials are published under a Creative Commons Attribution (CC-BY) license on a widely accessible material exchange platform¹. This licensing approach enables educators to freely access, utilize, and adapt the materials to suit their instructional contexts and preferences. To support ongoing professional development and dissemination, the workshop team continues to offer sessions centered around the materials. These sessions can be booked by whole classes, where teachers can gain practical insights into their effective implementation within diverse classroom settings. Additionally, the materials are also presented and discussed in teacher professional development programs. By engaging with the materials in this context, educators have the opportunity to share their practical experiences, exchange pedagogical strategies, and collectively contribute to the ongoing refinement and improvement of the materials as we get valuable insights and feedback from practicing teachers. Our approach is related to design-based research, which is described in Aguayo et al. [4] or Barab et al. [5] among others.

4 Interdisciplinary Teaching Examples for Describing and Working with Processes

In this section, we will explore the versatile use of activity diagrams in the school setting, highlighting their applicability across various age groups, from

¹ <https://www.rfdz-informatik.at/materialboerse/> (note: site in German)

kindergarten to university. Activity diagrams serve as visual representations of temporal sequences, incorporating branching paths and providing a structured overview of processes. Before delving into specific examples, we briefly recap what activity diagrams are and how we use them in the educational context.

Activity diagrams find valuable application in visualizing daily routines, such as tooth brushing, and biological processes, such as the life cycle of a butterfly. By presenting these activities graphically, students are better able to comprehend and engage with the steps involved, fostering their understanding of the sequences. Additionally, activity diagrams aid in maintaining a sense of organization, helping students remember tasks or plan for future activities. While particularly beneficial for younger students, activity diagrams can also be utilized across different age groups. Their applicability extends beyond specific subjects, making them a valuable resource in various educational subjects. For example:

Interdisciplinary (School Projects):

Activity diagrams are useful for students in the lower grades who are planning and describing scenes for video creation, particularly for interactive videos that involve multiple storylines (using tools like H5P).

Computer Science:

In the field of game programming and design, activity diagrams serve as a crucial component in outlining planned sequences of events, to visualize and comprehend the flow of a game.

Mathematics:

Activity diagrams can be used to describe specific calculation algorithms, facilitating the understanding of complex mathematical processes. They can be utilized to illustrate the step-by-step procedures involved in solving mathematical problems.

Science:

Activity diagrams find application in representing procedures or processes in various scientific subjects. For instance, they can be employed to illustrate chemical reactions, laboratory procedures, and scientific experiments.

4.1 Example: Lina’s Routine of the Day (preschool)

Now we present an example derived from a collaborative project with a local kindergarten, where the daily routine of a fictional child named Lina is represented, displayed in figure 1. This activity diagram serves as an introductory example in our teaching materials and workshops, showcasing how activity diagrams can be used and providing an understanding of their structure. Additionally, it facilitates discussions about various familiar routines and processes, such as morning or bedtime rituals, teeth brushing, and hand washing.

The initially presented diagram showcases the sequential activities (with photographs, because the children² can not read yet) from Lina having breakfast at home to her time in kindergarten, including lunch. After lunch, she either

² We use “children” to refer to students, particularly the younger ones.

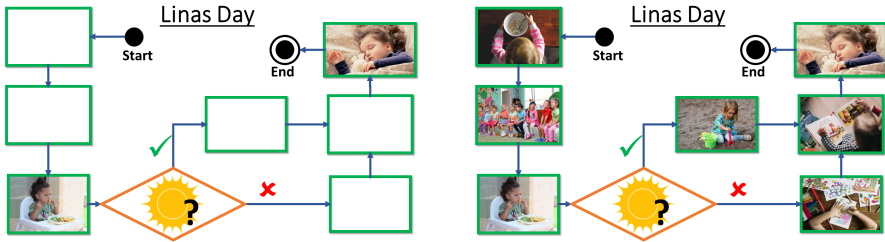


Fig. 1. Linas Day: An activity diagram for kindergarten children (left: diagram with empty fields, right: completed puzzle)

plays outside in the garden or engages in indoor drawing and coloring, depending on the weather (introducing conditions and branches). At home, she enjoys a bedtime story, concluding her day as she drifts off to sleep.

Students are encouraged to actively participate and complete provided activity diagrams like puzzles. For example, a diagram where certain activities are missing and need to be placed in the correct sequence. Additionally, arrows may be absent, and students are tasked with adding them to specify the flow of the activities. The underlying examples can encompass a wide range of processes tailored to different age groups, thematic interests, or connections to other topics: explaining the rules of simple games, outlining the plot of a fairy tale, or presenting basic cooking or baking recipes.

Considering the chosen age group in this kindergarten example, it is recommended that the children do not create their own activity diagrams but instead engage in collaborative project work with the guidance of kindergarten personnel. However, in higher age groups, allowing students to draw and create their own activity diagrams is both feasible and encouraged, as it promotes individual expression and deeper engagement with the concept and can also be used for higher purposes.

4.2 Experiences and Teachers' Reports

Simple sequences can already be used in kindergarten, e.g. in the form of a picture story (only sequences, without loops and branches). This is already understandable for children without reading skills. Simple properties (here actually only the sequence) can already be experienced or discussed. Simple branching (with simple conditions, e.g. Is the sun shining? yes/no) can also be used. The representation often automatically results in loops with simple conditions and thus already maps the most important control structures.

As part of a project conducted in collaboration with a local kindergarten, the use of visual activity diagrams was tested and successfully utilized. Even after the project concluded, these diagrams continued to be utilized in the university's computer science workshop and became a consistent component of workshops for kindergarten children. The children thoroughly enjoy the puzzle-like nature

of working with activity diagrams, as it facilitates discussions about various routines—an area of focus in kindergarten where children develop their own routines and engage in conversations about them. The project’s materials were also presented and discussed in professional development sessions and workshops for kindergarten personnel, receiving positive feedback. Many participants expressed an interest in creating their own daily schedules with personalized photos and marking the current activity on a large bulletin board. This approach provides a structure for the children, allowing them to anticipate upcoming activities and feel familiar with the day’s plan, which is particularly beneficial during atypical routines. Additionally, during some workshops, a beneficial and motivating activity involved printing out the activity diagram steps on large sheets of paper and placing them on the floor. The children would then physically step on each activity, mimicking its execution. This approach worked well for activities with a physical or sportive aspect, enabling the students to imitate specific steps, such as pretending to apply soap from a dispenser or simulating towel drying.

Similar instructional examples utilizing activity diagrams can be found in other age groups, from primary school to university students, because the complexity of the examples and diagrams can be increased accordingly. Illustrating game rules through activity diagrams is an approach for different age groups and can enhance comprehension and provide a clear overview, facilitating the learning process and game introduction. Notably, activity diagrams offer a more organized representation compared to textual descriptions, making them easier to understand and comprehend. While in schools, students often create activity diagrams manually by hand, at the university level, digital tools such as Power-Point graphics, Visual Paradigm, and similar software are commonly employed. In these groups, the focus of using activity diagrams may shift from simply discussing and understanding processes to planning a (programming) project.

We also employ activity diagrams in a teacher education course that introduces pre-service teachers to computational thinking concepts. UML diagrams, including activity diagrams, are utilized to plan the creation of educational videos. These diagrams serve as a blueprint for implementation, and students receive feedback from their peers before proceeding. A similar approach is applied in computer science workshops focused on video production (using stop-motion), particularly in the lower grades. Students plan their videos using object and activity diagrams. The posters with the diagrams help students maintain focus and keep track of the scenes.

5 Interdisciplinary Teaching Examples for Story Plot Development and Book Reports

In this section, we present the utilization of object and class diagrams in the school setting, especially in different subjects. Object diagrams are beneficial for visualizing complex information regarding concrete objects with properties and relationships between each other, while class diagrams provide a higher-level representation of properties and relationships between object types.

Object diagrams can be employed in various subjects, such as literature, film studies, and non-fiction text analysis. They assist in designing main characters for stories, summarizing non-fiction texts, and characterizing figures in films, books, or plays. Object diagrams provide a clear and concise visualization of important information, making it easier to evaluate content and develop structures before writing a text. Most students already have early experience with similar diagrams: fact sheets, where there are objects that have properties and property values. Hence, this is a good starting point and a good way to build on already existing knowledge from students.

The very common class diagrams, on the other hand, offer a general representation of relationships between object types. Unlike object diagrams that emphasize concrete instances, class diagrams focus on object types and their relationships in a higher-level representation. Although in computer science, class diagrams are often introduced first and objects are instantiated. But according to our experience and discussions with teachers, for students, the path from the concrete to the abstract is in this case usually easier to follow.

Both types of diagrams can be used with different age groups. However, we recommend starting with object diagrams only in late primary school and introducing class diagrams later, towards the end of lower secondary school (or even at the beginning of upper secondary school), as they require more abstract thinking. Both diagrams can (again) be used in a wide range of subjects:

Interdisciplinary, Language Classes:

Object diagrams can be used for outlining the main characters from a film, book, or play, together with their characteristics and their relevant relationships and interconnections. They can also be very helpful in developing characters for own stories. Class diagrams can be utilized to illustrate the elements and structure of more generalized things, like literary genres (e. g. fairy tales).

Economics and History:

The diagrams can help model the relationships between different actors, such as customers, products, and services, providing a visual representation of their interactions in economic settings. In History, they can assist in visualizing hierarchies, political systems, social structures, and relationships between historical events or developments.

Natural Sciences:

With Class diagrams, it is possible to illustrate the classification and hierarchy of living organisms, showcasing their relationships and categorizations. Object diagrams can be beneficial in modeling the components of ecosystems or food webs, helping students grasp the complex interactions and dependencies between different organisms.

Mathematics:

Class diagrams can also be used for teaching geometry, visualizing the ordering and similarities of geometric objects such as rectangles, squares, rhombuses, and parallelograms.

Computer Science:

Class diagrams are widely used to depict the relationships between classes,

objects, and components in software systems, facilitating system design and development. Sometimes they are also used in the context of database design (e. g. at our university). Object diagrams can be utilized to visualize the instantiation and interactions between objects in software applications.

5.1 Example: Story Plot Development in School Projects (lower grade)

We now focus on the development and analysis of story plots as a good example of the usage of object diagrams. What follows is an explanation of the purpose and advantage of using diagrams when developing plots. What follows next, is a teaching example that can be part of interdisciplinary teaching, school projects, and also single subjects (e. g. in the creation of videos on subject-related topics) using this approach. Hence, this type of diagram is a main part of our workshops for computational thinking and video planning and production. This is, by the way, the most booked workshop at our lab and we also published the used learning material for teachers to use.

When writing stories (as text development or as the basis for a film or a performance), the concept is an elementary part. Before the development of the linear text, the story should already have been designed and thought through. For many students, however, it is precisely this part that is less interesting and is often skipped. For a good story, however, not only the plot is important. If you only concentrate on the plot, you will not create a well-developed, thought-out story. The characters are also important. When developing a concept, people often only think about the plot development. They do not consider what characteristics the figures have, how they relate to each other, and how they develop or integrate their abilities into the story. Even if these considerations do not explicitly become part of the text, they may change the formulations and form more of an interconnected story in which the characters are not arbitrarily interchangeable. Since many students like to skip this point, it may be necessary to frame it differently, to clarify the advantages, and to convey this method as a profitable possibility.

In our teaching example (used in the mentioned workshops), we use object diagrams to do this. We use them to design the main characters of stories when writing fairy tales or planning videos. We introduce this diagram type based on already familiar fact sheets. The lessons follow the same steps as described in the example of the activity diagram in the section above: After an introductory phase, students actively engage by completing or enhancing incomplete diagrams to familiarize themselves with object diagrams. Subsequently, they are tasked with creating their own object diagrams for the purpose of planning their own story, mostly in the form of group projects. These elements are then arranged on a poster and later connected. This approach facilitates group discussion and the development of better storylines. The resulting poster or diagram serves as a valuable tool for organizing the story, aiding in maintaining a coherent narrative, and reducing content-related errors, such as inconsistencies with characters or objects, during the writing process. Moreover, for teachers, it is easier to evaluate

the content in this form than a linear text and provide early feedback, because it allows them to develop the structures, additions, extensions, etc. before starting to write a text, which is then more difficult to change.

5.2 Experiences and Teachers' Reports

Experience with the concept of designing stories and planning videos with object diagrams has yielded positive results, both in workshops and also two academic courses (one in teacher training on computational thinking and one as part of an extension study program). The material therefore is frequently used and not surprisingly making it the most popularly booked workshop in our lab. During the development process, we experimented with other types of diagrams and approaches, but they did not work that well. For instance, we attempted to use Entity-Relationship (ER) diagrams in earlier examples. However, this approach is more challenging for the students, because of the level of abstraction. They needed to work with (concrete) objects first. Consequently, we switched to using object diagrams, which have been much more successful, clearer, and easier for the students. In previous applications, object diagrams were mainly used. In some cases, an initial abstraction has already been made by considering some common features of the objects (e.g. which properties all persons have). Subsequently, these approaches could be extended to the design of class diagrams. This would also allow stories to be analyzed on a meta-level (e.g. protagonist, antagonist, ...). In this step from the concrete to the abstract students can develop the competencies to think about abstract concepts. These competencies can then be used in contexts where class diagrams fit better (e.g. non-fiction texts in science or mathematics). In practice, particularly in the field of IT, class diagrams are frequently used and serve as a good foundation for understanding technical descriptions, especially those related to software. Therefore, they are usually introduced in computer science classes, but students sometimes encounter difficulties. According to our experience and discussions with teachers, following the path from the concrete to the abstract is usually easier for students. They need to be capable of thinking more abstractly, making it more suitable for older age groups and even not recommended when working with instantiated objects. A similar approach to the teaching example above can be applied to text or story analysis, where the main characters from a book, film, or play, along with their characteristics and relationships, are outlined. This can also be done with non-fiction texts. We have tested such an approach in a cooperative school project, but it did not provide presentable data yet. But it showed the importance of a good introduction of the diagram type as well as giving examples and exercises to the students prior to them creating diagrams on their own. Otherwise, they will likely create some sort of mind map instead. But it also showed that, with object diagrams, important information can be visualized clearly and concisely, even when dealing with numerous objects and relationships. Evaluating the content in this form is easier than with linear text. This approach is beneficial for works such as the Nibelungen saga or complex films like "Fantastic Beasts," which are sometimes discussed in English classes. These stories involve multiple characters

with distinct traits and various connections to one another. Object diagrams can aid in preparing for or analyzing films and plays, allowing students to become familiar with the characters, their key characteristics, and their relationships. Therefore, they can be a valuable component of literature analysis.

6 Conclusion and Future Work

In this paper, we have demonstrated the diverse applications of UML diagrams in the classroom. We have presented UML diagrams in non-computer-science subjects. We have provided detailed explanations of teaching and working with activity diagrams, object diagrams, and class diagrams, all accompanied by teaching examples and discussions of our experiences. By showcasing the versatility of UML diagrams, we aim to inspire other educators and foster cross-disciplinary exchanges. Of course, not everything and every topic have to be structured and analyzed with UML diagrams. But we experienced that many students have problems with structuring information and focusing on relevant elements and UML therefore can be a versatile and easy method for students as well as teachers. We also want to mention that we see structure not as a limiting factor to creativity but as a method to express (and formulate) one's own, creative ideas. We hope that our insights and examples serve as a valuable resource for teachers, encouraging them to explore the potential of UML modeling in their classrooms. In the future, we will continue working with the material and improve it further as well as develop other material in our lab. We also want to do further research and gather scientific data on the presented approaches.

References

1. Lehrplan der Volksschulen (02-2023), https://www.ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2023_II_1/Anlagen_0004_4CCCAF59_2631_4F7A_90D0_D7A4C7D611CC.pdf#sig
2. Lehrplan für allgemeinbildende höhere Schulen (1985-2023), <https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10008568>
3. Lehrplan Pflichtgegenstand Digitale Grundbildung (2022), https://www.ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2022_II_267/BGBLA_2022_II_267.html
4. Aguayo, C., Eames, C., Cochrane, T.: A framework for mixed reality free-choice, self-determined learning. *Research in Learning Technology* **28**, 1–19 (03 2020)
5. Barab, S., Squire, K.: Design-based research: Putting a stake in the ground. *Journal of the Learning Sciences* **13**, 1–14 (01 2004)
6. Reiska, P., Möllits, A.: Interdisciplinary of concept maps. In: *Proceedings*. p. 151 ff. CMC 2018 (2018)
7. Sabitzer, B.: A neurodidactical approach to cooperative and cross-curricular open learning: "COOL informatics". Habilitation, University of Klagenfurt (2014)
8. Seidl, M., Brandsteidl, M., Huemer, C., Kappel, G.: *UML@classroom: Eine Einführung in die objektorientierte Modellierung*. dpunkt-verlag, Heidelberg (2012)
9. Wing, J.M.: Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **366**(1881), 3717–3725 (2008)

Identifying Computational Thinking Behaviors in the Robotics Programming Activity*

Megumi Iwata¹[0000-0001-7944-5157], Kateryna Zabolotna¹[0000-0003-3920-9859],
Kati Mäkitalo¹[0000-0003-4037-2872], Jari Laru¹[0000-0003-0347-0182], and
Jonna Malmberg¹[0000-0002-8890-4068]

University of Oulu, Pentti Kaiteran katu 1, 90570 Oulu, Finland
megumi.iwata@oulu.fi

Abstract. Robotics programming is a context which can enhance the development of computational thinking (CT) by demonstrating abstract concepts in a concrete way. Assessing applied CT components from behavioral observation has not yet been studied extensively. We aim to advance the means to identify CT behaviors in a robotics programming activity. In this paper, we introduce a pilot study where we examine an existing CT behavior scheme by applying it into the robotics programming contexts. The ninth grade students worked on the tasks of building the robot arm and programming it to make different movements. We analysed the video data to identify CT behaviors in their interactions. All items in the CT behavior scheme were observed in the robotics programming activity. CT behaviors were observed most in the independent programming phase. The students applied CT while testing the commands and debugging. In the building phase the complex process with the physical materials increased the opportunity to apply decomposition and abstraction. The results indicate that the structured instructional design of the activity minimised the students' freedom, creativity and thinking, which might limit the chance to apply CT components. We conclude that CT behavior scheme can be used in the robotics programming contexts, however, modifications in the descriptions of the items to adjust to the robotics contexts are necessary. The findings contribute to the CT research community by advancing the methodological approach of assessing CT and by deepening the understanding of the robotics programming as the contexts to use the CT components.

Keywords: Computational thinking · Behavioral observation · Robotics programming

1 Introduction

Computational thinking (CT) is considered as the cognitive aspect of computational literacies [10] and the necessary skill for everyone [21]. In Finland, CT is

* This study was supported by the GenZ strategic profiling project of the University of Oulu funded by the Academy of Finland [project No.318930] and the University of Oulu. The data collection was carried out with the support of Leaf Research Infrastructure, University of Oulu, Finland.

a part of the national framework of digital competence which applies to early childhood education and care, pre-primary education and nine-year of primary and lower secondary education (comprehensive school) [6]. CT has been studied most extensively in the past 10 years [15]. Yet, definition of CT is still under discussion. One of the frequently used definitions was provided by Aho summarising CT as “the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms” (p. 832) [1]. Recognising CT as a transferable competence beyond programming and computing, Ezeamuzie and colleagues defining CT as “the cognitive skill required to design algorithmic solutions for problems in different knowledge areas” (p.502) [5].

In more practical level, researchers have proposed operational definitions of CT mapping the concepts, the processes, and the practices underlying when CT is applied. In this paper, we call them as CT components. We select the four CT components: problem decomposition, abstraction, pattern recognition and algorithm design, which are identified as the common CT components [3,20]. Decomposition is understood as simplifying a problem/task by dissecting it into manageable parts that can be addressed separately [3,17]. Abstraction is a practice of concentrating on the essential part to solve a problem/task while concealing minor details [3,5]. Pattern recognition is identifying patterns/rules based on observations and applying these rules to a other instance [3,17]. Algorithm design is formulating logical and ordered instructions to solve a problem/task translating abstract ideas into concrete procedure [3,17].

Robotics programming can be an effective way to introduce CT in K-12 education [12,16]. A constructivist learning environment can support understanding and demonstrating the abstract concept of CT in a concrete way [8]. In addition, the use of physical materials increases cognitive challenges by adding another layer of complexity in the activity, which can enhance the use of CT [13,9]. The previous study indicates that the middle school students prefer to learn computing with hands-on physical computing environments compared to screen-based computing environments [12]. The authors indicate that open-ended activity design appears to increase students’ interests and confidence in coding despite of the additional challenges of troubleshooting with physical components.

There has been an increasing interest in studying the assessment of CT in the past years. The most common assessment methods of assessing CT competences are pre- and post-test, self-report and artifact analysis [19]. Using observation as a assessment method allows assessing CT competences directly in the activity which provide richer and content-related information of students’ CT competences [14]. The structured systematic observation was used to assess the students’ competences in the block-based programming contexts using Scratch [14]. One of the scales of the structured systematic observation was “computational concepts” which lists six items to assess the students’ competence. Observational analysis was used to identify the development of CT of 5-year-old children [18]. The authors assessed CT through analysing the movement of the robot that the children controlled. However, assessing CT through behavioral observation has not been well studied [3].

In this study, we aim to advance the behavioral observation method to identify applied CT components from the students' interactions in the robotics programming activities. We conduct a pilot study where we apply an existing framework of CT behavior scheme to examine its usability in the robotics programming contexts. We set the following two research questions.

- 1 Which CT components are present in the robotics programming activity?
- 2 To what extent are the CT behaviors observed in students' interactions in the different phases of the robotics programming activity?

This study contributes CT research community by deepening the conceptual understanding of CT in robotics programming contexts and by developing behavioral observation as a methodological approach to identifying the use of the CT components.

2 Related Work

2.1 CT in the Robotics Programming Contexts

In robotics activities, students are encouraged to actively interact with the physical objects, to play and to design. Often the robotics activities were designed as student-centered and facilitated by the instructors who help students in the programming and other challenges [8]. A previous study shows that minimum instructions encourages students' trials-and-errors behavior [4].

There are a view of CT in robotics that claims that CT appears when the students program to make the robot move [8,4]. On the other hand, there is broader view that CT is not limited to the programming part of the robotics activities, but it appears in all phases of robotics activities. The latest study propose six components of CT in robotics activities recognising CT as an approach applying computer science concepts and procedures to understand, model, and solve a problem using tangible objects or programming software [13]. The proposed components are divided into three categories: problem analysis, systems and creation, which includes two components in each category: identify the problem, organise and model the problem, formal system (software), physical system (tangible objects), devise a solution, and evaluate the solution and perform iteration [13].

2.2 Frameworks for Identifying CT Behaviors

To our knowledge, two frameworks for observing CT behaviors have been introduced. Computational Thinking Behavior Scheme (CTBS) was developed by Boom and colleagues to identify CT behaviors in block-based programming contexts using Scratch [3]. The framework defines four CT components (decomposition, abstraction, pattern recognition and algorithm design). CTBS lists 11 items as behavior indicators to identify the applied CT components. The observation of CT behaviors are done by measuring the time spent rather than correctness

of the applied CT [3]. The authors found the correlation between the applied CT components and the quality of the end results (the programming projects in Scratch). They note that the instrument was developed specifically to the context of their study, thus the items should be examined carefully.

Another framework focuses on the CT components in the robotics contexts. Keith and colleagues developed the coding system for CT in the robotics activity for the children aged 8-13 using LEGO Mindstorms EV3 robotics kit [11]. The framework categorises four CT components (analysis, algorithmic thinking, debugging and designing). The framework was developed considering the characteristics of the robotics contexts. For example, design of the robot is considered as one aspect to support the use of CT components [11].

After examining the two frameworks above, we decided to use CTBS in the following pilot study. The reason is because, firstly, it was valid in some extent as the authors found that the CT components observed with CTBS are correlated with the quality of the program. Secondly, lack of items related to the programming in the framework by Keith and colleagues can be challenging to adapt to our pilot study, which has both building and programming phases.

3 Research Methods

3.1 The Context of the Study: Robotics Programming Workshop

Participants of the Workshop This pilot study was implemented in the two-day robotics programming workshop organised and facilitated by the researchers. The workshop was offered as a part of a week-long work-life orientation¹, which is required in the Finnish National Core Curriculum for Basic Education. Participants of the workshop included 21 ninth grade students in the international school in Finland. Among several other options, the students chose to participate in this workshop by themselves. There was no requirement of the prior knowledge to participate for the workshop. Nevertheless the students had been introduced to programming within the Finnish National Core Curriculum for Basic Education.

Description of the Tasks The goal of the workshop was building the robot arm and programming it to make different movements. The robot arm called "Alvin" and the software application to program Alvin were designed and provided by a company in Finland². In the workshop, the students worked as a group of two or three following the provided instructions. No strict timetable was provided, thus each group worked on the tasks on their own pace. The instructors were available if the students needed additional help. The tasks of the workshop were divided into two phases: building phase and programming phase, as described below.

¹ Työelämään tutustuminen (TET) in Finnish

² <https://simua.com/>

Building Phase In the building phase the students assembled the robot arm (Fig. 1). The robot arm was made from the pre-cut wooden parts, six servo motors and a microcontroller (ESP32). The students assembled them using the screws, the hammer and other tools. The base part of the robot arm can be rotated and the arm part can be flexed and extended with the joints. It has the hand part that can rotate, open and close to grab objects. The video instructions were provided to support the students in the building phase. The purpose of this phase was to build the same functional robot that can be programmed rather than designing and creating their own artifacts using their creativity. Although these instructions explained the steps in detail, the students had a chance to make decisions of how they approach to the tasks.

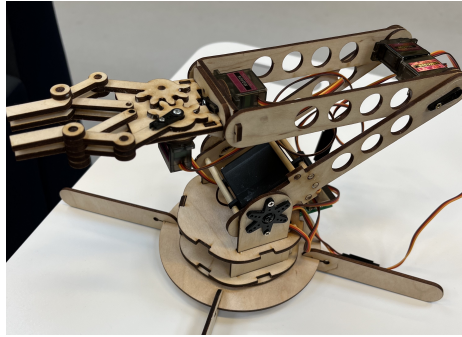


Fig. 1: Robot arm Alvin

Programming Phase After building the robot arm, the students started programming to make different movements. The goal of the programming phase was creating procedures (collection of movements) by combining the different pre-programmed movements. The pre-programmed movements included turn, bow and bend, and open and close the hand, which can be used with the structures, such as for-loop, while-loop. Programming was done by sending the commands through the software application on the tablet device. The interface of the software application was designed as if communicating with the robot with text messages (Fig. 3). For instance, to bend the robot arm, the movements of each servo motor can be programmed by sending the commands, such as "move the servo 1 for 40 degrees". The printed booklet instructions³ including guidance for programming and the exercises were provided (Fig. 2).

Guided Programming Phase During the programming phase, the instructor held the short lecture, where they went through the beginning of the programming part in the booklet instructions together with the students. They explained how

³ The booklet was still a prototype and not a commercial product.

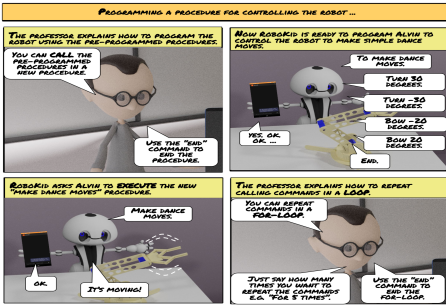


Fig. 2: Booklet instructions

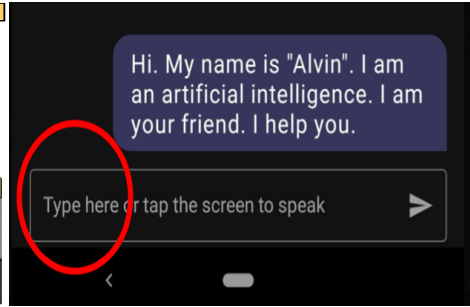


Fig. 3: Programming interface

to send the commands, how to create classes, how to test the commands in the interface, and how to create and execute procedures. In the data analysis, we called this lecture as guided programming phase, and other moments, when the students worked on the programming exercises by themselves, as independent programming phase.

3.2 Data of the Study and Analysis

Data Collection and Ethical Consideration The data collection was carried out with the support of Leaf Research Infrastructure⁴, at the University of Oulu, Finland. The data of this pilot study was the video of a group of two students recorded on the second day. The video included both building and programming phases. The length of the video was 8394 seconds (140 minutes). The video was taken in front of the desk where the students were sitting. On the desk, the robot arm was clearly visible. The video did not show the programming interface, so the log data was used to understand the processes of the programming phase.

The data collection was carried out as a part of a larger research program, the GenZ profiling project of the University of Oulu. It was approved by the ethical committee of the university and was carried out according to the national and international ethical guidelines [2,7]. Informed consent was collected from both participants and their guardians. Students' names were replaced with ID numbers (pseudonymized) prior to data processing.

Data Analysis The data analysis was conducted in *the Observer XT* environment. The unit of the data analysis was group rather than individuals. We first identified the process of the activity by categorising into three phases: building phase, individual programming phase and guided programming phase. We then divided the students' interactions into "on-task interaction" and "off-task interaction" to remove irrelevant moments. The interactions between the students

⁴ <https://www.oulu.fi/en/research/research-infrastructures/leaf-research-infrastructure>

and the instructors, which subject was related to the task, were coded as an on-task interaction. After that, we conducted coding of the CT behaviors based on CTBS. Appendix 1 shows CT behavior scheme by Boom and colleagues [3] with the adjustments in the behavior indicators to make it suitable to the contexts. We changed the term "code blocks" and "code chunks" to "commands" and "story or game" to "pieces". In the analysis we focused on "meaningful episode" in the activity, where the students applied the CT components. A meaningful episode included the students' interactions with silence of less than 10 seconds. We calculated the duration and the frequency of each item in Observer XT in order to analyse how the CT components appeared during the activity.

We did not include as CT behavior when the students were working on the task but the interactions did not include their active thinking on the task. For instance, when the students were identifying the next step by just checking the instruction without discussing it, we did not code as CT behavior.

Student A: *OK, what's next?*

Student B: (Watching the video) *Ah... this part.*

Neither we coded as CT behavior if a student just tell the other what to do.

Student A: *Take that off.*

Student B: *OK.*

Another example of non-CT behavior was when the students had an unclear issue but did not engage in discussing.

Student A: *Is there a limit to the bend?*

Student B: *I don't know. (Change the subject.)*

4 Results

Table 2 shows the length of each phase. The building phase lasted for 3249 seconds (54 minutes, 38.7% of the total length of the video). It started in the beginning of the activity as the students were continuing building the robot arm from the previous day. The independent programming phase took place after the building phase for 3751 seconds (63 minutes, 44.7% of the total length of the video). The guided programming phase, which was in the middle of the independent programming phase, was 1168 seconds (19 minutes, 13.9% of the total length of the video). The length of the on-task interactions was 5696 seconds (95 minutes, 67.9% of the total length of the video).

Table 1: Phases of the robotics programming activity

	Building	Independent programming	Guided programming	Other	Total
Duration (s)	3249	3751	1168	226	8394
Percentage (%)	38.7	44.7	13.9	2.7	100.0

4.1 CT Behaviors Observed in the Robotics Programming Activity

The CT behaviors observed in the three phases of the robotics programming activity are summarised in Table 3 in Appendix 2. All four CT components in CTBS were observed in the video data. Total duration of the meaningful episodes was 2596 seconds (43 minutes) and it was 31% of the total length of the video. Total frequency of the moments, which CT behaviors were observed, was 158 times. Among the CT components, "algorithm design" (ALG) was observed most frequently (133 times) and the longest duration (2082 seconds). The second most frequent CT component was "decomposition", which total duration was 299 seconds and observed 36 times. Between the two least observed CT components, "abstraction" was observed longer duration (136 seconds) than pattern recognition (80 seconds).

Independent Programming Phase The CT behaviors were observed the most in the independent programming phase. In total the CT behaviors were observed 119 times and the duration was 1812 seconds which is around 70% of the total CT behaviors. In the independent programming phase, most of the CT behaviors were directly related to the process of designing algorithms in the commands. Below is an example of CT behaviors from the independent programming phase with the CTBS items assigned (see Appendix 1).

Example The students were teaching the robot a new movement "dance move" which combined several movements that they have previously tested. Student A read the instructions and suggested the steps to proceed while Student B held the tablet to program the robot arm.

Student A: *Let's start (writing) "To make dance move" (command) and then "Make dance move" (command)* [DCM1, DCM2].

Then two students discussed the content of the commands, for example, degrees of turning and the order of the movements in "the dance move" [ALG1]. After that, the students sent the commands and observed the robot arm checking whether the robot arm moved as expected [ALG2]. As there was an error, the students tried to fix the error [ALG3].

Building Phase About 20% of the CT behaviors were observed in the building phase. In the building phase, the students had many problems with the physical materials, where the CT behavior of debugging was observed. In addition, the process of the building was complex as they needed assemble many parts in the correct orders. In such process, the CT behaviors of decomposition and abstraction was used. Below is an example of debugging in the building phase, which was coded under the CTBS item [ALG3].

Example The students were trying to assemble a part on the robot arm with several screws. Student A held the screw driver and Student B held the part and the robot arm together. While screwing the students realized that the part was

not aligned anymore. The students decided to unscrew and try again.

Student B: *Wait wait wait. You are not...*

Student A: *Ohh it is not?*

Student B: *You're punching through the thingy (wooden piece) for some reason.*

Student A: *Right. Maybe it's not aligned with the hole.*

Student B: *There is no hole, right?*

Student A: *There's a hole used to be.*

Student B: *There is?*

Student A: *Yeah there is. Wait. Right, let's totally unscrew this and then...*

Guided Programming Phase In the guided programming phase, the CT behaviors were least observed. Less than 10% of the CT behaviors were present in the guided programming phase. As an example of CT behaviors from the guided programming phase is as follows.

Example The students were learning the basics of programming following the instructor's example command. The students discussed the content of the example command [ALG1]. Then they checked that it worked [ALG2].

After that the students wanted to customise the example command but they did not find how to do it. The students asked the instructor. The below interaction was coded as pattern recognition and algorithms [PTR2, ALG1].

Student A: *What if we later on want to add something into the procedure? How to do that?*

Instructor: *You can ask Alvin how to do that. And you can copy paste, and modify and edit any text, and overwrite. . .*

Student A: *Ah.*

Student B: *Select all, copy, then paste.*

5 Discussion and Conclusion

5.1 Usability of CTBS in the Robotics Contexts

In this study we tested the existing framework of identifying the behaviors for applied CT components in the new context. All the items in CTBS were observed in the video data of the pilot study. Thus, even though CTBS was originally developed for the block-based programming context, it is possible to apply it into the robotics programming contexts. The CT behaviors were observed most in the independent programming phase. This is understandable as it is closest to the context which CTBS was originally developed. In the guiding programming phase, the least CT behaviors were present. This was also expected as the students were mostly listening the instructor's short lecture and there were less interactions between the students.

The building phase, which is unique to the robotics contexts, although the CT behaviors were observed, the overall presence was far less than the independent programming phase. One reason is because the coding framework was not

optimised for the robotics contexts. Some of the behavioral indicators in CTBS, such as PTR2 ("Use of copy-paste"), were more closely related to programming (see Appendix 1). The behavioral indicators can be modified further to adapt the characteristics of the robotics contexts. For instance, in the building phase, the students observed the model of the robot arm while assembling their own. This could be considered as copy-paste in robotics contexts as they were transferring the observed constructions to their own context. Another item: ALG1 ("Putting commands together"), could be modified to the robotics programming contexts by adding descriptions related to physical materials, such as "Identifying the steps and the order to assemble the parts".

5.2 Influences of the Activity Design on the Use of CT

Complexity in Building Phase "Decomposition" and "abstraction" were somewhat observed in the independent programming and the building phases. It was more present in the building phase than programming phases. One reason might be that there were more complexity in the building phase, including many steps involving different physical parts needed to be assembled in the correct orientation and the correct order. The use of CT in the complex process including physical materials are discussed also in the previous studies [13,9]. In the complex process, the students had to discuss and agreed on the steps to proceed. Compared to the building phase, the programming phase was more simplified and straightforward. The level of programming was rather low, the interface was easy to use and the detailed instructions were provided.

Structured Activity with Step-by-Step Instructions Both "Decomposition" and "abstraction" are considered as part of problem analysis [13]. Because the instructions were provided, which included each step to build the robot arm and many example commands, in most of the tasks the students did not need to analyse the problems and plan the procedure by themselves. The complex problems were already divided into small pieces and written as step-by-step instructions. The students were strictly following the instructions, thus their focus was fixed to the immediate next steps rather than the further goals. This may be the reason of why DCM2 ("Identifying the immediate next step") was most observed among "decomposition" items. If the students have more freedom in the tasks, they may have more chances to plan their tasks.

Limited Possibility of Customisation "Pattern recognition", which was the least observed CT components in our study, is the skill to identifying the similar characteristics (rules) to transfer them into the different contexts [3]. This can appear, for example, in the process of customising the commands. However, in the pilot study, the students did not have enough opportunity to customise the commands by themselves as the instructions in the activity were quite complete and not encouraging making modifications. Design and creativity are the aspects that is important in the robotics contexts [11,13]. Building of the robot arm could

be designed as ill-structured allowing the students to personalise the robot, which can increase their interests [12] and the chance to apply the CT components.

5.3 Limitation of the Study and Future Research Direction

A limitation of this study is the small amount of data. Only part of one group's activity was analysed. This was partially because the aim of the pilot study was to test the preliminary usability of CTBS in the robotics contexts for the further development. The next step of the research is to test the updated CTBS with other participants in the same workshop. Another limitation is related to the data analysis. In the data analysis, the on-task interactions with maximum of 10 seconds silence was conducted before coding of the CT behaviors. Thus, some moments of applied CT components, which were not discussed between the students, might be excluded from the CT behaviors. For example, in the guiding programming phase, the students were mostly silence, although they were following the instructions and programming the robot arm. This needs to be examined carefully in the future research. Finally, due to the time constraint, validation of the coding was incomplete. As the coding process of identifying CT behavior is subjective, we recognise that the validation of the coding scheme is the immediate next task.

To conclude, this study found the possibility of assessing applied CT components through behavioral observation in the robotics programming activity using CTBS. The framework should be adjusted to the contexts and further examination is needed. The robotics programming contexts added complexity in the activity by involving physical materials. The instructional design should be considered carefully to increase the chance to apply CT in the activity. In a structured activity, students' freedom and creativity are minimised and it can result in limiting their own thinking and the opportunity to use CT. The activity should be designed to maintain the balance of task difficulties and the space for student's freedom, creativity and thinking.

6 Appendix 1

CT components	Behavioral indicator	Example episode
Decomposition (DCM)	DCM1: Putting problem into pieces / building sub problems DCM2: Identifying the immediate next step DCM3: Discussing if then relations of the pieces (is related to programming elements)	Students discuss the current step as a part of the overall process: <i>Should all the pieces have to be assembled?</i>
Abstraction (ABS)	ABS1: Focusing on important information; neglecting unimportant details ABS2: Simplifying anything problem, sub problem, functions, commands, etc.	Students neglect a part of the process: <i>We don't need to do everything yet.</i>
Pattern recognition (PTR)	PTR1: Identifying similar characteristics (sub problem, functions, commands etc.) PTR2: Use of copy-paste PTR3: Aha moments (must be related to an event when student understood relationship between things)	Students recognise the similarities in the process: <i>It's all of these, right? So just go like this. I think it's just way easier.</i>
Algorithm design (ALG)	ALG1: Putting commands together ALG2: Testing and judging algorithm (i.e., run a set of code and actively observing a running sequence) ALG3: Debugging - try to find error and adjust algorithm	Students discuss which parts should be assembled in which order: <i>This part needs to be put through here, and after that put this one.</i>

CT components and the behavior indicators were adapted from CT behavior scheme by Boom and colleagues [3]. The terms in the behavior indicators were modified and the example episodes were added by the authors.

7 Appendix 2

CTBS item ¹	Building		Independent programming		Guided programming		Phases total	
	Duration ²	Freq. ³	Duration	Freq.	Duration	Freq.	Duration	Freq.
DCM1	- (-)	- (-)	8 (100.0)	1 (100.0)	- (-)	- (-)	8 (100.0)	1 (100.0)
DCM2	157 (55.8)	14 (42.4)	125 (44.2)	19 (57.6)	- (0.0)	- (0.0)	282 (100.0)	33 (100.0)
DCM3	9 (100.0)	2 (100.0)	- (0.0)	- (0.0)	- (0.0)	- (0.0)	9 (100.0)	2 (100.0)
DCM total	166 (55.7)	16 (44.4)	125 (44.3)	19 (55.6)	- (0.0)	- (0.0)	299 (100.0)	36 (100.0)
ABS1	101 (100.0)	2 (100.0)	- (-)	- (-)	- (-)	- (-)	101 (100.0)	2 (100.0)
ABS2	24 (68.8)	1 (50.0)	11 (31.2)	1 (0.0)	- (0.0)	- (0.0)	34 (100.0)	2 (100.0)
ABS total	125 (92.1)	3 (75.0)	10.7 (31.2)	1 (25.0)	- (0.0)	- (0.0)	136 (100.0)	4 (100.0)
PTR1	19 (100.0)	2 (100.0)	- (-)	- (-)	- (-)	- (-)	19 (100.0)	2 (100.0)
PTR2	- (-)	- (-)	- (-)	- (-)	17 (100.0)	1 (100.0)	17 (100.0)	1 (100.0)
PTR3	- (-)	- (-)	31 (70.0)	1(50.0)	13 (30.0)	1(50.0)	44 (100.0)	2(100.0)
PTR total	19 (23.8)	2 (40.0)	31 (38.5)	1 (20.0)	30 (37.7)	2 (40.0)	80 (100.0)	5 (100.0)
ALG1	4 (0.9)	1 (2.8)	318 (68.0)	30 (83.3)	145 (31.1)	5 (13.9)	447 (100.0)	36 (100.0)
ALG2	20 (2.6)	1 (2.1)	716 (96.5)	45 (93.8)	7 (0.9)	2 (4.2)	742 (100.0)	48 (100.0)
ALG3	287 (30.8)	7 (24.1)	605 (69.2)	22 (75.9)	- (0.0)	- (0.0)	874 (100.0)	29 (100.0)
ALG total	293 (14.1)	9 (8.0)	1638 (78.7)	97 (85.8)	152 (7.3)	7 (6.2)	2082 (100.0)	133 (100.0)
CT total	603 (23.2)	30 (19.0)	1812 (69.8)	119 (75.3)	181 (7.0)	9 (5.7)	2596 (100.0)	158 (100.0)

¹ See Appendix 1 for the descriptions of CTBS items.

² Duration of the item in the phase. Unit is second. () after the value indicates the percentage of the total duration of the item in all phases.

³ Frequency of the item in the phase. Unit is n. () after the value indicates the percentage of the total frequency of the item in all phases.

References

1. Aho, A.V.: Computation and computational thinking. *The computer journal* **55**(7), 832–835 (2012)
2. ALLEA - All European Academies: The european code of conduct for research integrity revised edition (2017), <https://www.allea.org/wp-content/uploads/2017/05/ALLEA-European-Code-of-Conduct-for-Research-Integrity-2017.pdf>, last accessed 14 August 2023
3. Boom, K.D., Bower, M., Siemon, J., Arguel, A.: Relationships between computational thinking and the quality of computer programs. *Education and Information Technologies* **27**(6), 8289–8310 (2022)
4. Chevalier, M., Giang, C., Piatti, A., Mondada, F.: Fostering computational thinking through educational robotics: A model for creative computational problem solving. *International Journal of STEM Education* **7**(1), 1–18 (2020)
5. Ezeamuzie, N.O., Leung, J.S.: Computational thinking through an empirical lens: A systematic review of literature. *Journal of Educational Computing Research* **60**(2), 481–511 (2022)
6. Finnish National Agency for Education: The framework for digital competence (2023), <https://eperusteet.opintopolku.fi/#/en/digiosaaminen/8706410/osaamiskokonaisuus/8709075>, last accessed 4 August 2023
7. Finnish National Board on Research Integrity TENK: The ethical principles of research with human participants and ethical review in the human sciences in finland (2019), https://tenk.fi/sites/default/files/2021-01/Ethical_review_in_human_sciences_2020.pdf, last accessed 14 August 2023
8. Ioannou, A., Makridou, E.: Exploring the potentials of educational robotics in the development of computational thinking: A summary of current research and practical proposal for future work. *Education and Information Technologies* **23**, 2531–2544 (2018)
9. Iwata, M., Pitkänen, K., Laru, J., Mäkitalo, K.: Exploring potentials and challenges to develop twenty-first century skills and computational thinking in k-12 maker education. In: *Frontiers in Education*. vol. 5, p. 87. Frontiers Media SA (2020)
10. Kafai, Y.B., Proctor, C.: A revaluation of computational thinking in k–12 education: Moving toward computational literacies. *Educational Researcher* **51**(2), 146–151 (2022)
11. Keith, P.K., Sullivan, F.R., Pham, D.: Roles, collaboration, and the development of computational thinking in a robotics learning environment. *Computational thinking education* pp. 223–245 (2019)
12. Love, T.S., Asempapa, R.S.: A screen-based or physical computing unit? examining secondary students’ attitudes toward coding. *International Journal of Child-Computer Interaction* **34**, 100543 (2022)
13. Romero, M.: Assessment of computational thinking in an ill-defined problem-solving task with modular robots. In: *International Conference of the Learning Sciences (ICLS)*. Montreal, Canada (2023)
14. Saez-Lopez, J.M., Marcos, R.G., Esteban, V.C.: Visual programming languages integrated across the curriculum in elementary school: A two year case study using “scratch” in five schools. *Computers & Education* (2016)
15. Saqr, M., Ng, K., Oyelere, S.S., Tedre, M.: People, ideas, milestones: a scientometric study of computational thinking. *ACM Transactions on Computing Education (TOCE)* **21**(3), 1–17 (2021)

16. Shahin, M., Gonsalvez, C., Whittle, J., Chen, C., Li, L., Xia, X.: How secondary school girls perceive computational thinking practices through collaborative programming with the micro: bit. *Journal of Systems and Software* **183**, 111107 (2022)
17. Shute, V.J., Sun, C., Asbell-Clarke, J.: Demystifying computational thinking. *Educational research review* **22**, 142–158 (2017)
18. Terroba, M., Ribera, J.M., Lapresa, D., Anguera, M.T.: Observational analysis of the development of computational thinking in early childhood education (5 years old) through an intervention proposal with a ground robot of programmed directionality. *European Early Childhood Education Research Journal* **30**(3), 437–455 (2022)
19. Tikva, C., Tambouris, E.: Mapping computational thinking through programming in k-12 education: A conceptual model based on a systematic literature review. *Computers & Education* **162**, 104083 (2021)
20. Wang, C., Shen, J., Chao, J.: Integrating computational thinking in stem education: A literature review. *International Journal of Science and Mathematics Education* **20**(8), 1949–1972 (2022)
21. Wing, J.M.: Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **366**(1881), 3717–3725 (2008)

Computational Thinking Readiness of Incoming High School Students in Taiwan

Greg C. Lee¹[0009-0002-5624-543X], Jia-Yi Chen²[0009-0003-9929-4263], and Yu-Wen Yang³[0009-0009-4612-3595]

¹ National Taiwan Normal University, Taipei, Taiwan
leeg@csie.ntnu.edu.tw

² National Taiwan Normal University, Taipei, Taiwan
81147002S@ntnu.edu.tw

³ National Taiwan Normal University, Taipei, Taiwan
wen860806@gmail.com

Abstract. The new national K-12 curricula went into effect four years ago in 2019. Of which, the technology curriculum has shifted towards cultivating Computational Thinking (CT) and programming skills of students. The first group of students who completed middle school education under the new national curriculum entered high school in fall of 2022. A multi-year study is underway to evaluate how the new curriculum has improved students' CT skills. A CT and programming assessment tool was created for this study. Two types of tasks, namely goal-based and problem-based tasks, were designed to test different CT and programming skills. The two goal-based tasks require pattern recognition and generalization CT skills as well as simple repetition and selection programming skills to solve. The two problem-based tasks additionally require students to have good abstraction skills in solving the given tasks. In this paper, results from the first year of on-going study are reported. A total of 17 schools, 130 classes and 4,475 students participated in this study. The incoming high school students were tested during the first four weeks of classes before additional programming lessons were conducted. Thus the result reflects students' learning outcome from middle schools. Overall, the majority of students were able to solve the two goal-based tasks (94% and 92%). However, only less than one fifth of students were able to solve the two problem-based tasks (20% and 10%). This result showed that students need more practice to improve their abstraction skills. Further analysis of students' programs showed that students have the most difficulty in using variables. Findings from this study provide good feedback to middle school teachers. Furthermore, statistical data provides a good baseline for future studies.

Keywords: Computational Thinking · Assessment types and tasks · Online assessment system · CS education

1 Introduction

In recent years, research on Computational Thinking (CT) has focused on the development of effective instructional strategies, the cultivation of foundational

skills required, and the formulation of assessment methods and frameworks for CT. In 2019, the new K-12 national curriculum went into effect, which also mandates to have Information technology (IT) classes in each of the two semester of grades 7-9. The new IT curriculum emphasizes on Computational Thinking and programming for problem solving. Thus, the curriculum is more oriented toward computer science than computer applications. Many grades 7-9 teachers in Taiwan, as with many other countries, uses Scratch [7], Blockly [1], MIT App Inventor [4], Greenfoot [3], etc. as computer programming learning tools in practice. Understanding the effectiveness of curriculum implementation is one of the important goals after the implementation of the new curriculum. However, previous assessments of Computational Thinking lack quantitative tools that can be used for a large number of tests, so it is difficult to provide a comprehensive report on the effectiveness of curriculum implementation.

In 2010, Koh, Basawapatna, Bennett, and Repenning [2] developed a visual semantic assessment tool called the Computational Thinking Pattern (CTP) graph particularly for student-created games and simulations. The graph can be used to indicate the existence of CT transferred from games to science simulations. The Fairy Assessment [8] was proposed a few years later in another study to assess two aspects of CT skills, thinking algorithmically, and making effective use of abstraction and modeling. In the assessment, although students were limited to the Alice based learning environment, this study has been regarded as one of the major developments in the assessment of CT skills. In 2015, Moreno-Leon & Robles [5, 6], assessed students' CT skills with "Dr. Scratch" by analyzing students' visual programming projects. Dr. Scratch is an analytical tool that automatically analyzes Scratch projects and assigns a CT score in terms of abstraction, decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, and data representation. The tool demonstrates how students' programming skills can be improved with the feedback provided.

In summary, there have been some studies evaluating students' CT abilities, some of them assessed students' specific CT abilities by pre-designed tasks, or conversely, directly analyzing students' existing game projects to understand their CT abilities. In this study, we conducted a large scale experiment to assess incoming high school students' Computational Thinking readiness after they have had three years of middle school computer classes. The assessment was based on actual problem solving tasks through programming that require different types of CT and programming skills. In the following sections, the assessment tasks are first explained before the research setup and finding are presented.

2 Chippy Assessment Tool

The Chippy assessment is composed of two types of tasks, with two goal-based tasks to assess students' pattern recognition ability and simple programming ability to perform the same routine repetitively. Students are shown an animation of the task at hand and must recognize the repetitive pattern in the animation before writing Scratch or Blockly code to complete the task. Students can run

their program and see an animation of the effect of executing their program code step by step. On the other hand, the problem-based tasks are described in words with sample input and output. The tasks are already familiar to the students so are easy for them to understand. The problem-based tasks have test data that correspond to different possible instances/cases of the problem. When students' programs are executed, feedback can then be provided to alert them of different scenarios that have not yet been correctly considered.

The four tasks used in this study are given in Fig. 1. The two goal-based tasks are Light and Robot Vacuum. The Light task repetitively projects different colored light onto the stage. The projected light color is determined by switching on or off the red, blue, and green lights above the stage. Students must watch the animation, observe the repetitive color pattern of the projected lights, and write program to perform the same task. For the Robot Vacuum task, the task animation shows that the robot goes around the classroom to clean dust off the floor. Once students recognize the same turning/moving directions can be used to sweep each quarter section of the classroom, the program code are quite simple as shown in Fig. 1(b).

The two problem-based tasks are Cake Promotion and Drink Orders. The Cake Promotion task asks students to calculate the discount, the shipping fee, and the total price. Students must be able to set proper variables, break problem into taking order and calculation subtasks before designing proper algorithm to solve the problem. The Drink Orders task is similar to the Cake Promotion task but added requirement to use list/array data structure and wrinkle to compute subtotal/total of the order. Both tasks require students to demonstrate good abstraction, problem decomposition and algorithmic design skills. The two tasks have 5 and 6 different possible cases, respectively, that students should consider when designing algorithm. Possible correct programs for the two tasks are shown in Fig. 1(c) and (d). The CT skills needed to complete each task is summarized in Table 1.

Table 1. Computational Thinking skill required for each task.

Tasks \ Skills	Pattern Recognition	Abstraction	Decomposition	Algorithm Design
Light	✓			
Robot Vacuum	✓			✓
Cake Promotion		✓	✓	✓
Drink Orders		✓	✓	✓

result instantly. Furthermore, students were given feedback on where or why the program was not given a 100 score. For the goal-based tasks, animation of students' program will reflect the moves as instructed, giving students visual cue to where the algorithm failed. For the problem-based tasks, all possible problem scenarios were listed and those that were solved incorrectly were clearly marked.

3.4 Expected Outcome

The maximum score for the assessment test is 400. Having completed the Information Technology curriculum in middle schools ($7^{th} \sim 9^{th}$ grades), high school freshmen (10^{th} grade) are expected to have the competency to complete both goal-based tasks and at least one of the two problem-based tasks with 45 minutes. Students with more practices or can think more quickly can possibly complete all four tasks. Therefore, as shown in Table 2, with each task having a score of 100 points, the **expected** total score of students is between 251 and 350, an equivalent of completing 2.5 to 3.5 tasks. Scores above 351 indicates having **excellent** CT and programming skills. Scores between 151 and 250 indicates not being able to complete one problem-based task and thus only having **moderate** CT and programming skills. Any score below 150 indicates not being able to complete even the two goal-based tasks; therefore, having **inadequate** or **no** CT and programming skills.

Table 2. The CT skills description corresponding to each score range.

Score Range	Relative to the $7^{th} \sim 9^{th}$ grades IT curriculum
351-400	Excellent CT and programming skills.
251-350	Expected CT and programming skills.
151-250	Moderate CT or programming skills.
51-150	Inadequate CT and programming skills.
0-50	No CT and programming skills.

4 Results and Findings

4.1 Quantitative Analysis

The average score among all 4,475 students participated in this study is 198. Given the expected score of 251 or better, this average score suggests that the CT readiness of incoming high school students, in general, is somewhat below expectation. Table 3 gives the number and the percentage of student scoring in each score range. It can be seen that only 12% (9%+3%) of students performed as expected or better, while most students (71%) exhibited moderate CT and programming skills. It is alarming that close to one fifth (2%+15%) of students still have inadequate CT skills as they enter high schools.

Table 3. Number of students and percentage of students in each score range.

CT/Prog. Skills	No CT	Inadequate	Moderate	Expected	Excellent	Total
Score Range	0-50	51-150	151-250	251-350	351-400	Avg. = 198
No. of Students	74	661	3193	392	155	4475
% of Students	2%	15%	71%	9%	3%	100%

Next, we look at the results by task. Table 4 shows the descriptive statistics. The table shows that majority of students attempted the two goal-based tasks (99% and 96%). Furthermore, majority of students did complete these two tasks (94% and 92%) correctly. This shows that students are capable of finding repetitive patterns from the given problem animation and write programs to perform the same task. For the problem-based tasks, only 56% and 35% of students attempted the two tasks, respectively. This shows that close to half of students either did not attempt or ran out of time to solve problem-based tasks. Of those attempted problem-based tasks, a majority of students (73% and 87%) were not able to receive any partial scores, while only 20% and 10%, for the two tasks, respectively, of student were able to receive full score. These statistical results show that majority of students were not proficient in problem analysis and Computational Thinking, which led to difficulty in solving the problem-based tasks.

Table 4. Number of people and percentage in each score range.

Task	Attempted	Average Score	Score Distribution
Light	99%	93.6	<p>A bar chart showing the score distribution for the 'Light' task. The y-axis represents percentage from 0% to 100%. There are two bars: a blue bar at 0% labeled '6%' and a blue bar at 100% labeled '94%'. Below the x-axis, the categories are '# 0' and '# 100'.</p>
Robot Vacuum	96%	92.2	<p>A bar chart showing the score distribution for the 'Robot Vacuum' task. The y-axis represents percentage from 0% to 100%. There are two bars: a blue bar at 0% labeled '8%' and a blue bar at 100% labeled '92%'. Below the x-axis, the categories are '# 0' and '# 100'.</p>
Cake Promotion	56%	22.7	<p>A bar chart showing the score distribution for the 'Cake Promotion' task. The y-axis represents percentage from 0% to 100%. There are four bars: a blue bar at 0% labeled '73%', a blue bar at approximately 10% labeled '5%', a blue bar at approximately 20% labeled '2%', and a blue bar at 100% labeled '20%'. Below the x-axis, the categories are '# 0', '# 1-50', '# 51-99', and '# 100'.</p>
Drink Orders	35%	12.0	<p>A bar chart showing the score distribution for the 'Drink Orders' task. The y-axis represents percentage from 0% to 100%. There are four bars: a blue bar at 0% labeled '87%', a blue bar at approximately 10% labeled '0%', a blue bar at approximately 20% labeled '2%', and a blue bar at 100% labeled '10%'. Below the x-axis, the categories are '# 0', '# 1-50', '# 51-99', and '# 100'.</p>

4.2 Qualitative Analysis

After looking through students' programs for the problem-based tasks, two observations can be made about student's CT and programming abilities.

1. Poor understanding and proper usage of variables

Although students have learned to use variables, proper usage of variables requires high level of abstraction skill. In general, students do have understanding of storing values in variables, but often can only use variables as constants. For example, in Fig. 2(a), although student's program did use a variable "Cake" to store the number of cakes ordered, that program failed to declare a second variable to keep track of the running total. In this case student did not know how to "update" variable value as required in this task. As another example, in Fig. 2(b), the program had five variables to keep track of the drink prices, but also used the same variables for checking the ordered drink number. Students conceptually associated drink number with drink price in the same variable; thus, unable to use one variable for order checking and another variable for overall cost computation.

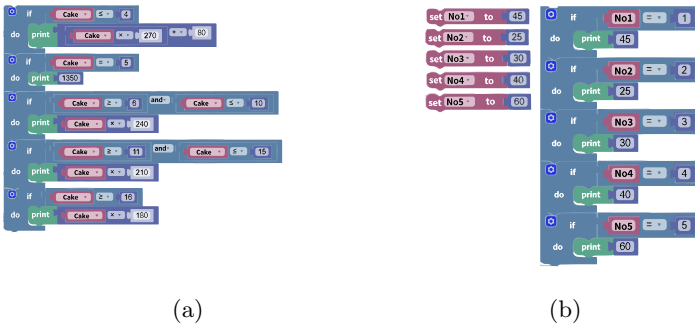


Fig. 2. Sample student program for (a) Cake Promotion, (b) Drink Orders tasks.

2. Inefficient formulation of different problem instances

Another common problem exhibited by students' programs is that the program did not properly condition different cases of the problem with variables. Furthermore, many programs used multiple *if* statements, instead of nested *if-then-else* statement to match natural logical reasoning of different cases. For example, in Fig. 3(a), the program failed to use *if-then-else* structure, but the actual error lay in not being able to keep a running total using a second variable. In Fig. 3(b), in addition to being unable to read in value for variable *a* and the lack of a variable for running total again, the program used five *if-do* statements to check for drink order number. In both of these examples, students decomposed the problem into a few independent cases. In fact these should be exclusively disjoint cases.

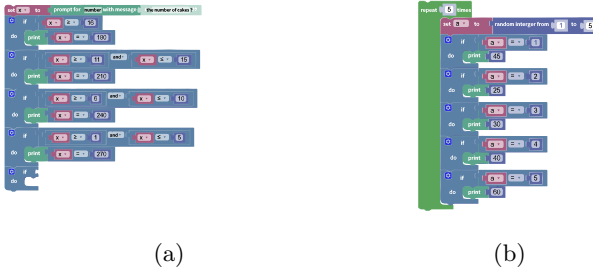


Fig. 3. Sample student program for (a) Cake Promotion, (b) Drink Orders tasks.

5 Conclusions

In this study, incoming high school students were put to the test to assess their CT and programming learning achievement from their middle school information technology education. The results were rather surprising and alarming. Two recommendations are relayed back to the middle school teachers. First, when training students to think computationally, there is a need to focus more on abstraction of problems, including formulating different problem instances logically. Secondly, more examples and practices are needed to help build conceptualization and good usage of variables.

In summary, this first year study provides a good baseline for future studies. We will continue to conduct this study annually, expanding to more schools and classes. Qualitative results will lead to development of teaching strategies to meet the learning objectives of the information technology curriculum.

References

1. Blockly, <https://developers.google.com/blockly>. Last accessed 3 June 2023.
2. Koh, K.H., Basawapatna, A., Bennett, V., Repenning, A.: Towards the automatic recognition of computational thinking for adaptive visual language learning. 2010 IEEE Symposium on Visual Languages and Human-Centric Computing. (2010).
3. Kölling, M.: The Greenfoot Programming Environment. *ACM Transactions on Computing Education*. **10**, 1–21 (2010).
4. MIT APP Inventor, <https://appinventor.mit.edu/>. Last accessed 3 June 2023.
5. Moreno-León, J., Robles, G.: Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills. Presented at the Scratch Conference, 12–15 (2015).
6. Moreno-León, J., Robles, G.: Dr. scratch: a Web Tool to Automatically Evaluate Scratch Projects. *Proceedings of the Workshop in Primary and Secondary Computing Education*. 132–133 (2015).
7. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y.: Scratch: Programming for All. *Communications of the ACM*. **52**, 60–67 (2009).
8. Werner, L., Denner, J., Campe, S., Kawamoto, D.C.: The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. (2012).

Insights and Conclusions from Analyzing the Hungarian Bebras Initiative in 2021-2022

Zsuzsa Pluhár¹[0000-0003-2688-4652] and Bence Gaál¹[0000-0001-8771-7140]

¹ Eötvös Loránd University, Budapest, Faculty of Informatics
pluharzs@ik.elte.hu, gaalbence@inf.elte.hu

Abstract. Empowering young people to become active agents in modern society and shape the world around them is a critical imperative for education. It is essential to inspire and equip them with the skills and knowledge to make a positive impact on their communities and the world at large. In addition, digital skills and computational thinking are crucial components of modern education that can enable young people to navigate an increasingly complex and technologically advanced world. Developing these skills can not only enhance their employability but also empower them to find innovative solutions to global challenges and drive positive change in society.

The Bebras initiative, with over 70 countries participating, is a successful approach to involving school students in computer science problem-solving and deep thinking. In Hungary, the Bebras challenge is conducted in cooperation with VILLE, and the results from 2021 were analyzed in detail. Further analysis was carried out on the 2021 and 2022 tasks separately, raising additional research questions that led to the development of a methodology and research project. The present study summarizes these analyses, along with the methodology and research project, and establishes additional research questions.

The methodology and the findings of this new research project can have an impact on the development of computer science challenges in the topic of problem-solving, and motivation in CS and CT and they can serve as a guide for future studies and analyses in this field.

Keywords: Bebras challenge, Computational Thinking, motivation in Computer Science, Computer Science education.

1 Introduction

Computational thinking (CT) is a crucial skill for all students in the 21st century, not just those pursuing computer science or mathematics. However, the concept can be complex and is often misunderstood as being narrowly related to computing or computers. [1] The significance of computational thinking has aroused the swift advancement of educational initiatives and programs dedicated to its cultivation. The development of students' computational thinking skills presents challenges in terms of its adaptability and emphasis across diverse educational environments, as it extends be-

yond the boundaries of programming and computer science, encompassing interdisciplinary aspects. [2]

Applying CT in education significantly enhances students' capacity for critical and analytical thinking, fostering their aptitude to think systematically and computationally. [3] Thinking computationally refers to the utilization of computer science principles for problem-solving purposes. This cognitive process encompasses the systematic decomposition of a problem into more manageable subcomponents, the identification and analysis of patterns and interconnections within the problem domain, the formulation of algorithmic representations or procedural instructions to address the problem systematically, and the critical evaluation of the solution's suitability and effectiveness in achieving the intended objectives.

1.1 The Bebras Challenge

The Bebras Challenge [4, 5, 6] is an international initiative aimed at promoting alongside the computational thinking (CT) the computer science and informatics among primary and secondary school students. The challenge consists of solving short concept-based tasks that are based on informatics concepts and can be answered in a few minutes. Organized annually for almost twenty years, the challenge has gained popularity in many countries as an informal school event. Participants in the challenge are usually supervised by teachers who may integrate the challenge into their teaching activities. The challenge is structured into six age groups, each with its own set of tasks: Pre-Primary (grades 1-2), Primary or Little Beavers (grades 3-4), Benjamins (grades 5-6), Cadets (grades 7-8), Juniors (grades 9-10), and Seniors (grades 11-12). Students solve 18 to 24 tasks in 45-55 minutes.

The Bebras Challenge aims to make informatics education more attractive for learners by providing problem-solving experience and insight into what lies beyond digital technology. [4] It provides a platform for students to showcase their abilities and compete with other participants from diverse schools and nations. This event enables them to connect with like-minded individuals and potentially form new friendships within their area of interest.

The Bebras Challenge, which has a strong resemblance to the CS unplugged methodology [7], is primarily conducted online. Its tasks are designed to be solved without the aid of a computer, promoting hands-on learning. The tasks are concise, engaging, and centered around fundamental computer science concepts, enabling students to comprehend the underlying principles behind technology. The CS unplugged methods are highly effective in providing students with a glimpse into another fascinating aspect of computer science - the inner workings of a computer. [4,6]

Overall, the Bebras Challenge is an excellent way to promote informatics education and CT among students, and it has been successful in promoting CT, as evidenced by the increasing number of participants each year. [8] The challenge's structure, which includes different age groups and sets of tasks, makes it accessible to a wide range of students, and its online format makes it easy to participate in from anywhere in the world.

1.2 The Bebras Challenge in Hungary

Since 2011, the Hungarian Bebras Challenge [7,8] has been running with the goal of promoting informatics and CT among students. The challenge aims to motivate students in exploring computing, develop problem solving and critical thinking skills, provide activities that can be done in classrooms without requiring computers, and suggest school and after-school activities to teachers.

The number of participants has been steadily increasing each year (see Fig.1), with students of various ages and schools taking part in the challenge. Participants are given 45 minutes to solve tasks online, with easy, medium, and hard levels of difficulty to choose from.

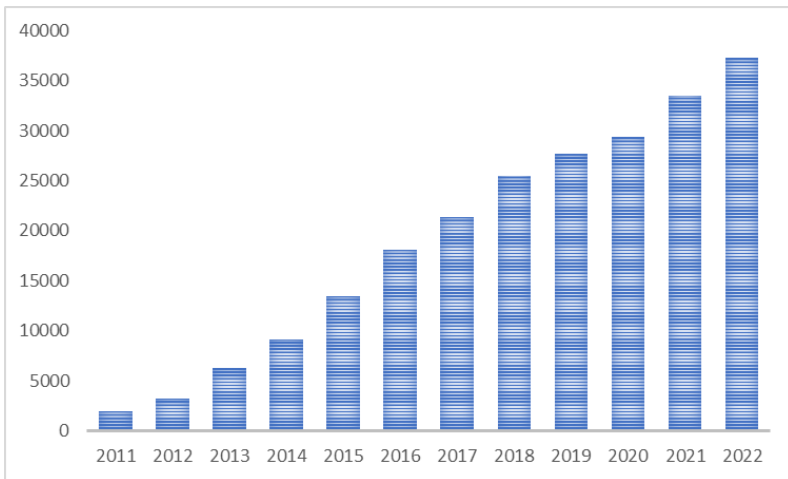


Fig. 1. The number of participants in the Hungarian Bebras challenge from 2011 to 2022.

Tasks are always presented in the same order based on their difficulty, and participants are allowed to switch between tasks while they compete. To prevent students from feeling discouraged by negative marks for incorrect answers, the Challenge awards bonus marks to all participants at the beginning.

In cooperation with Lithuania, Finland and India the Hungarian Bebras organizers started in 2021 a new project to use a learning analytics enriched environment, ViLLE [8]. The number of the participating countries was extended in 2022 with Sweden, Ireland, Spain, as well.

Multiple choice questions were used in the first year (2021), with four options. It was improved to use interactive tasks in 2022.

2 Analysis of the results of the Challenge in 2021-2022

2.1 Methods

We analyze the scores and the differences in scores between girls and boys every year. In addition to the descriptive statistics, the Independent T-test (on score's interval scale) is executed using SPSS.

When we analyzed the results in each task separately, our aim was only to analyze the students' success without being influenced by the task's difficulty level. So, we normed the results of the students for -1 (wrong answer), 0 (not answered), and +1 (correct answer) and ran Mann-Whitney tests in ordinary scales.

2.2 Success in the Tasks

2021.

In 2021, 33467 students participated in 5 age groups. There was no big difference between the number of girls and boys, only in the oldest age group (senior), where the difference between the number of boys and girls was more than two times. (See Fig. 2)

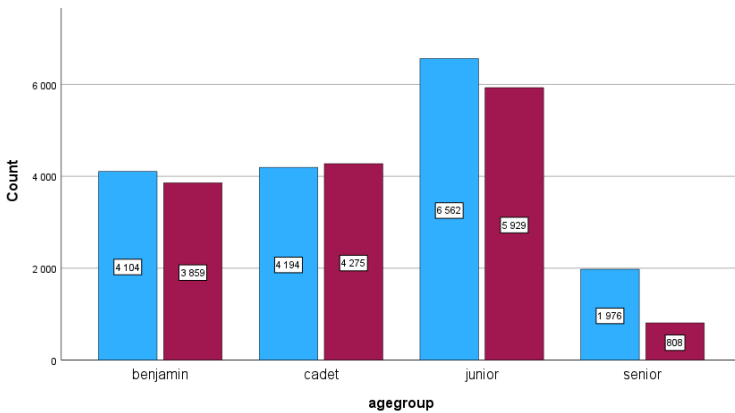


Fig. 2. The number of participants in the Hungarian Bebras challenge 2021 clustered by age groups and gender.

The first analysis showed significant differences between boys and girls only in the total scores of the two oldest age groups. (see Table 1)

	sex	N	mean	sd	t	df	p
Juniors	girls	5929	112.20	36.486	3.129	12459	<.001
	boys	6562	110.21	34.606			
Seniors	girls	808	89.06	28.193	5.690	1668	<.001
	boys	1976	96.00	31.590			

Table 1. descriptive statistics about Junior and Senior categories in 2021

Because of this first result, our deeper, task and gender separated analysis focuses on the 2 oldest age groups.

More descriptive analysis of the difficulty level shows that success in task-solving moves together in the case of girls and boys in both age groups. However, some can be seen some differences (see Fig.3 and Fig 4.).

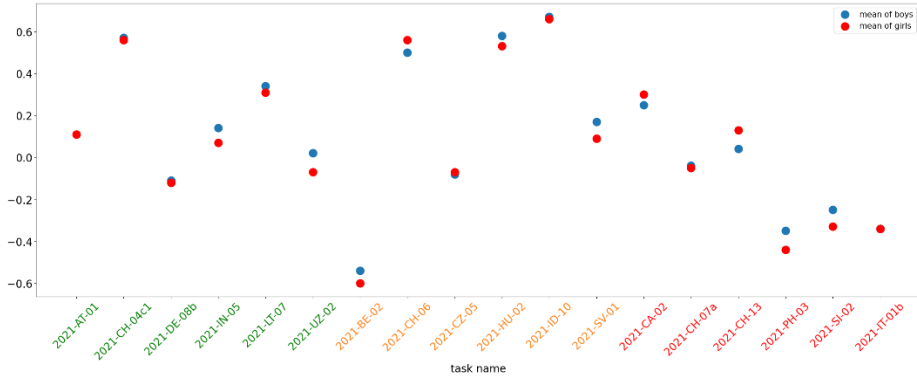


Fig. 3. The means of the normalized results in the Hungarian Bebras challenge 2021 in Junior age group for the used tasks.

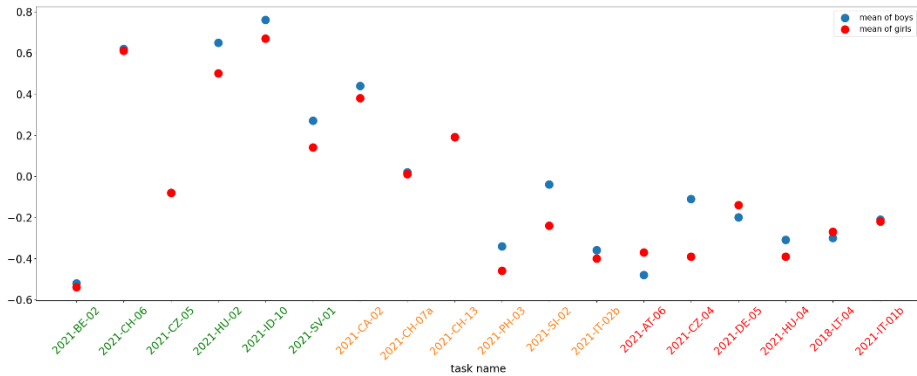


Fig. 4. The means of the normalized results in the Hungarian Bebras challenge 2021 in Senior age group for the used tasks.

To analyze deeper the tasks more separately we collected the tasks for each age group where significant differences in the success can be found (see Appendix A).

We used 25 tasks in these two age groups, and 13 tasks had shown a significant difference for girls in 4, and for boys in 9 cases.

We could not find a relationship between the significant differences and the difficulty level of the tasks. Some tasks were wrongly graded for difficulty level (see Fig 3

and Fig. 4), the number of correct solutions was higher (easier than graded) or lower (harder than graded).

In Junior level two tasks showed lower numbers in solutions, were harder as expected (2021-BE-02 and 2021-UZ-02) and showed significant differences. The other tasks that showed a significant difference were either easier or at an appropriate level of difficulty.

In senior level 3 easy, 2 medium and 2 hard tasks showed significant differences and except for one task (2021-PH-03), they were of the expected level of difficulty and performed according to their classification for the age group (see Fig.4).

If we can find significant differences in both age groups in the same tasks, the differences are for the same gender (boys). The 3 tasks showing significant difference for girls, and used in both age groups, didn't show differences in the oldest age group.

2022.

In 2022, 37350 students participated from 283 schools in 5 age groups in the competition. There was no big difference between the number of girls and boys. The highest difference can be found again in the oldest age group (see Fig.5).

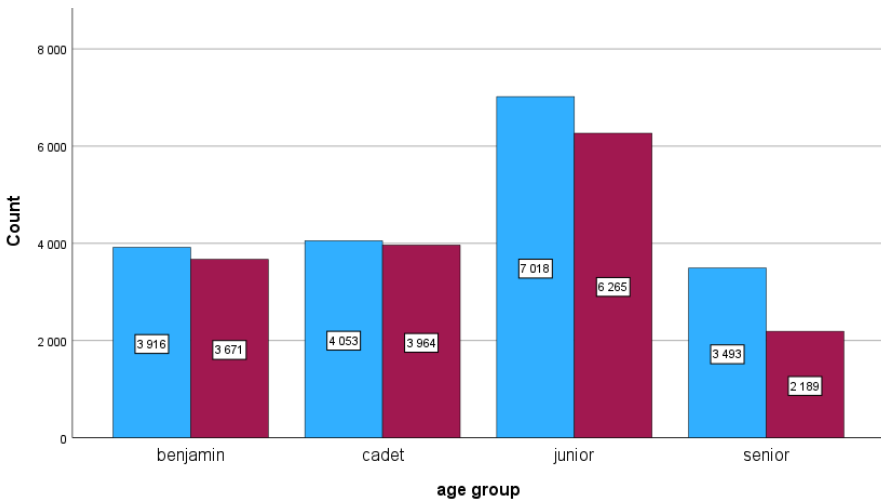


Fig. 5. The number of participants in the Hungarian Bebras challenge 2022 clustered by age groups and gender.

For this year we created the same analysis as for the results in 2021.

In 2022 we have found a significant difference in the two oldest age groups between girls and boys again (see Table 2).

	sex	N	mean	sd	t	df	p
Juniors	girls	7018	80,71	34,306	13.734	13279	<.001
	boys	6265	89,32	37,970			
Seniors	girls	2189	65,86	30,608	5.861	4943	<.001

	boys	3493	70,92	33,372		
--	------	------	-------	--------	--	--

Table 2. descriptive statistics about Junior and Senior categories in 2022

The descriptive analysis of the difficulty level shows again that success in task-solving moves together in the case of girls and boys in both age groups. However, some differences can be seen again (see Fig.6 and Fig.7).

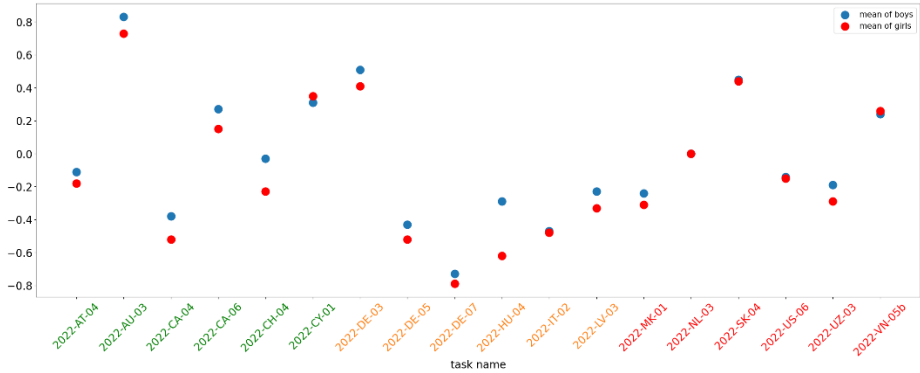


Fig. 6. The means of the normalized results in the Hungarian Bebras challenge 2022 in Junior age group for the used tasks

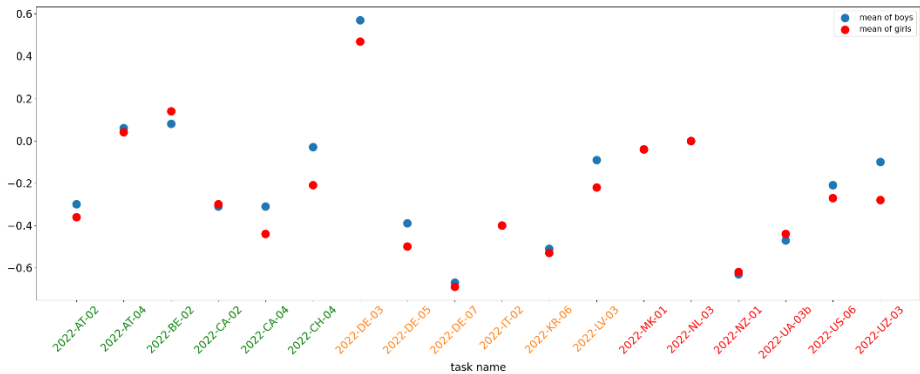


Fig. 7. The means of the normalized results in the Hungarian Bebras challenge 2022 in Senior age group for the used tasks

To analyze deeper the tasks separately we collected the tasks for each age group where significant differences in the success can be found (see Appendix B).

Of the 24 tasks used in these two age groups, 13 tasks had shown significant difference for girls in 1, and for boys in 12 cases.

In the case of using a task in more age groups we can see significant differences not in all age groups, this year as well. We could not find a relationship between the significant differences and the difficulty level of the tasks.

Overall, the tasks in the Junior age category were poorly calibrated. In general, in terms of correct answers, the tasks were more difficult for the participants than expected.

Although most of the easy tasks in the Senior age category should have been rated as medium or difficult (see Fig. 6. and Fig. 7.), they did not necessarily show a significant difference between girls and boys. Only two tasks from 6 show significant differences and was harder as expected.

If we can find significant differences in more age groups in the same tasks, the differences are for the same gender.

3 New Research – Gender Texting Analysis

We identified and will discuss the features of tasks where the impact was more relevant, and where the difference showed significance.

Our next step in this research based on your analysis is to find whether the story or the texting of the tasks influence the success of students in these tasks.

3.1 Methodology

The determination of the gender of tasks was prepared a larger-scale study involving teachers and students (participants and not participants as well).

Data collection was conducted through an online survey.

In the first part of the survey the respondents are asked to provide their gender, age group, and whether they participated in the current competition that included the tasks in question.

The second part of the survey (see Fig. 8) utilizes a 5-point Likert scale to gauge the gender of the tasks based on the task body presented. Respondents are provided with the following options to choose from:

Very masculine - Rather masculine - Neutral - Rather feminine - Very feminine



Fig. 8. An example question of the questionnaire. (Answers from left to right: very masculine, rather masculine, neutral, rather feminine, very feminine)

In the questionnaire, the participants are given the 2021 and 2022 Junior, Senior Bebras tasks and have to decide whether they considered the tasks to be masculine or feminine based on a quick reading. Participants have to make these quick and intuitive

tive judgements based only on the brief descriptions of the tasks and the associated graphical elements. In this way, the study focuses on exploring how the "gender" of the task will be perceived by the participants through their first impressions.

4 Discussion

The aim of future research on the questionnaire is to determine whether gender perceptions of tasks are synchronous in those tasks where significant differences were found between boys' and girls' scores. If task perceptions show significant differences in perceptions, it is necessary to determine whether this is caused by the text, graphic elements or other cultural preconceptions of the tasks. If necessary, replacing these elements of the tasks by repeating the research will clearly identify what is causing the gender bias. Finally, an international comparison would be undertaken, where we would look at the tasks in different countries, and through this we would investigate the role of linguistic and cultural factors.

A next step – to analyze the results and texting, visual elements of other countries and languages could extend the research.

We believe that the results of our research can greatly help the future design of the Bebras tasks to ensure that the implementation of inclusive task design is not compromised in any way, as one of the most important aspects of the Bebras Initiative is to make information technology accessible to all, creating equal conditions for challenge participants.

Another aspect of the significant difference between the results of girls and boys could show a general problem in science education based on preconceptions of gender roles. A deeper analysis of attitudes and changes of attitudes during school years can be the following research sub-topic. Based on the result of gender motivation research the aspect of the action can have another dimension.

References

1. Li, Y., Schoenfeld, A.H., diSessa, A.A. et al. Computational Thinking Is More about Thinking than Computing. *Journal for STEM Educ Res* 3, 1–18 (2020). <https://doi.org/10.1007/s41979-020-00030-2>
2. Lodi, M., Martini, S. Computational Thinking, Between Papert and Wing. *Sci & Educ* 30, 883–908 (2021). <https://doi.org/10.1007/s11191-021-00202-5>
3. Saidin, Noor Desiro & Khalid, Fariza & Martin, Rohanilah & Kuppusamy, Yogeswary & Munusamy, Nalini. (2021). Benefits and Challenges of Applying Computational Thinking in Education. *International Journal of Information and Education Technology*. 11. 248-254. [10.18178/ijiet.2021.11.5.1519](https://doi.org/10.18178/ijiet.2021.11.5.1519).
4. Dagiene, V., Futschek, G.: Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks. In: R. T. Mittermeir, M. M. Syslo (Eds.), *Informatics Education – Supporting Computational Thinking*. Lect. Notes in Computer Science. Vol. 5090, Springer, 19–30. (2008).

5. Dagiéné, V., & Stupurienė, G.: Bebras - a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in education*, 15(1), 25-44. (2016).
6. Dagiene, V., Futschek, G., & Stupuriene, G.: Creativity in Solving Short Tasks for Learning Computational Thinking. *Constructivist Foundation*14(3), 382-396. (2019).
7. Pluhár, Zs.: Extending computational thinking activities. *Olympiads in Informatics*, 15, 83-89 (2021).
8. Pluhár, Zs. et al.: Bebras Challenge in a Learning Analytics Enriched Environment: Hungarian and Indian Cases. In: Bollin, A., Futschek, G. (eds) *Informatics in Schools. A Step Beyond Digital Education. ISSEP 2022. Lecture Notes in Computer Science*, vol 13488. Springer, Cham. https://doi.org/10.1007/978-3-031-15851-3_4 (2022).

Appendix

Appendix A – Result of the tasks’ analysis in Hungarian Bebras Challenge 2021 separately:

junior					senior				
	Boys	N=6562	Girls	N=5929		Boys	N=1976	Girls	N=808
task name	mean	sd	mean	sd	task name	mean	sd	mean	sd
2021-AT-01	0,13	0,988	0,11	0,991	2018-LT-04	-0,3	0,82	-0,27	0,804
2021-BE-02	-0,54	0,836	-0,6	0,794	2021-AT-06	-0,48	0,767	-0,37	0,802
2021-CA-02	0,25	0,938	0,3	0,92	2021-BE-02	-0,52	0,844	-0,54	0,837
2021-CH-04c1	0,57	0,816	0,56	0,822	2021-CA-02	0,44	0,883	0,38	0,901
2021-CH-06	0,5	0,857	0,56	0,827	2021-CH-06	0,62	0,775	0,61	0,784
2021-CH-07a	-0,04	0,963	-0,05	0,959	2021-CH-07a	0,02	0,968	0,01	0,97
2021-CH-13	0,04	0,971	0,13	0,965	2021-CH-13	0,19	0,971	0,19	0,968
2021-CZ-05	-0,08	0,986	-0,07	0,991	2021-CZ-04	-0,11	0,95	-0,39	0,848
2021-DE-08b	-0,11	0,98	-0,12	0,98	2021-CZ-05	-0,08	0,985	-0,08	0,986
2021-HU-02	0,58	0,807	0,53	0,839	2021-DE-05	-0,2	0,906	-0,14	0,882
2021-ID-10	0,67	0,717	0,66	0,729	2021-HU-02	0,65	0,752	0,5	0,862
2021-IN-05	0,14	0,976	0,07	0,982	2021-HU-04	-0,31	0,842	-0,39	0,778
2021-IT-01b	-0,34	0,881	-0,34	0,888	2021-ID-10	0,76	0,63	0,67	0,716
2021-LT-07	0,34	0,938	0,31	0,948	2021-IT-01b	-0,21	0,945	-0,22	0,933
2021-PH-03	-0,35	0,896	-0,44	0,854	2021-IT-02b	-0,36	0,84	-0,4	0,785
2021-SI-02	-0,25	0,925	-0,33	0,895	2021-PH-03	-0,34	0,916	-0,46	0,855
2021-SV-01	0,17	0,959	0,09	0,967	2021-SI-02	-0,04	0,972	-0,24	0,922
2021-UZ-02	0,02	0,995	-0,07	0,993	2021-SV-01	0,27	0,949	0,14	0,973

id	junior	senior
2018-LT-04		no sign.dif.
2021-AT-01	no sign.dif.	
2021-AT-06		Z=-3.585; p<=.001
2021-BE-02	Z=-4.492; p<.001	no sign.dif.
2021-CA-02	Z= -2.926; p=.003	no sign.dif.
2021-CH-04c1	no sign.dif.	
2021-CH-06	Z= -3.572; p<.001	no sign.dif.
2021-CH-07a	no sign.dif.	no sign.dif.
2021-CH-13	Z= -5.036; p<.001	no sign.dif.
2021-CZ-04		Z= -6.916; p<.001
2021-CZ-05	no sign.dif.	no sign.dif.
2021-DE-03		
2021-DE-05		no sign.dif.
2021-DE-08b	no sign.dif.	
2021-HU-02	Z= -3.091; p=.002	Z= -4.784; p<.001
2021-HU-04		no sign.dif.
2021-ID-10	no sign.dif.	Z= -3.287; p=.001
2021-IN-05	Z= -4.372; p<.001	
2021-IT-01b	no sign.dif.	no sign.dif.
2021-IT-02b		no sign.dif.
2021-LT-07	no sign.dif.	
2021-PH-03	Z= -5.354; p<.001	Z= -2.997; p=.003
2021-SI-02	Z= -4.120; p<.001	Z= -4.868; p<.001
2021-SV-01	Z= -4.585; p<.001	Z= -3.192; p=.001
2021-UZ-02	Z= -4.733; p<.001	

Appendix B – Result of the tasks’ analysis in Hungarian Bebras Challenge 2022 separately:

	junior				senior				
	Boys	N=6562	Girls	N=5929		Boys	N=1976	Girls	N=808
task name	mean	sd	mean	sd	task name	mean	sd	mean	sd
2022-AU-03	0.83	0.545	0.73	0.675	2022-AT-04	0.06	0.972	0.04	0.956
2022-CA-06	0.27	0.956	0.15	0.98	2022-CH-04	-0.03	0.963	-0.21	0.937
2022-CY-01	0.31	0.946	0.35	0.929	2022-DE-03	0.57	0.785	0.47	0.83

2022-HU-04	-0.29	0.944	-0.62	0.759	2022-DE-05	-0.39	0.901	-0.5	0.828
2022-SK-04	0.45	0.883	0.44	0.886	2022-DE-07	-0.67	0.722	-0.69	0.698
2022-VN-05b	0.24	0.961	0.26	0.956	2022-LV-03	-0.09	0.986	-0.22	0.956
2022-AT-04	-0.11	0.953	-0.18	0.931	2022-CA-04	-0.31	0.936	-0.44	0.879
2022-CH-04	-0.03	0.965	-0.23	0.933	2022-IT-02	-0.4	0.893	-0.4	0.89
2022-DE-03	0.51	0.806	0.41	0.855	2022-MK-01	-0.04	0.98	-0.04	0.979
2022-DE-05	-0.43	0.872	-0.52	0.794	2022-NL-03	0	0.024	0	0
2022-DE-07	-0.73	0.634	-0.79	0.568	2022-US-06	-0.21	0.913	-0.27	0.892
2022-LV-03	-0.23	0.942	-0.33	0.907	2022-UZ-03	-0.1	0.944	-0.28	0.896
2022-CA-04	-0.38	0.882	-0.52	0.805	2022-AT-02	-0.3	0.864	-0.36	0.842
2022-IT-02	-0.47	0.821	-0.48	0.818	2022-BE-02	0.08	0.911	0.14	0.903
2022-MK-01	-0.24	0.9	-0.31	0.882	2022-CA-02	-0.31	0.821	-0.3	0.825
2022-NL-03	0	0.032	0	0.018	2022-KR-06	-0.51	0.617	-0.53	0.582
2022-US-06	-0.14	0.872	-0.15	0.872	2022-NZ-01	-0.63	0.535	-0.62	0.506
2022-UZ-03	-0.19	0.864	-0.29	0.841	2022-UA-03b	-0.47	0.53	-0.44	0.533

id	junior	senior
2022-AT-02		no sign.dif.
2022-AT-04	Z=-3.753; p<.001	no sign.dif.
2022-AU-03	Z=-9.879; p<.001	
2022-BE-02		no sign.dif.
2022-CA-02		no sign.dif.
2022-CA-04	Z=-9.293; p<.001	Z=-5.196; p<.001
2022-CA-06	Z=-7.047; p<.001	
2022-CH-04	Z=-11.723; p<.001	Z=-6.881; p<.001
2022-CY-01	Z=-2.793; p=.005	
2022-DE-03	Z=-7.418; p<.001	Z=-4.837; p<.001
2022-DE-05	Z=-5.461; p<.001	Z=-4.011; p<.001
2022-DE-07	Z=-4.972; p<.001	no sign.dif.
2022-HU-04	Z=-20.689; p<.001	
2022-IT-02	no sign.dif.	no sign.dif.
2022-KR-06		no sign.dif.
2022-LV-03	Z=-6.450; p<.001	Z=-4.693; p<.001
2022-MK-01	Z=-4.479; p<.001	no sign.dif.
2022-NL-03	no sign.dif.	no sign.dif.
2022-NZ-01		no sign.dif.
2022-SK-04	no sign.dif.	
2022-UA-03b		no sign.dif.
2022-US-06	no sign.dif.	no sign.dif.
2022-UZ-03	Z=-6.523; p<.001	Z=-6.715; p<.001
2022-VN-05b	no sign.dif.	

Poster Descriptions

Integrating Computational Thinking with Mathematical Problem Solving*

Arnold Pears¹[0000-0002-5184-4743], Javier Bilbao²[0000-0002-2784-8496],
Valentina Dagiene³[0000-0002-3955-4751], Yasemin
Gulbahar⁴[0000-0002-1726-3224], András Margitay-Becht⁵, Marika Parviainen⁶,
Zsuzsa Pluhar⁵[0000-0003-2688-4652], and Pál György Sarmasági⁵

¹ KTH Royal Institute of Technology pears@kth.se

² Applied Mathematics Department University of the Basque Country Bilbao, Spain
javier.bilbao@ehu.eus

³ Institute of Educational Science, Vilnius University, 01513 Vilnius, Lithuania
valentina.dagiene@mif.vu.lt

⁴ Ankara University, Türkiye ysmnglbhr@gmail.com

⁵ Eötvös Loránd University, Hungary abecht@inf.elte.hu pluzsu@gmail.com
psarmasagi@inf.elte.hu

Abstract. The Erasmus+ project Computational Thinking and Mathematical Problem Solving, an Analytics Based Learning Environment (CT&MathABLE) provides comprehensive learning analytics driven support for developing Computational and Algebraic Thinking in K-12 schools. Through the deployment of digital technology the project provides educators with new approaches to skills development that builds on well supported learning pathways and is individually tailored to the learner. This is achieved through a novel learning systems architecture which supports individualized development paths and integration of Computational Thinking and mathematical conceptual development with tailored problem solving and assessment frameworks.

Keywords: Computational Thinking · CT · Algebraic Thinking · AT · Mathematics Education · Curriculum · Learning Pathways.

1 Introduction

As part of curricula reforms, many European countries have already included elements of Computational Thinking (CT) skills in compulsory schooling [1]. CT is a type of analytical thinking that employs mathematical and engineering thinking to understand and solve complex problems within the constraints of the real world [3]. Algebraic Thinking is defined as the ability to generalize, represent, justify, and reason with abstract mathematical structures and relationships [2]. One of the most attractive ways to do this is by integrating AT

* This work has been funded through the Erasmus+ Programme KA220-SCH project CT&MathABLE: “Computational Thinking and Mathematical Problem Solving, an Analytics Based Learning Environment”, 2022-1-LT01-KA220-SCH-000088736.

and CT education into computer science or similar courses. For instance, the Ministry of Culture and Education in Finland highlights new literacy competencies, which include ICT skills, media literacy and programming. On the other hand, Lithuania on the other hand has introduced compulsory CT education from early grades through to the end of compulsory schooling. To address this variation in curricula, an in depth analysis of six European national curricula has been conducted to expose CT and AT content in mathematics and other subjects, initially focusing on students aged 9 to 14 years. This analysis forms the foundation for developing individualized learning pathways. Our analysis reveals considerable similarity, allowing for the development of core learning paths, however, there is considerable regional variation.

2 Method

Establishing learning pathways of broad relevance requires an in-depth understanding of the relationships between conceptual development in the domain and the linkage of this understanding to conceptual progression within each topic or skill area. We define the ability to think computationally as a combination of higher-order cognitive skills: a) abstraction, b) algorithmic thinking, c) analytical thinking and decomposition, d) data collection, analyses and representation, e) evaluation and adjustment, and f) transferability (generalization). Algebraic Thinking lies at the core of Mathematics and serves as an integral component of the broader construct Mathematical Thinking. Our approach builds upon a review of related literature, which establishes a research-informed classification of Algebraic Thinking skills and competencies. This classification structure was used to derive an initial set of codes that enable us to annotate the curricula of six European countries (Finland, Hungary, Lithuania, Spain, Sweden, and Türkiye). Following this comprehensive analysis and classification of curricula a final coding structure was developed that captured the conceptual content of CT and Algebraic Thinking as evident in the analyzed curricula.

3 Analysis and Results

As one might expect the six national curricula differed both structurally and in terms of content and order of introduction of concepts. A curriculum typically consists of a series of topics, and within each topic area a list of its detailed learning material and outcomes is specified. The curricula forming the empirical data for our study therefore need to be consolidated during analysis. Appendix 1 contains the number of detailed learning statements in each curriculum ordered by country. One reason for the richness of the Hungarian curriculum is that it contains two kinds of details. One is the preparation for the knowledge, and the other is the real learning outcome. Some topic details are divided into 2-3 parts in a country, while it is in only one row in the others. These differences were reduced during the steps of consolidation. Duplication, associated with cognitive

progression in key topics was an important aspect of the analysis, since these sequences need to be incorporated into the CT&MathABLE learning pathways.

After eliminating duplicates, the Hungarian curriculum was selected as a reference point, since it is the most detailed. Each row of the other curricula was assigned to the corresponding topic in the Hungarian curriculum, if it existed, or a new topic (and perhaps code) was created where needed. The most important outcome of the current study is the identification of topic areas and concentrations within the curricula of our sample of six countries. The countries we studied exhibit a strong correlation, with commonality between curricula of over 47% between any four countries and over 80% with another three. Each country has its own national focus, with greater coverage of certain topics compared to the other countries studied. Spain places emphasis on problem-solving and pattern recognition; Finland prioritizes equations and operations; Hungary focuses on comparison, sorting, and equations; Lithuania emphasizes measurements and problem-solving; Sweden places considerable focus on problem-solving and ratios; and for Türkiye highlights measurements and equations as particularly significant. Further details are provided in Appendix 1.

4 Conclusion

We have comprehensively analysed mathematics education literature to formulate a precise definition of the cognitive development areas encompassed by Algebraic Thinking. This definition, along with a higher-order CT definition, have been applied to coding of statements within the mathematics curricula of six European nations. We observe substantial similarity, as well as intriguing differences, in terms of the frequency with which certain codes are referenced. Content analysis reinforces the assertion that the core of the curricula shares significant similarity, with a congruence of nearly 50%. A general learning pathway is also evident, commencing with the exploration of simple objects through classification and categorization, as learners delve into their properties and relationships. Drawing from their experiences, learners identify patterns and acquire the ability to generalize. Their mathematical vocabulary expands and matures, culminating in the integration of advanced concepts and definitions, which, combined with their arithmetical skills, equip them for problem-solving. This overarching learning trajectory encompasses the key elements of Algebraic Thinking and will serve as a foundation for the subsequent phase of the project, facilitating the creation of tasks necessary to support individual learning paths within each trajectory. This personalisation will be achieved by leveraging learning analytics, aiding each learner in defining a unique trajectory along the path, based on their previous task performance and demonstrated accomplishments.

References

1. Bocconi, S., Chiocciariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M., Jasutė, E., Malagoli, C., Masiulionytė-Dagienė,

- V., Stupurienė, G., Giannoutsou, N., Inamorato dos Santos, A., Punie, Y., Cachia, R.: Reviewing computational thinking in compulsory education: state of play and practices from computing education. Joint Research Centre (European Commission), Publications Office of the European Union, LU (2022), <https://data.europa.eu/doi/10.2760/126955>
- Bråting, K., Kilhamn, C.: Exploring the intersection of algebraic and computational thinking. *Mathematical Thinking and Learning* **23**(2), 170–185 (Apr 2021). <https://doi.org/10.1080/10986065.2020.1779012>
 - Denning, P.J., Tedre, M.: Computational Thinking: A Disciplinary Perspective. *Informatics in Education* **20**(3), 361–390 (Jul 2021). <https://doi.org/10.15388/infedu.2021.21>, <https://infedu.vu.lt/journal/INFEDU/article/701>, publisher: Vilnius University Institute of Data Science and Digital Technologies

Appendix 1

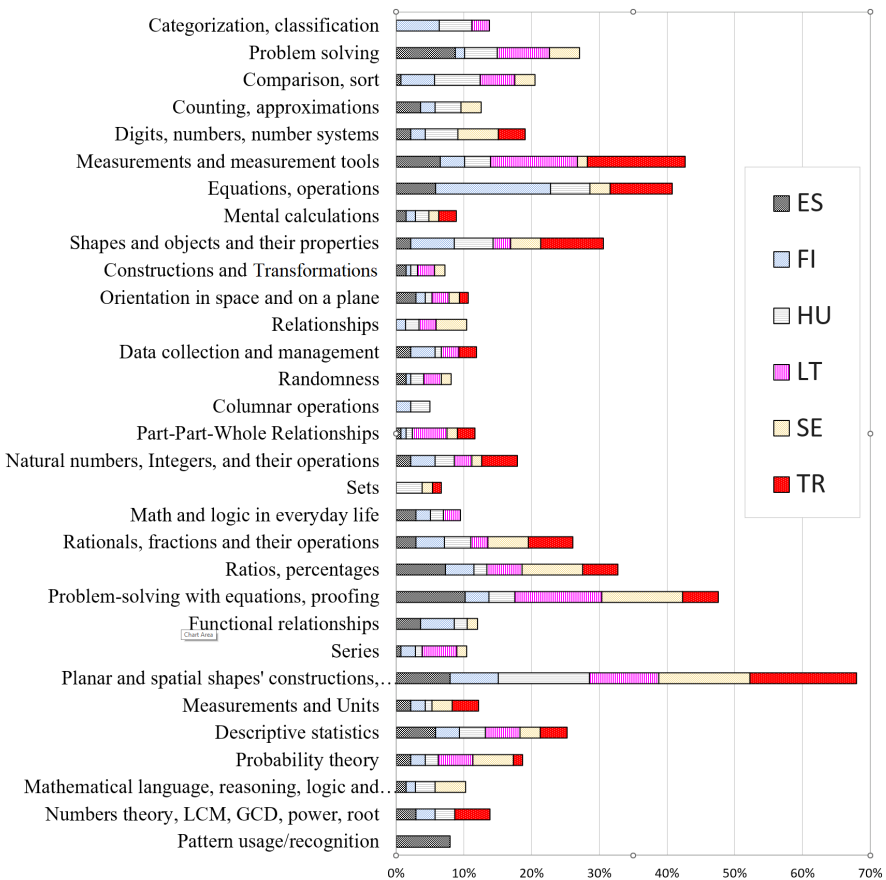


Fig. 1. Curricula comparison

A Constructionist Approach for Transitioning to College-Level Mathematics Education

András Margitay-Becht^{1,2}

¹ Saint Mary's College of California, 1928 Saint Mary's Road, Moraga, CA 94575, USA

² Eötvös Lóránd University, Pázmány Péter sétány 1/C, 1117 Budapest, Hungary
abecht@inf.elte.hu

Abstract. Math phobia is ever present in our society: a significant portion of students find math stressful. This paper describes a constructionist approach to providing support to freshman computer science students studying mathematics. Utilizing the high-level of existing programming knowledge of incoming computer science students at Eötvös Lóránd University, a workshop series is proposed to be offered where the students will learn how they can construct the introductory concepts and techniques of linear algebra using their existing programming knowledge, thereby helping them achieve a deeper, more meaningful understanding of the field.

Keywords: Math education at university level, Math phobia, Constructionism, Constructivism

1 Introduction

Math phobia is the fear of doing math. It can greatly impact the entire life of an individual: “Highly math-anxious individuals are characterized by a strong tendency to avoid math, which ultimately undercuts their math competence and forecloses important career paths. [...] Although the causes of math anxiety are undetermined, some teaching styles are implicated as risk factors.”[1]

According to the 2012 PISA study[2], about 30% of students have anxiety doing Mathematics. These results describe the *strong* anxiety definition: students actively feel bad while doing math. There is a much wider group of students who simply dislike math without the feeling of helplessness, and thus end up avoiding engaging with it similarly. The result of these self-limiting of Math are adults who are struggling to perform the simplest of calculations, and often react by talking themselves into thinking mathematics is not even important “in the real world”.

Math anxiety has been analyzed for a long time. Richardson and Suinn [3] developed a rating scale of Math anxiety back in 1972. Betz[4] analyzed the prevalence of math anxiety in 1978, finding that females and students who had poorer preparation in High School were more math anxious. Four decades later, Khasawneh et. al. [5] performed a scoping review of relevant literature, summarizing findings of the field.

They found that Math anxiety is prevalent among many disciplines and across multiple countries.

What is the impact of math anxiety and poorer math performance on university studies? Abraham et. al. [6] reviewed literature focusing on college readiness in math. They found that more students were in remedial Math classes than English or Reading. Atuahene and Russel [7] were also investigating American college students' math readiness, and found similarly dire results. DiMartino and Gregorio [8] looking at Italian Math majors found a crisis in transitioning to university level Math studies. Those students who managed to change their attitudes towards mathematics succeeded in transitioning, but those who did not ended up failing. They called on the importance of supporting this transition, focusing on students' feelings and attitudes during the first year. Geisler and Rolka [8] found, that students who consider math to be a series of rules and formulas tend to struggle when transitioning to university. On the other hand, students who view Math as a dynamic field with relevant applications usually succeed.

This paper is a proposed solution to explicitly expressed math phobia by a specific group of students at a specific university. Using a constructionist approach the students can take control of their own understanding of mathematics, with the aim of it reducing the fear and anxiety they experience when learning the material.

2 Constructionist math education for computer science students

In his seminal work *Mindstorms*[9], Seymour Papert highlighted the opportunity presented by the advent of personal computers. He hoped that the universal appeal of the computer, "the Proteus of machines" will ensure that all children will be enthralled by it. He then hoped to translate this excitement into a general excited view of mathematics and sciences. Noss and Hoyle utilized this concept to teaching mathematics, first by using the LOGO programming language [10], then further developing it into the Microworlds context [11]. The constructionist approach to teaching mathematics integrated the highly approachable Scratch programming language to form Scratch-Math[12]. The idea behind the current project is to re-capture the success of Scratch-Math by re-creating a similar experience for an older group of students.

2.1 Constructing Linear algebra

At Eötvös Lóránd University we are providing the students with an introductory mathematics course during their first semester. The course is made up of roughly half a semester's worth of summary of high school mathematics, but the second half introduces a new area: linear algebra. This proposal is about supporting linear algebra education through a constructionist approach.

The motivation to provide constructionist support comes from interaction with some students in my mentor class. At our university all incoming freshmen are assigned to mentor courses where a student from a higher grade and a mentor instructor meets them weekly to discuss difficulties and solve problems. Most students start the

program having had some experience with programming; in my groups 38 out of 40 said they are comfortable writing code, but the most feared subject they had was mathematics. Yet when we re-contextualized the mathematics topics through the lens of programming (for example, by explaining how formal logic and conditional statements are rather similar, and they can translate one to the other), they reported higher understanding, less fear and greater motivation to work on the math. This proposal is an attempt to create a systemic support system for the study of linear algebra accessible to all interested students instead of the ad-hoc delivery as part of another course, delivering reduced anxiety to a broader group of students.

2.2 Proposed delivery method

The intervention would be delivered as a series of optional workshops provided in a synchronous online manner in the evenings of the fall of 2023. The workshops being optional and an expansion of the course material means that the experiment is low stakes for the students: no student will be worse off by trying out the experience, and can freely discontinue participation. If, on the other hand, this approach would be provided *instead of* the traditional class material, such options would not be possible. Based on student feedback, the online modality and evening delivery ensures that most students can participate in their preferred mode of studying: at home, with their own equipment. This also means that students from different lecture groups can join, possibly addressing a broader audience.

The workshops would be advertised through the mentor classes to reach all incoming freshmen. The first sessions would take place a few weeks before the linear algebra material would start and would focus on familiarizing the students with the Python programming environment we will be using, and representations of numbers, vectors and matrices using the Pandas framework. The workshops then would follow the material weekly, first quickly summarizing the current topic, then leading the students through some programming exercises where they would implement the algorithms discussed during their math class in the programming environment used.

Please note: the student solutions are not expected to be optimal, or even necessarily always correct. Indeed, the programming environment contain solutions to all of the problems the students will be dealing with. The point is, that by re-implementing simple mathematical algorithms on vectors (for example vector addition, subtraction, calculating lengths of vectors, angles between them, dot products, cross products) and then matrices (inverting a matrix, finding the eigenvalues, etc.), the students are constructing functional algorithms – and also constructing a deep understanding of how those mathematical concepts and algorithms work.

The expected results are deeper understanding of linear algebra, a more instinctive understanding of the tools, methods and algorithms, and as a positive side effect, a bit of practice of applied computational thinking.

3 Conclusion

Providing a series of constructionist workshops in mathematics for computer science students at Eötvös Lóránd University has the potential to not only increase their understanding of mathematics and provide them with additional programming practice, but also help them overcome any existing math fear of math. This work continues the work started at Saint Mary's College of California teaching students without any programming background how to move past their fear of quantitative subjects, specifically programming, and become more self-assured, confident and have higher self-efficacy[13]. The author hopes that this approach translated over to mathematics can reverse already developed math phobia, and potentially re-ignite interest not only in mathematics, but also in the more mathematically-based areas of computer science.

References

1. Ashcraft, M.H.: Math Anxiety: Personal, Educational, and Cognitive Consequences. *Curr Dir Psychol Sci.* 11, 181–185 (2002). <https://doi.org/10.1111/1467-8721.00196>.
2. OECD: Key findings - PISA 2012, <https://www.oecd.org/pisa/keyfindings/pisa-2012-results.htm>, last accessed 2023/01/31.
3. Richardson, F.C., Suinn, R.M.: The Mathematics Anxiety Rating Scale: Psychometric data. *Journal of Counseling Psychology.* 19, 551–554 (1972). <https://doi.org/10.1037/h0033456>.
4. Betz, N.E.: Prevalence, distribution, and correlates of math anxiety in college students. *Journal of Counseling Psychology.* 25, 441–448 (1978). <https://doi.org/10.1037/0022-0167.25.5.441>.
5. Khasawneh, E., Gosling, C., Williams, B.: What impact does maths anxiety have on university students? *BMC Psychology.* 9, 37 (2021). <https://doi.org/10.1186/s40359-021-00537-2>.
6. Abraham, R.A., Slate, J.R., Saxon, D.P., Barnes, W.: College-Readiness in Math: A Conceptual Analysis of the Literature. *Research and Teaching in Developmental Education.* 30, 4–34 (2014).
7. Atuahene, F., Russell, T.A.: Mathematics Readiness of First-Year University Students. *Journal of Developmental Education.* 39, 12 (2016).
8. Di Martino, P., Gregorio, F.: The Mathematical Crisis in Secondary–Tertiary Transition. *Int J of Sci and Math Educ.* 17, 825–843 (2019). <https://doi.org/10.1007/s10763-018-9894-y>.
9. Papert, S.: *Mindstorms: Children, computers, and powerful ideas.* (1980).
10. Hoyles, C., Noss, R. eds: *Learning mathematics and Logo.* MIT Press, Cambridge, Mass (1992).
11. Noss, R., Hoyles, C.: Constructionism and Microworlds. In: Duval, E., Sharples, M., and Sutherland, R. (eds.) *Technology Enhanced Learning: Research Themes.* pp. 29–35. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-02600-8_3.
12. Noss, R., Hoyles, C., Saunders, P., Clark-Wilson, A., Benton, L., Kalas, I.: *Making Constructionism Work at Scale: The Story Of Scratchmaths.* Presented at the October 1 (2020). <https://doi.org/10.7551/mitpress/12091.003.0007>.
13. Margitay-Becht, A., Das, U.: Enhancing student learning through hidden motivational learning outcomes. In: Enomoto, K., Wagner, R., and Nygaard, C. (eds.) *Enhancing student learning outcomes in higher education.* Libri Publishing Ltd. (2023).

Enhancing Teacher Education Through STEAM Integration in Informatics*

Anita Juškevičienė^[0000–0003–1772–5537]

Vilnius University, Akademijos str. 4, Vilnius 08412, Lithuania
anita.juskeviciene@mif.vu.lt

Abstract. In the realm of modern education, the fusion of Science, Technology, Engineering, Arts, and Mathematics (STEAM) principles with informatics has emerged as a transformative approach to teacher education. This poster presentation delves into the innovative intersection of STEAM education and informatics, elucidating its profound impact on the professional development of educators.

As the educational landscape evolves in response to technological advancements, this poster encourages institutions and educators to embrace the synergy of STEAM education and informatics, fostering a cadre of proficient, innovative, and adaptable teachers prepared to lead their students into an era of boundless possibilities.

Keywords: Informatics · STEAM education · 3C4Life · teacher education.

1 Introduction

1.1 Motivation

Teacher shortages have become a pressing issue in Europe, prompting the initiation of an EU-funded endeavor with the primary objective of enhancing the appeal of the teaching profession. collaborative initiative, known as 3C4life [1] underscores the necessity for educators to modernize their pedagogical abilities, competencies, embrace digital methodologies, and exhibit self-sufficiency. 3C4life consortium takes on a critical challenge that hinders optimal conditions for STEM education throughout the continent: the insufficiency of STEM education systems in adequately supporting teachers to thrive in their roles.

The culmination of 3C4life project's efforts has resulted in the creation of an online platform, "teach4life," tailored to cater to aspiring STEM student teachers, emerging teaching professionals, and seasoned in-service STEM educators. The platform serves as a means to bolster the appeal of the STEM teaching profession and enable comprehensive growth.

* Supported by the ERASMUS+ grant program of the European Union under grant no. 626 139-EPP.I-2020-2-DE-EPPKA3-PI-POLICY, project "Perspectives for Lifelong STEM Teaching – Career Guidance, Collaborative Practice and Competence Development".

In order to enhance the allure of the teaching profession, several imperative prerequisites have been identified:

- The cultivation of a positive perception of the teaching vocation is imperative, both among educators themselves and within the broader society.
- Embracing the teaching profession as an ongoing journey of development, characterized by the integration of innovative teaching methodologies, is crucial.
- Encouraging and establishing collaborative practices and communities for professional learning is a pivotal step.
- Teachers must be provided with consistent support from the outset of their careers and throughout their professional trajectories, facilitating continuous professional advancement.

Project's measure represents a cutting-edge digital platform for STEM teachers, marked by its inventive approach to professional progression. The initiative is structured around three key viewpoints:

- Career: It encompasses both vertical and horizontal trajectories, offering comprehensive career guidance.
- Cooperation: Fosters collaborative practices among pertinent stakeholders within the national educational sphere. This primarily involves pre- and in-service STEM educators, higher education institutions engaged in training and research, providers of professional development, policymakers in education, and practitioners in schools.
- Competence: Focuses on the continual development of contemporary teaching and leadership competencies, ensuring educators remain updated and equipped with relevant skills.

The competence area provides activity ideas for implementing STEM activities in the classroom using innovative pedagogy:

- The use of inquiry-based learning allows learners to pose research questions, formulate hypotheses and test them through research.
- The principle of authentic context allows for the inclusion of research topics that are relevant to learners' interests in STEM learning activities.
- The social aspects of science encourage learners to engage in dialogue, discussion and debate. Their nature is often controversial. Learners have to form opinions and make decisions involving scientific, moral, ethical or social issues.

This study explores the dynamic integration of STEAM elements within teacher education, specifically focusing on the realm of informatics. By infusing pedagogical practices with creative and interdisciplinary thinking, teachers are equipped with a diverse toolkit to engage and inspire the next generation of informatics enthusiasts.

Poster highlights the positive influence of STEAM-infused informatics education on teachers' confidence, competency, and enthusiasm for teaching complex

technological concepts. By fostering an environment of innovation, adaptability, and cross-disciplinary collaboration, this approach paves the way for educators to nurture the analytical, computational, and creative thinking abilities of their students.

The poster also underscores the role of ongoing professional development and mentorship in sustaining the momentum of STEAM-integrated informatics education. It explores the transformative power of continuous learning, networking, and collaborative platforms, which further enrich the teaching and learning experience. Through captivating visuals and informative content, this poster not only underscores the significance of STEAM education in teacher training for informatics but also advocates for a paradigm shift in educational approaches to nurture a new generation of teachers equipped to navigate the ever-evolving digital landscape.

1.2 What to be presented at ISSEP 2023

The poster showcases diverse approaches to incorporating STEAM concepts into teacher training, i.e., project-based learning and collaborative problem-solving, designed to empower educators with the skills needed to foster a holistic understanding of informatics. In the teach4life platform competence area (see Fig. 1) you can find a large collection of tasks for the classroom that follow innovative pedagogical concepts, sorted by subject and methodology. For example, in an

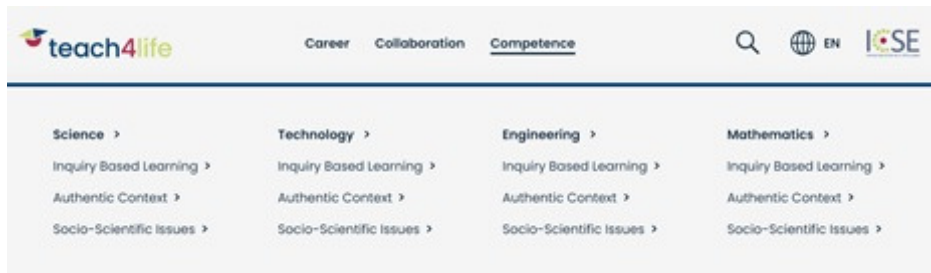


Fig. 1. Teach4life platform competence area.

engineering subject Authentic Context is presented the lesson “How to build a (pneumatic) pontoon bridge to evacuate civilians?” Engineering Design Process is an analytic and creative problem-solving process that engages a person in opportunities to make something physical and/or digital that matters. Project based learning can be implemented by using design thinking approach that also integrates computational thinking practices [2]. Design thinking serves as an educational framework due to its inherent attributes that foster the cultivation of specific skills essential for a productive learning approach. These skills lay the foundation for an enriching learning experience, encompassing factors such as

the drive for exploration, receptiveness to novel concepts, imaginative thinking, and various metacognitive proficiencies [3]. Design thinking has the potential to expand the scope of STEAM instruction. Moreover, it offers educators a structured avenue to foster enhanced creativity and interdisciplinary engagement, not only as a foundational element in their own pedagogical approach but also as an integral facet of students' immersive STEAM encounters [4]. The first activity was implemented based on this topic, however we decided to develop smart car security system by using Arduino controller kits (see Fig. 2, left).

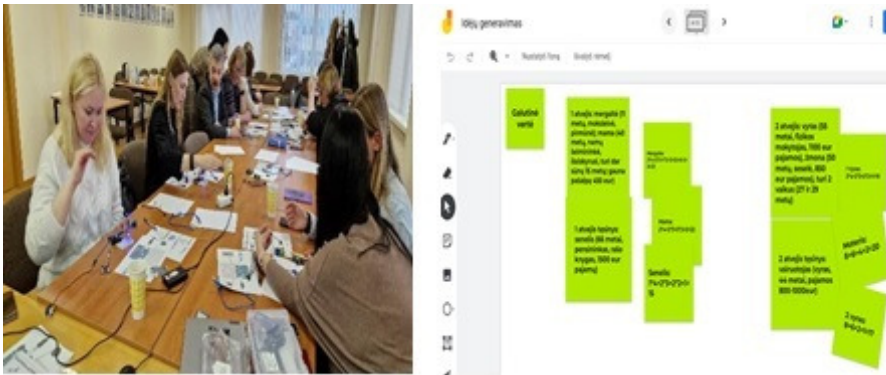


Fig. 2. Design thinking implementation activity (left); Jamboard environment (right).

Collaborative problem solving - idea generation is one of the stages of design thinking. There are various methods and tools for collaboration. To stimulate teachers' creativity and digital literacy, we have chosen a very timely topic - the morality of autonomous machines - and an easily accessible and usable online tool - Jamboard. The second activity was implemented based on Engineering area Socio-Scientific Issues in STEM Education topic "Should self-driving cars have a moral conscience?". Using Jamboard, we asked teachers to identify the factors that they think might influence the life-saving decisions of autonomous machines. We then asked them to rank them in order of importance and give numerical values. In the final stage, we presented several possible scenarios and mathematically calculated whose life should be saved first (see Fig. 2, right).

Acknowledgment. This work was supported by the ERASMUS+ grant program of the European Union under grant no. 626 139-EPP-I-2020-2-DE-EPPKA3-PI-POLICY, project "Perspectives for Lifelong STEM Teaching – Career Guidance, Collaborative Practice and Competence Development".

References

1. 3C4life project Homepage, <https://icse.eu/international-projects/3c4life/>. Last accessed 11 Aug 2023
2. Juškevičienė, A., Pears, A., Jevsikova, T., and Stupurienė, G.: Computational Thinking Design Application for STEAM Education. In *Data Science in Applications*. Cham: Springer International Publishing pp. 1–26. (2023) <https://doi.org/https://doi.org/10.1007/978-3-031-24453-7>
3. Scheer, A., Noweski, C., and Meinel, C.: Transforming constructivist learning into action: Design thinking in education. *Design and Technology Education: An International Journal* **17**(3),(2012)
4. Henriksen, D.: Creating STEAM with design thinking: Beyond STEM and arts integration. *The STEAM Journal*, 11 **3**(1), 11 (2017)

GeNIUS: Conditions for Successfully Teaching Computer Science Infused Natural Science Classes in Schools

Elena Yanakieva¹[0000–0002–2900–7252], Annette Bieniusa¹[0000–0002–1654–6118],
Christoph Thyssen¹, Thomas Becka³, Julia Albicker²[0000–0002–1757–9050],
Niklas Westermann², Barbara Pampel², and Johannes Huwer²

¹ University of Kaiserslautern-Landau, Kaiserslautern, Germany

{elena.yanakieva, annette.bieniusa, christoph.thyssen}@rptu.de

² University of Konstanz, Konstanz, Germany {barbara.pampel, julia.albicker,
johannes.huwer, niklas.westermann}@uni-konstanz.de

³ Eduard-Spranger-Gymnasium, Landau, Germany

Abstract. Algorithmic and computational thinking are usually seen as solely computer science skills. However, we believe that algorithms are commonly utilized in other subjects, such as conducting experiments in natural science settings. This poster presents our work as part of the project "GeNIUS", which aims to develop scenarios for computer-science-infused natural science lessons in schools and derive the necessary conditions for successfully conducting such lessons. So far two such scenarios have been conceptualized and tested in German schools. Conditions such as reliable technical infrastructure, teachers' and students' prior experience with computer science basics have proven to be crucial for successful lessons.

Keywords: STEM · Computational Thinking · Algorithmic Thinking · Multi-disciplinary Teaching.

1 Introduction

Our ever-evolving digital society necessitates that schools adjust and provide future generations with the required skills to thrive. The Standing Conference of the Ministers of Education and Cultural Affairs in Germany (KMK)⁴, responsible for education and schooling, has brought out a strategy called "Education in the Digital World" in 2016 [2], which aims to develop digital skills across schools. They have identified various computer science (CS) competencies anchored in established subjects that have not been sufficiently addressed. E.g., competence area "5. problem solving and acting", especially "5.5 recognizing and formulating algorithms" has been identified to be an integral part of the natural sciences, including biology, chemistry, and physics, as part of "designing and carrying

⁴ <https://www.kmk.org/kmk/information-in-english.html>

out experiments”, which is a competence requirement in their curriculum plan. Thus, CS-infused lessons, compatible with natural science lessons, are necessary.

We identified that the competencies proposed by KMK in its strategy fall under computational thinking (CT). CT describes the process of formulating a problem to executing a solution to the problem [6, 1]. Integration of CT and algorithmic concepts into lessons for conceptualizing problems and operationalizing their solutions show overlaps with scientific work [4, 3]:

- Understanding diagrams that describe or represent real-world problems;
- Planning tasks by systematically arranging the necessary processing steps;
- Using real data to examine critically and, if necessary, revise solutions;
- Break down complex processes into smaller parts;
- Create flow charts to represent different parts of a process.

With this poster, we aim to present our project GeNIUS, and its first results. GeNIUS aims to identify the necessary conditions to successfully offer scientific informatics education (SIE) as part of experimental natural science lessons while simultaneously providing lesson scenarios that can be easily adopted.

2 Project Description

As part of GeNIUS, lesson scenarios will be conceptualized in cooperation with teachers, according to the principle of participatory action, and conducted in schools, both such that enforce mandatory CS lessons and such that do not. The results will be evaluated concerning the conditions necessary to successfully integrate CT skills into the curriculum, specifically in natural science classes. These conditions may concern the conceptual parts but also the essential infrastructure.

Analogous to digitalized scientific research, algorithmic procedures for obtaining measurements in experiments can be individually adapted and actively experienced by students using simple programming environments and sensors. Through automation, time previously used for manual measurements can be used for other purposes. This makes it possible to implement didactic concepts that were once not feasible, which in turn requires didactic evaluation. Conditions for the success of SIE will be derived from evaluating such SIE scenarios. In addition, it will be investigated whether:

- students’ subject-related competencies improve as a result of SIE,
- how students’ and teachers’ perceptions of respective subjects change and
- whether they grow subject-related interest and self-efficiency expectations.

The evaluation findings will be publicly available in the form of best practice examples. They will prepare science teachers in Germany for meaningful SIE in further training courses. The following section provides an insight into the design and evaluation of the first SIE scenarios.

3 First Results

So far, we have developed two lesson scenarios, with focus on students in German schools. These can be used as part of the curriculum of the combined subject "natural science" in grade 6 or, if the teacher sees fit, in any of the separate subjects - biology or physics in the later grades.

The first scenario focuses on insulation and energy efficiency in housing, and the second one tests how light affects plant growth. The students are expected to have already learned about the topics in the respective subject. The scenarios focus on providing a better understanding of the topic through hands-on experimentation. In both scenarios, the first phase is the experiment conceptualization. The teacher leads a discussion over the topic, and the students are encouraged to think about a possible experiment to answer the posed questions. A possible analogous experiment is discussed, which leads to the idea of potential automation of the process, as part of which an algorithm is formulated.

The next phase provides the CS basics of block-based programming and utilizing microcontrollers to acquire sensory data. The microcontroller used in both experiments is Calliope Mini [5], as it is the most commonly available in German schools. This phase can be omitted if the students are familiar with the basics.

In the next phases students work in groups of two. In the first scenario, each group receives a cardboard box, which serves the purpose of the house, and each group insulates their house with materials of their choice. Next, they implement the previously designed algorithm as a program for the Calliope Mini to measure the temperature. In the second experiment, two groups work on one plant. One group measures light and distance, while the second one measures moisture and temperature. Light serves as the control parameter; moisture and temperature remain constant. Three different settings are prepared: 100%, 50%, and 25% light transmission. The insulation experiment lasts approximately 30 minutes, while the plant growth one continues for seven days. In the end, a lesson is dedicated to discussing the observed results.

We conducted these scenarios in schools and outlined some critical conditions necessary for successful lessons.

Technical infrastructure Many schools in Germany lack the necessary infrastructure to carry out long-term experiments such as the plant growth one. The school needs a reliable Internet connection to reliably save the data and display it in real-time for the students to observe the plant growth from anywhere. Furthermore, most schools we tested these scenarios in demand having accounts or certificates to establish an Internet connection. Calliope Mini supports only connection via simple SSID and password. Furthermore, some applications are disabled on the school devices, which can require specific workarounds.

Teachers' prior experience We are working with highly motivated teachers that acknowledge the necessity of digitalizing natural science lessons. However, motivation and high engagement are not enough when unforeseeable situations occur. Our experience shows that teachers who have prior experience with microcontrollers and feel more secure in teaching CS concepts can resolve tech-

nical difficulties among the students quicker. Therefore, as part of GeNIUS, we are planning training sessions for teachers, in which CS skills will be developed, by using microcontrollers, focusing on Calliope Mini. These will help use them meaningfully as part of the school lessons.

Students' prior experience Students showed improved enthusiasm when using microcontrollers in lessons that are not commonly related to CS. We observed differences in handling technical difficulties between students already familiar with handling Calliope Mini and the ones who did not have prior experience. The latter was easily discouraged if something did not work out as intended - e.g., the connection did not succeed.

Despite the above-mentioned limitations, the tendency exists for students to feel more motivated to find out answers through digitally-guided hands-on experimentation. Furthermore, all teachers have stayed as highly motivated to conduct further scenarios with us in the upcoming school years. In addition, more teachers are showing interest to be part of GeNIUS.

4 Summary and Future Work

We briefly presented our project GeNIUS, which focuses on determining conditions for successfully integrating computer science concepts into natural science curriculum in schools.

In the future, we plan to enhance the two presented scenarios and conceptualize further ones for the separate subjects of biology, chemistry, and physics. We plan to evaluate both CS and subject-related competencies that the students acquire as part of these lessons. Also, further evaluation of the importance of the students' prior CS experience and preconceptions is intended.

Acknowledgments

This research was funded by the Federal Ministry of Education and Research (project "GeNIUS" grant number 16MF1011A and 16MF1011B).

References

1. Aho, A.V.: Computation and computational thinking. *The computer journal* **55**(7), 832–835 (2012)
2. DER KULTUSMINISTER, D.L., DEUTSCHLAND, I.D.B.: Strategie der kulturministerkonferenz "bildung in der digitalen welt" (2017)
3. Drieling, K.: Der experimentelle Algorithmus: Das Beispiel Bodenversalzung. na (2006)
4. Eickelmann, B., Vahrenhold, J., Labusch, A.: Der Kompetenzbereich "Computational Thinking". Erste Ergebnisse des Zusatzmoduls für Deutschland im internationalen Vergleich (2019)
5. Reese, K., Wolf, V.: Calliope mini. *LOG IN: Vol. 38, No. 1* (2017)
6. Wing, J.M.: Computational thinking. *Communications of the ACM* **49**(3), 33–35 (2006)

From Wooden Blocks to Whimsical Robots: The “Programmieren spielend entdecken” Series to Nurture our Future Innovators

Fatmir Racipi, Stephanie Eugster, and Mathias Kirf

PHSG St. Gallen University of Teacher Education

Abstract. This poster presents the educational initiative of the extracurricular learning centre Smartfeld in St. Gallen, Switzerland. It focuses on the Programmieren spielend entdecken (Discovering Programming through Play) series, which is designed to provide a motivating introduction to programming for students of all ability levels in primary and secondary schools in the canton of St. Gallen. It consists of four different workshops tailored to different school levels. The PSE workshops aim to improve IT skills and various competencies, including social skills, while providing grade-specific programming experiences. Smartfeld, a unique interdisciplinary venture, brings together regional higher education institutions and the start-up environment. Located in the vibrant Swiss Innovation Park Ost, Smartfeld's mission is to foster creativity and future skills, promote STEAM education, and prepare students and educators for the digital age in an inspiring learning environment.

This poster does not have empirical results, but teacher feedback and workshop attendance data support its success, particularly among primary school educators. The workshops use a visual programming language to facilitate understanding and focus on core programming concepts. The inclusion of robots and microcontrollers increases engagement and motivation levels remain consistently high. The Smartfeld initiative has delivered over 400 workshops to over 7000 students. The use of visual programming languages has proved effective in facilitating problem solving and stimulating interest in programming, highlighting the importance of hands-on, experiential learning in the digital age.

Keywords: robot, programming, introduction, primary education, secondary education, creativity, playful, microcontroller, extracurricular, learning lab

1 Extracurricular learning venue “Smartfeld”

This poster describes an example of practice in the offer of the education laboratory “Smartfeld” in St. Gallen, Switzerland. It presents the series “Programmieren spielend entdecken” PSE (engl.: discover programming through play), an offer for all levels of primary and secondary school. The overall objective is to provide motivating programming introductions for pupils with diverse performance levels across schools within the canton of St. Gallen. These workshops aim to bolster IT proficiency and a range of competencies, including social skills. Classes are invited to partake in workshops tailored to their grade level, hosted at the extracurricular learning center, Smartfeld.

Smartfeld is an interdisciplinary initiative that stand as a unique entity in Switzerland, consisting of key stakeholders such as the Switzerland Innovation Park Ost, Swiss Federal Laboratories for Materials Science and Technology (EMPA), the St. Gallen Centre of Vocational Education and Training (GBS), the University of Applied Sciences of Eastern Switzerland (OST), the University of Teacher Education St. Gallen (PHSG) and the University of St. Gallen (UniSG). The overarching mission revolves around nurturing creativity and future skills while preparing pupils and educators for the demands of the digital era. Nestled within the dynamic Switzerland Innovation-park Ost, replete with burgeoning start-ups, Smartfeld combines technology and creativity, thereby fostering a deliberate emphasis on STEAM subjects and the creation of inspirational learning and experimental ecosystems.

This poster presents a practical demonstration of an innovative and playful approach to programming introduction within the school environment. All findings and conclusions presented herein are grounded in teacher feedback, practical classroom experiences, and an overall analysis of workshop participation data since the inception of the program.

Feedback analysis underscores the high acclaim received by active educators. It serves as a captivating and motivating initiation into the world of programming for pupils while instilling confidence in teachers' abilities to integrate programming into their curricula. There exists a substantial demand for such programs, particularly among primary-level educators. The intensive format of the workshops expedites efficient grade-appropriate introductions, fostering a compelling start and enabling swift progression. By introducing vital concepts within this learning ramp, pupils gain a foundation that can be further reinforced within their schools. Consequently, more complex educational goals can be pursued with a foundation already established. Moreover, the authentic context of the start-up hub co-located within the Smartfeld facility fosters direct and credible implementation, emphasizing the relevance and applicability of these skills in the professional world. This localized experience provides pupils with an understanding that such opportunities are not limited to Silicon Valley but are also manifest in their region.

2 Programmieren spielend entdecken – Workshops

The PSE series represents a structured introduction to programming designed for schools within the region. Comprising four half-day workshops tailored to various grade levels, ranging from the first primary school class to the final secondary school class, this initiative seeks to nurture pupils' programming skills over the course of their educational journey. Each workshop integrates age-appropriate robots or micro-controllers, fostering a hands-on, progressively challenging learning experience.

1st & 2nd primary class: Cubetto

Commencing with the utilization of the non-computer-based learning robot "Cubetto," this introductory stage engages pupils in the fundamental concepts of programming. Wooden blocks are employed to construct simple sequences in playful tasks.

3rd & 4th primary class: Thymio

This phase bridges the gap between robots and computers, initiating pupils into the world of visual programming languages. Individual sequences are constructed using various programming blocks, empowering pupils to execute tasks like illuminating an LED upon button press. Early sensor integration, such as the ultrasonic sensor show the combination of input and output through different pre-programmed modules.

5th & 6th primary class: mBot

Advancing to more sophisticated programming paradigms, pupils delve into concepts like loops and iterations to program self-guided robots. The visual programming language evolves with the incorporation of more intricate blocks, stimulating cognitive growth and problem-solving skills to program an obstacle-dodging robot.

1st - 3rd secondary class: micro:bit

The apex of the PSE series integrates microcontrollers, enabling pupils to interface with multiple sensors and actuators concurrently. This workshop promotes a deeper understanding of device control and encourages creative problem-solving.

The overarching objective is to provide pupils with a tangible programming experience, fostering an immediate connection between their actions and technical devices. This approach enhances pupils' engagement, curiosity, and persistence in pursuing diverse programming objectives.

Throughout these workshops, a visual programming language is exclusively employed. This pedagogical choice underscores the efficacy of block-based programming, as it simplifies program comprehension and enables a clear focus on core programming concepts. By minimizing text-based errors, pupils can more effectively engage with the structural aspects of programming, thus facilitating a more comprehensive understanding of programming principles.

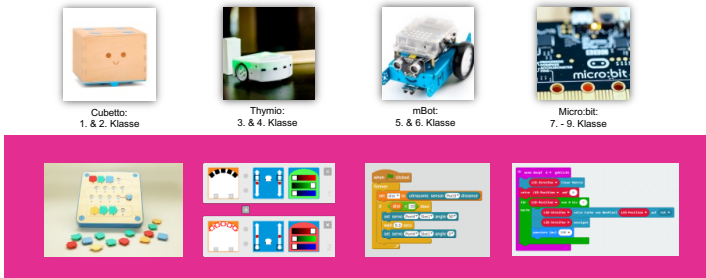


Fig. 1. An overview of the four different workshops of the “Programmieren spielend entdecken” series with their corresponding robots or micro-controller and their programming language.

3 Conclusion

The Smartfeld has already conducted around 400 workshops engaging over 7000 pupils as part of the PSE series. The PSE series underscores the favourable reception of the four workshops within the PSE series, as evidenced by sustained interest from regional educators visiting the Smartfeld regularly. Emphasis on visual programming languages has yielded positive outcomes, guiding pupils effectively toward problem-solving.

The decision to focus primarily on visual programming languages has proven to be the right one, as we have seen that it leads the pupils to the solution of the problems. With the block-based programming language, it is easier to focus on the concepts and not have to deal with the common errors of text-based programming languages.

The incorporation of robots and microcontrollers proves to be successful as it enhances the pupil’s enjoyment for programming and facilitating their initiation into the field. Consistently high motivation levels among participants stem from direct feedback afforded by interactions with robots and microcontrollers. Notably, these conclusions are based on initial feedback and insights garnered during workshop implementations, necessitating further investigation into motivation as a function of block-based languages or haptic device usage.

The figures show that the Smartfeld is being actively used. It is particularly well used by teachers at the primary level. It is posited that this preference may be attributed to the generalist training of primary school teachers as opposed to their specialized secondary-level counterparts. Within the primary level, workshops tailored for 5th and 6th-grade classes witness the highest attendance rates, partially attributable to the integration of "media and computer science" into standard curriculum at this educational tier.

Finding Patterns in Productive Failure Steps? An Explorative Case Study in a Teaching Learning Lab for Computer Science

Frauke Ritter¹[0000-0001-5744-9456] and Nadine
Schlomske-Bodenstein¹[0000-0003-0835-165X]

¹ Department for CS, University of Education, Karlsruhe, Germany
² {frauke.ritter,nadine.schlomske-bodenstein}@ph-karlsruhe.de
<https://www.ph-karlsruhe.de>

Abstract. Teaching *algorithmic thinking* is a core task of education in the 21st century. The *productive failure* approach is not yet widely used in computer science, but promising insights in better understanding learning processes in the process of acquiring *algorithmic thinking*. A calliope workshop is designed and implemented to identify $N = 13$ learner's patterns of *productive failure* steps in students in an exploratory qualitative case study. *Productive failure* patterns will then serve in a subsequent study for training student teachers in formative assessment. The results of this exploratory case study provide preliminary indications of how learning and teaching workshops on *algorithmic thinking* can be designed.

Keywords: Algorithmic Thinking · productive failure · Computational Thinking.

1 Introduction

In order to become an empowered citizen in our rapidly digitizing world, it is of great importance to teach students digital literacy according to, and thus especially *computational thinking* (CT) ([1]). For this to succeed, pedagogical approaches are needed that teach students CT, especially *algorithmic thinking* (AT) as a part of CT ([13]), for us the definition of AT steps according to [11] is crucial. An approach that has already been successfully tested in mathematics is *productive failure* (PF) [3], which belongs to the category PS-I (first problem solving then instruction) according to [5]. The theory states that students learn the content better when they first problem solve (PS) and then receive instruction (I). As an example of a PS-I approach, the approach of [3] and [6] includes four interdependent steps: (a) *activating and differentiating prior knowledge related to the target concepts*, (b) *attending to critical conceptual features of the target concepts*, (c) *explaining and elaborating these features*, and (d) *organizing and merging the critical conceptual features into the target concepts*. In the field of CS education, the approach is not very common yet, only [12] has presented it so far. Therefore, this study has the approach, in a first research step, to

analyze the steps of students who are taught AT using a developed workshop designed according to the principles of PF, and thus possibly identify typical PF patterns for CS education. This data will then be used to train student teachers in our CS Teaching and Learning Lab (TLL) to recognize PF and thus better train formative assessment of students. Overall, this is an exciting new area of research in CS education, which is presented here as a first step using the exploratory study conducted. Our research questions are: **RQ1: What kind of productive failures can be identified during the process of acquiring algorithmic thinking?** **RQ2: To what extent can patterns of productive failures be identified?**

2 Methods

The research question is operationalized through an exploratory single case study [14] that follows a qualitative approach, collecting and analyzing qualitative, triangulated data. Students’ programmed codes were captured at several points during the collaborative group work phases as students worked on the problem-based tasks. We are analyzing the qualitative through qualitative content analysis [7] and developed the following coding scheme to better understand the *productive failures* that were made and to better capture how pre-service teachers identified and categorized them.

Setting and Participants The sample is N=13 high school students, ages 13-14. They participated in a 90-minute workshop. The content was to program a counter step using block-based programming and the calliope. The students’ prior knowledge was programming with Scratch (sequences, loops, conditions, variables). We conducted the Calliope workshop taking into account research-based practices ([9]) to ensure that the workshop was implemented as designed. We considered [10] *treatment fidelity* categories.

Material: Workshop Design We developed a workshop (fig. 1) following [11]’s AT steps and PF [3, 4]’s design principles. Students solved a problem using Calliopes and block-based language. The workshop consisted of 3 phases: Problem Solving, Instruction, and Final Problem Solving (Fig. 1). In the Problem Solving phase, students work independently and individually, but have the opportunity to elaborate and explain at any time. In the instructional phase, different approaches, including failed ones, are presented and compared, and the problem is finally clarified with all the necessary content so that

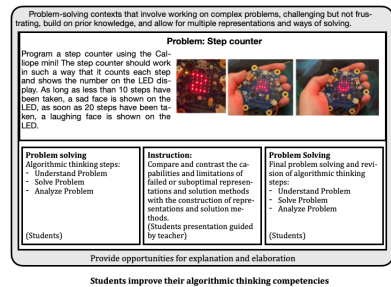


Fig. 1. Workshop design according to the principles of *productive failure* according to [3], [4] and the *algorithmic thinking* steps according to [11]

each student can create his or her individual solution in the final phase. The problem is challenging but not frustrating.

Instruments and Analysis The students' steps were videotaped on the computer to identify and qualitatively assess the steps in the learning process, especially the steps where PF occurs. In addition, the entire intervention was videotaped so that the students' presentations could be qualitatively assessed. Students worked through the AT steps using a paper-pencil worksheet and the block-based programming language makeBlock. Both the worksheets and the programs created were recorded. The collected qualitative data will be analyzed using a reductive qualitative content analysis according to the interpretive paradigm of [7], using a developed category system based on the categories of [8] adopted by [2] to identify PF patterns.

3 Results and Summary

In our initial analysis of the data, we analyzed the first phase of the workshop, the PS step before instruction, to identify PF. For this purpose, we evaluated the individual video recordings in combination with the created programs. The results show that the most frequently assigned category was Problem Solving (77 out of a total of 221). Orientation (55) and Criteria Development (40). Problem Analysis (21) and Problem Evaluation (27) were less frequent and Problem Critique appeared only once. Additionally we have categorized a total of 15 PFs (RQ1), 7 of which were in the solution development phase (logic errors, flow control errors, number representation errors), 3 in the criteria development phase, and 4 in the orientation phase (e.g., incorrectly selected physical elements or LED display errors). Due to the small number of PFs found, no valid patterns can be identified at this time (RQ2).

This case study is limited in several ways that need to be addressed in future work. First, it included only a small sample, and second, it did not examine the level of AT of the learners at the individual level. Of course, these two aspects could be the reason why only a few PFs could be found. Therefore, in a subsequent study, care should be taken to make the tasks even more open and/or to select a more heterogeneous, larger learning group. The problem solving process in the AT steps can be seen well in the categories, although Problem Analysis and Problem Evaluation are somewhat underrepresented, so that the basic approach in the AT steps according to [11] in combination with PF should be pursued further. In the next step, we will collect more student data and further analyze the data. In addition, we will study several workshops using the principle of PF. This data will be the basis for training students in our CS TLL to recognize PF and thus better train students' formative assessment. In particular, the relationship to misconceptions needs to be explored, as [15] or semantic errors in learning text-based languages are also found in PF steps of block-based

programming. Overall, there are interesting connections to different CS research approaches to be explored.

References

1. Juškevičiene, A., Dagiene, V.: Computational thinking relationship with digital competence. *Informatics in Education* **17**(2), 265–284 (2018)
2. Kapur, M.: Productive failure. *Cognition and Instruction* **26**(3), 379–424 (2008)
3. Kapur, M.: Productive failure in learning the concept of variance. *Instructional Science* **40**(4), 651–672 (2012)
4. Kapur, M.: Examining Productive Failure, Productive Success, Unproductive Failure, and Unproductive Success in Learning. *Educational Psychologist* **51**(2) (2016)
5. Loibl, K., Roll, I., Rummel, N.: Towards a Theory of When and How Problem Solving Followed by Instruction Supports Learning. *Educational Psychology Review* **29**(4), 693–715 (2017)
6. Loibl, K., Rummel, N.: The impact of guidance during problem-solving prior to instruction on students' inventions and learning outcomes. *Instructional Science* **42**(3), 305–326 (2014)
7. Mayring, P., Fenzl, T.: *Handbuch Methoden der empirischen Sozialforschung*. Springer Fachmedien Wiesbaden (2019)
8. Poole, M.S., Holmes, M.E.: Decision Development in Computer-Assisted Group Decision Making. *Human Communication Research* **22**(1) (1995)
9. Sanetti, L.M., Cook, B.G., Cook, L.: Treatment Fidelity: What It Is and Why It Matters. *Learning Disabilities Research and Practice* **36**(1) (2021)
10. Smith, S.W., Daunic, A.P., Taylor, G.G.: Treatment fidelity in applied educational research: Expanding the adoption and application of measures to ensure evidence-based practice. *Education and Treatment of Children* **30**(4), 121–134 (2007)
11. Standl, B.: Solving everyday challenges in a computational way of thinking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10696 LNCS**(November), 180–191 (2017)
12. Steinhorst, P.: Investigating Productive Failure in Computer Science. In: *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*. vol. 2 (2022)
13. Wing, J.M.: Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **366**(1881), 3717–3725 (2008)
14. Yin, R.K.: *Case Study research and applications: design and methods*. Sage Publications, Inc., Los Angeles, 6th editio edn. (2018)
15. Žanko, Ž., Mladenović, M., Krpan, D.: Analysis of school students' misconceptions about basic programming concepts. *Journal of Computer Assisted Learning* **38**(3), 719–730 (2022)

Gender Differences in Problem Solving Observed in Logo Novices

Jacqueline Staub¹ and Angélica Herrera Loyo²

¹ Fachbereich IV, University Trier, Behringstrasse 1, 54296 Trier, Germany
staub@uni-trier.de

² ABZ, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland
angelica.herrera@inf.ethz.ch

Abstract. Women are still underrepresented in particular fields, such as science, technology, engineering and math (STEM). In computer science, a particular aspect that is being investigated is the different approaches boys and girls take to programming tasks. Existing hypotheses state that boys are more inclined to unconventional solutions, while girls prefer to take a more structured approach. This study investigates the validity of this hypothesis, particularly in the context of novice programming in Logo using Turtle Graphics. Through a comparative analysis of the programs written by 42 children, a Chi-Square-Test shows no significant difference between boys and girls ($\chi^2 = 0.5046$ in terms of creativity and structurability); the Mann-Whitney U Test however shows a small gender bias in favor of males on the creativity variable ($p=0.06$). Our next steps consist in developing a more accurate measurement for structurability, given that we reached an interrater reliability score of only 0.237 for structuredness but 0.805 for creativity.

1 Introduction

As a central research topic, gender studies try to understand the inequalities, practices and interactions between men and women. Statistics show that in many areas, women's participation is underrepresented, in particular in STEM (science, technology, engineering and mathematics) [12,8,1]. Some studies show gender differences in interest [10], confidence [11], and attitudes [13] in computer science. One particular difference lies in the way boys and girls proceed when they are programming. Several aspects point towards social factors that contribute to the difference: Korkmaz and Altun [9] found that the low participation of women in computer studies has a background of insecurity about the skills required for programming. Hanström [7] found that, in order to apply the method of trial and error, confidence in one's own abilities is essential.

Other results point towards cognitive differences that affect the low representation of women in computer science. Programming is an activity that requires several cognitive skills. Usually, emphasis is placed on: (1) logical thinking, (2) abstraction, and (3) creativity. Gender studies suggest that men outperform women in logic and abstraction skills as well as in solving unconventional

problem statements [2,3]. In contrast, when it comes to learning languages or solving conventional problem statements, women have been shown to outperform men [3,4,6].

In this work, we focus on the two indicators *structurability* and *creativity* and perform a qualitative study on whether novice Logo programmers show gender differences in terms of these two indicators.

2 Study

The primary objective of this work is to analyze whether there are gender-specific differences in terms of *creativity* and *structurability*. In order to answer this question, the authors planned an intervention with two groups of eleven and twelve year old students (N=49, 26 male, 23 female). Prior to the intervention, both classes were introduced to the basics of Logo programming (i.e, sequence, loops, procedures) for one school year with two hours every two weeks. The subsequent intervention lasted for one week, with 12 lessons per group.

After a short repetition on the previously-covered concepts, the students focused on the concepts, both framed within the topic of Logo animation. We worked with a textbook that proposes a sequence of predefined task [5]. After each day, the children's solutions were downloaded and stored for subsequent analysis. 7 children (4 boys and 3 girls) were excluded from the study since no data was saved. In order to rate the student's solutions on the spectrum of creativity and structurability, we defined the following point-rating:

Creativity:

- (2) The child uses concepts/approaches that have not been discussed in class. They develop custom tasks.
- (1) The child uses some commands/approaches that have not been discussed in class. They develop some programs without predefined tasks.
- (0) The child follows the approaches discussed in class. They rarely develop their own tasks.
- (-1) The child follows the approaches discussed in class. They follow the given tasks.
- (-2) The child exactly follows the approaches as per instruction. They strictly follow the given tasks.

Structurability:

- (2) The child consistently and purposefully uses loops, procedures and parameters to create short and concise solutions.
- (1) The child uses loops, procedures and parameters mostly purposefully.
- (0) Sometimes the child uses loops, procedures and parameters.
- (-1) Oftentimes the child does not use loops, procedures and parameters or they appear disorganized.
- (-2) The child does not use loops, procedures and parameter. If used, the concepts are not purposeful, hardly understandable and appear disorganized.

After conducting the intervention, the anonymized data was rated according to the above point-rating system in regard of creativity and structurability. Both

authors established an interrater agreement with regular discussions after every ten students with the aim to ensure consistency in the rating scheme. First, the data was anonymized in terms of gender information. Discrepancies existed predominantly on the structurability variable where only a fair agreement was reached ($\kappa = 0.237$). In contrast, the interrater reliability on the dimension of creativity yielded an almost perfect agreement ($\kappa = 0.805$).

3 Preliminary Results

The results of the interrater agreement is visualized below (see Figure 1). The figure on the left presents the results for all male participants; the figure in the middle shows the results for all female participants and the figure on the right shows all results combined. Each dot represents a group of children with a specific creativity-structurability score. The x-axis corresponds to the creativity score while the y-axis corresponds to the structurability score.

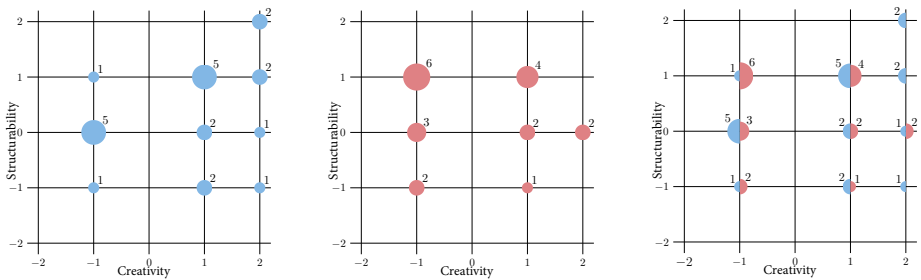


Fig. 1: The diagram illustrates the results for the 42 participants (left male participants, middle female participants, right all participants).

The score roughly clusters children in four categories (quadrants in the diagram): participants whose solutions are rated (i) creative and structured, (ii) creative and not structured, (iii) not creative and structured, (iv) not creative and not structured. Categories (i) and (ii) are reflected in the top and bottom right quadrants whereas categories (iii) and (iv) are reflected in the top and bottom left quadrants respectively. Appendix A presents some selected examples for children’s programs that fall into these four categories.

The statistical chi-tests on a bias on gender were all insignificant ($\chi^2 = 0.5045$ for structuredness and $\chi^2 = 0.5045$ for creativity) on a level of alpha=5%. The one-sided Whitney-Mann U test however showed a p-value of 0.06 on the creativity variable, indicating that there is a slight gender bias in favor of male participants. Under the assumption that gender plays no role for creativity, only in 6/100 similar experiments, we would have found such a result or a more extreme one.

4 Future Work

We analyzed programs from 42 students with an average of 8 programs per student. In a second phase, we aim to gather and analyze data from more students and different schools to conduct a more quantitative analysis. Moreover, we aim to establish a more precise objective to measure structurability or separate the variable into smaller segments that can be measured more easily. Finally, we want to incorporate more variables (e.g., spatial orientation, abstraction, socio-cultural aspects) to get a better understanding of gender-impact.

References

1. Bernhard Ertl, Silke Luttenberger, and Manuela Paechter. The impact of gender stereotypes on the self-concept of female students in stem subjects with an underrepresentation of females. *Frontiers in psychology*, 8:703, 2017.
2. Ann Gallagher, Jutta Levin, and Cara Cahalan. Cognitive patterns of gender differences on mathematics admissions tests. *ETS Research Report Series*, 2002(2):i-30, 2002.
3. Ann M Gallagher. Sex differences in the performance of high-scoring examinees on the sat@m. *ETS Research Report Series*, 1990(2):i-16, 1990.
4. Ann M Gallagher. Sex differences in problem-solving strategies used by high-scoring examinees on the sat-m. *ETS Research Report Series*, 1992(1):i-35, 1992.
5. Heidi Gebauer, Juraj Hromkovič, Lucia Keller, Ivana Kosírová, Giovanni Serafini, and Björn Steffen. *Programmieren mit LOGO*. ABZ, Ausbildungs- und Beratungszentrum für Informatikunterricht, 2015.
6. Diane F Halpern. A cognitive-process taxonomy for sex differences in cognitive abilities. *Current directions in psychological science*, 13(4):135-139, 2004.
7. M.B. Hanström, Sverige. Kungl. Tekniska Högskolan. Jämställdhetsrådet, and Tekniska högskolan i Stockholm. *Studiemiljö och jämställdhet på Kungl. Tekniska Högskolan: en intervjustudie med kvinnliga teknologer på Kemiteknisk och Farkostteknisk linje : rapport från KTHs jämställdhetsråd*. Trita-FL. KTH, 1994.
8. Lizhi He, George Zhou, Geri Salinitri, and Lianrong Xu. Female underrepresentation in stem subjects: An exploratory study of female high school students in china. *EURASIA Journal of Mathematics, Science and Technology Education*, 16(1), 2020.
9. Ö Korkmaz and H Altun. Engineering and ceit student's attitude towards learning computer programming. *The Journal of Academic Social Science Studies International Journal of Social Science*, 6(2):1169-1185, 2013.
10. Elizabeth K Lawner, Diane M Quinn, Gabriel Camacho, Blair T Johnson, and Bradley Pan-Weisz. Ingroup role models and underrepresented students' performance and interest in stem: A meta-analysis of lab and field studies. *Social Psychology of Education*, 22:1169-1195, 2019.
11. David MacPhee, Samantha Farro, and Silvia Sara Canetto. Academic self-efficacy and performance of underrepresented stem majors: Gender, ethnic, and social class patterns. *Analyses of Social Issues and Public Policy*, 13(1):347-369, 2013.
12. Diana Starovoytova Madara and Sharon Cherotich. Female underrepresentation in undergraduate education: Case study in school of engineering. *Research on Humanities and Social Sciences*, 6(14):157-175, 2016.
13. Chao Xu and Renée E Lastrapes. Impact of stem sense of belonging on career interest: The role of stem attitudes. *Journal of Career Development*, 49(6):1215-1229, 2022.

A Appendix

Listing 1.1: structured and creative

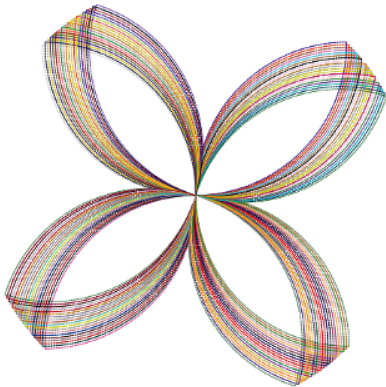
```

to cc :long :oban
  lt :oban
  repeat 90 [fd :long rt 1]
end

to petalo
  setpc random 16
  cc 3 0 rt 90 cc 3 0
end

to superflor
  repeat 5 [
    repeat 90/5 [
      petalo rt 1 petalo
    ]
    rt 72
  ]
end

```



a: structured and creative

Listing 1.2: not structured but creative

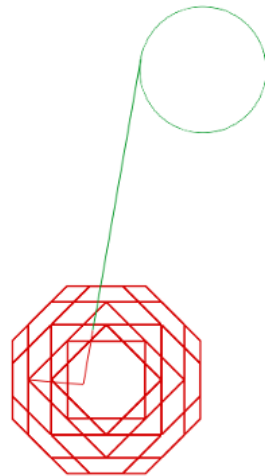
```

to jj
  repeat 8 [ fd 50 rt 45]
end

to hh
  jj repeat 3 [ fd 50 rt 45]
  fd 50 rt 90
end

to flor :g :h
  setpc 1 hh jj hh jj hh jj hh
  jj hh jj hh jj hh jj hh jj hh
  jj hh jj hh jj hh jj hh jj hh
  jj hh jj hh jj hh jj hh jj hh
  jj hh jj hh jj rt :g fd:h
  rt 180 rt :g rt 45 fd :h
  setpc 2 fd 200 fd :h
  repeat 360 [fd 1 rt 1 ]
end

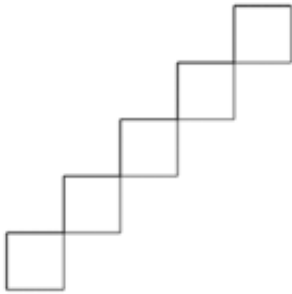
```



b: not structured but creative

Listing 1.3: structured but not creative

```
to stairs :gr :anz
  repeat :gr [
    quad :anz
    fd :anz rt 90
    fd :anz lt 90
  ]
end
```



a: structured but not creative

Listing 1.4: not structured not creative

```
to star8
  repeat 8 [
    fd 100 bk 100 rt 45
    fd 100 bk 100 lt 45
  ]
end
```



b: not structured and not creative

Teaching the Von-Neumann Model with a Simulator

Martin Weinert, Jan Hendrik Krone, and Johannes Fischer

Technische Universität Dortmund, 44227 Dortmund, Germany
{martin.weinert,hendrik.krone,johannes.fischer}@cs.tu-dortmund.de

Abstract. An important goal of computer science education is the demystification of computing machinery. The Von-Neumann-model can help to achieve this goal, since it demonstrates how programs are executed.

Therefore we developed a simulator and an accompanying series of lessons on the Von-Neumann-model. Such teaching series are already being used in practice, but are often separated from related computer science topics. We connected our lessons to the concepts of sequential logic systems, (higher) programming languages and the IPO model.

Keywords: Von-Neumann-Model · IPO · Series of Lessons

1 Introduction

One of the core goals of computer science education is the demystification of computing machinery. The Von-Neumann-model contributes to this [7,6] by providing a simple, yet precise description of a computer, which allows students to understand how computers operate.

Because of the complexity of the topic, it is important to apply some simplifications through the teaching material. It is a common idea to achieve this by using simplified simulators. Such simulators are already in use. Examples are Johnny [1,2] and MOPS [4] for use in schools or Microsim for university education. The simulators were developed with several design principles in mind. MOPS, for example, used the IPO model (Input-Processing-Output) as an orientation [4]. Although the simulators are simplified and put emphasis on different aspects, they still have difficulties to effectively teach the Von-Neumann-cycle [3]. In order to address these difficulties and to be able to focus program execution as a teaching topic, we decided to develop and use a new simulator.

2 Simulator KUR2

An important design decision of the simulator is the arrangement of the components. In contrast to other simulators like Johnny, MOPS or Microsim, we consciously separated the input and output components and arranged them in a way that resembles the IPO model. The user enters programs in machine code,

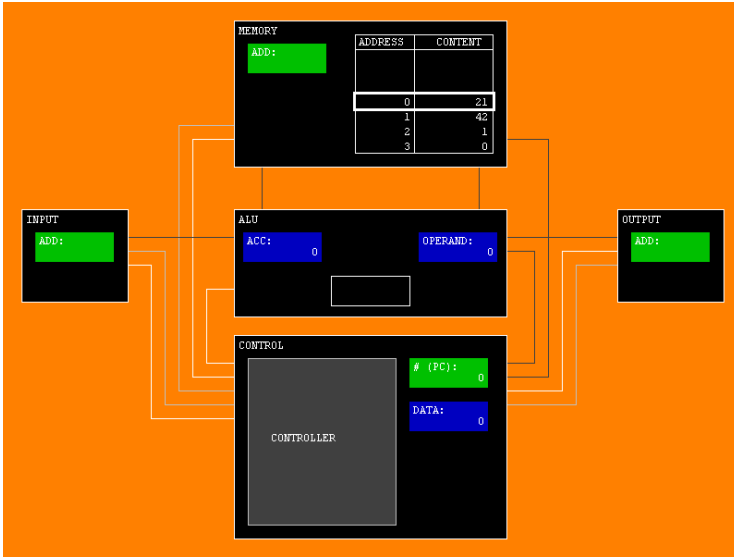


Fig. 1: Screenshot of the Simulator

which are then stored in the memory unit and executed afterwards. Due to the presentation of the internal registers, the flow of data can be followed.

We emphasized a very simplistic design, because we wanted to integrate the simulator into a story about John von Neumann. Therefore we did not want to include modern design elements into the graphical representation.

3 Lessons series and materials

In this section, we describe the lessons that we developed. They can be accessed on the authors' website [5] for further examination and usage.

3.1 Series of Lesson

The series of lessons is composed of six lessons:

Lesson 1: The students learn how the simulator is structured by puzzling together its pieces on a worksheet.

Lesson 2: The students interact with the simulator for the first time. They simulate first programs and learn how to write simple programs for basic calculations.

Lesson 3: The students see how the functions of the simulator can be implemented with sequential logic. They interact with a logic simulator, which shows how some of the functions of KUR2 can be performed with logic gates. They learn how sequential logic and the Von-Neumann-model are connected.

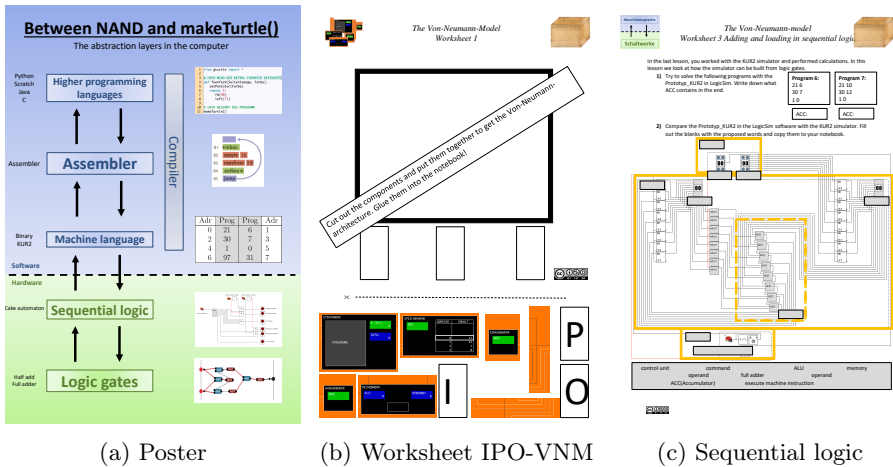


Fig. 2: Materials used in the lessons

Lesson 4: The students learn about jumps. First they formulate that programs are stored as a linear sequence of commands. Then they see that execution can deviate from this sequence through (conditional) jumps. At the end of the lesson the students write down these insights as rules of the von-Neumann-model.

Lesson 5: The students see that programs and data are stored in the same memory. They also learn how the memory is structured and addressed. These insights are again written down as rules of the model.

Lesson 6: Finally there is a last lesson where the students can choose from three activities. Those activities introduce connections to external (virtual) devices and show how text and pixel displays can be accessed or how a simple guess the number game can be played with the simulator. The activities are intended to teach that the machine is independent of the problem.

3.2 Supporting materials and key features

During the development of the series of lessons some aspects emerged as especially important. The first of those aspects came from the realization that the first contact with the simulator was quite overwhelming for the students. We tried to make it easier for the students by introducing the puzzle (see fig. 2b) in the first lesson, where they just put the simulator together and relate it to the IPO model.

Next we tried to make it easier to see the connections to related topics and build a coherent overall picture. To this end we created a poster (see fig. 2a) that shows the different layers of abstraction and their connections. This poster was presented in the computer lab to be visible at all times. These connections were also emphasized by adding the activity where the students would see the simulator implemented with logic gates (see fig. 2c).

Finally we tried to tie all lessons together with an engaging backstory. The idea is that someone has found a box containing John von Neumanns old belongings. Since the notebook that contains information on the Von-Neumann-model and the simulator KUR2 is partly destroyed and unreadable, the students have to reconstruct its contents through the lessons and worksheets.

4 Future work

Our observations during the lessons indicate that the connection of the Von-Neumann-model to underlying topics like sequential logic and the IPO model, and to emphasize how it fits into the broader context might be substantial for effective teaching. We will investigate how this affects the development of mental models of computing machinery in the future.

The aforementioned connection was mainly established with the poster that shows the connections between the Von-Neumann-model and other topics, as well as the simulator itself. We will look for additional ways to improve the connections.

All of the materials and the simulator are free to use for educational purposes. The materials and simulator can be accessed on the authors' website [5]. The simulator is implemented in Java and a JavaScript version is currently in development. The source code can be accessed via GitHub [8].

References

1. Dauscher, P.: Johnny - A Simulator of a Simple von-Neumann Computer (2012), <https://sourceforge.net/projects/johnnysimulator/>
2. Dauscher, P.: Aufbau und Funktionsweise eines Von-Neumann-Rechners - Ein möglicher Unterrichtsgang mit dem Open-Source-Simulator Johnny. LOG IN **33**(2) (2013)
3. Göbel, L., Hellmig, L.: Die Von-Neumann-Prinzipien erleben - Ein enaktiver Zugang zu einem abstrakten Thema. LOG IN **39**(1) (2019)
4. Haase, M.: MOPS Modellrechner mit PseudoAssembler. <http://www.viktorianer.de/info/mops.html> (2013)
5. Krone, J.H., Weinert, M.: Teaching materials for KUR2 (2023), <https://ls11-www.cs.tu-dortmund.de/staff/weinert/kur>
6. Lautebach, U.: Vom Gatter zum Compiler: Im Unterricht durch sieben Abstraktionsebenen. In: Gallenbacher, J. (ed.) Informatik allgemeinbildend begreifen. Gesellschaft für Informatik e.V., Bonn (2015)
7. Sorva, J.: Notional machines and introductory programming education. ACM Transactions on Computing Education **13**(2), 1–31 (jun 2013)
8. Weinert, M.: Code repositories for KUR2 (2019), <https://github.com/kur2>

An Approach to Introduce High-School Students to the P-vs-NP Question

Jisoo Song¹, Seoyeon Oh², Soyeon Jeong², and Seongbin Park²

¹ LoadIT, Seoul, Korea

² Korea University, Seoul, Korea

jisoo@loadit.co.kr, {oseo21320,jj2709,hyperspace}@korea.ac.kr

Abstract. In this poster, we are concerned with a way to introduce high school students to the P vs NP question. It is intended to be used in a one hour lecture of the high school credit system in the fall this year. Our approach is novel in that the question is explained from the perspective of imaginable worlds. While this may look abstract to high school students who do not have mathematical backgrounds, it certainly has an advantage in that students can have opportunities to stretch their imaginations to understand different kinds of worlds. Since our goal of giving a lecture about the P vs NP question is not to teach students technical details but to help them understand different perspectives on the hardness of computational problems, we believe that our approach will be sufficient enough to achieve the goal.

Keywords: P vs NP question · a possible world · logical consequence.

1 Introduction

In this poster, we describe an approach that can be used to introduce high school students to the P vs NP question. The approach is different from typical ways by which the question is introduced in that structural properties of different worlds in which the question is addressed are used. We believe that this approach will help high school students become interested in the question because it does not require mathematical backgrounds and it can possibly foster creative thinking.

2 A Proposed Approach

Many researchers believe that P and NP are not the same. A different way of interpreting this belief is that it looks unlikely that a world exists in which solving a problem and verifying the validity of a given solution are equally hard [1]. Now, this question can be explained from the perspective of different worlds as follows.

1. First, we start with a hypothetical world (World 1) in which the problem of deciding whether an arbitrary quantified boolean formula is true or not (TQBF problem) is solved by an oracle in one step [2].

- (a) To explain this world to students, it is necessary to point out that there are different kinds of resources from the perspective of algorithms. Examples include time, space, network bandwidth, randomness, and interaction. In addition, we can emphasize that we need to pay some cost in the form of any of these resources in order to compute something. In other words, computation is not free.
 - (b) In World 1, $P = NP$. This implies that all NP complete problems can be solvable efficiently. After consequences of the existence of an oracle for the quantified boolean formula problem are explained, we may raise the following question. Is this the world in which we live? Although the answer seems to be no, no definite answer is known yet and it may be possible to have an unconventional computer that serves as this oracle in the future.
2. Second, we introduce Algorithmica [3] to students. Like World 1, $P = NP$ in this world. However, it is unlikely that Algorithmica and World 1 are the same. The difference between these worlds lies in the existence of an oracle that solves TQBF in constant time or not. In other worlds, even though $P = NP$ in Algorithmica, we do not know whether such an oracle exists in this world. On the other hand, there are many structural similarities between World 1 and Algorithmica because all logical consequences of $P = NP$ are true in both worlds. For example, inductive learning becomes an easy problem in both worlds. In addition, the RSA algorithm is no longer safe in both worlds [3].
 3. Then, we can introduce a world in which P is not the same as NP . One such world is Heuristica [3] where hard instances of NP complete problems exist, but it is difficult to find such hard instances. At this stage, we can select several structural properties of the worlds mentioned and raise the question, "What are properties that look true in the world where we live?".

3 Conclusion and Future Directions

In this poster, we described an approach that can be used to introduce high school students to the P vs NP question. It is different from typical ways of introducing the question in that possible worlds in which the question is addressed are explained. We believe that our approach can serve as an eye opener for high school students because they never have had such a perspective on the hardness of computational problems. In other words, students may be adept at solving mathematical problems taught in school, or they may be aware of physical laws such as Newton's laws of motion which govern our world, but they do not know a possibility of combining a world and mathematical problems addressed in the world. Once students get interested in the subjects, we can possibly mention additional worlds such as Optiland, Obfustopia, and a quantum world [4, 5]. In addition, it will be interesting to point out that a world can be analyzed in terms of almost everywhere solvability [6].

Acknowledgements Seongbin Park is the corresponding author.

References

1. P, NP, and NP-Completeness: The Basics of Computational Complexity, Oded Goldreich, Cambridge University Press, 2010
2. The nature of computation, Cristopher Moore, Stephan Mertens, Oxford University Press, 2011
3. A personal view of average case complexity, Russell Impagliazzo, SCT '95: Proceedings of the 10th Annual Structure in Complexity Theory Conference (SCT'95)
4. Fifty Years of P vs. NP and the Possibility of the Impossible, Lance Fortnow, Communications of the ACM, Vol. 65, No. 1, 2022
5. Obfuscopia Built on Secret-Key Functional Encryption, Fuyuki Kitagawa, Ryo Nishimaki & Keisuke Tanaka, EUROCRYPT 2018, Lecture Notes in Computer Science, Vol. 10821
6. What one has to know when attacking P vs. NP, Juraj Hromkovič, Peter Rossmanith, Journal of Computer and System Sciences, Vol. 107, 2020

Promoting Artificial Intelligence and Data Literacy within Teacher Education

Valentina Dagienė¹, Martin Kandlhofer², Vaida Masiulionytė-Dagienė³,
Viktorija Olari⁴, and Ralf Romeike⁴

¹ Institute of Educational Science, Vilnius University, Lithuania
`valentina.dagiene@mif.vu.lt`

² Austrian Computer Society (OCG), Vienna, Austria
`martin.kandlhofer@ocg.at`

³ Institute of Data Science and Digital Technologies, Vilnius University, Lithuania
`vaida.masiulionyte-dagiene@mif.vu.lt`

⁴ Computing Education Research Group Freie Universität Berlin, Germany
`{viktoriya.olari,ralf.romeike}@fu-berlin.de`

Abstract. With the rising emphasis on integrating Artificial Intelligence (AI) and Data Literacy (DL) into school education within global and European educational frameworks, the demand for training teachers in DL and AI literacy has become pressing. Nonetheless, the current landscape lacks comprehensive professional development programs to address these needs. To address this gap and establish effective implementation of data and AI competencies in teacher training, we engaged in a collaborative effort with policymakers from Germany, Austria, and Lithuania. Our approach was rooted in action research, facilitating the development of a tailored teacher training program. We propose an initial approach that employs the data lifecycle to contemplate on DL skills pertinent to AI. The outcomes serve as a groundwork for constructing a comprehensive strategy for training K-12 educators of all disciplines in AI and DL. This work overviews the project “TrainDL - Teacher training for Data Literacy & Computer Science competences” objectives and implementation.

Keywords: Artificial Intelligence Education · AI Literacy · Data Literacy · Teacher Training · K-12 education

1 Introduction

Teacher training initiatives are actively underway in Austria, Germany, and Lithuania, with a specific focus on cultivating Data Literacy, Computer Science / Informatics competences. The central objective is to enhance the transferability of digital education skills. The project TrainDL (<https://train-dl.eu/en>) consortium’s immediate goal is to determine the optimal integration of Data Literacy (DL) and Artificial Intelligence (AI) competences into teacher training programs. This integration encompasses both university-level education and professional development for in-service teachers, particularly Computer Science (CS)

teachers and also those within the STEAM fields, and primary school teachers. Key stakeholders, including pre-service and in-service teachers, as well as policy-makers in the education domain are actively engaged. Through a participatory approach grounded in systems thinking and action research, the project is currently testing hypotheses, evaluating measures, and assessing their applicability over three distinct project cycles. This collaborative process is fostering ongoing knowledge exchange and establishing a metric to gauge the present status of DL and AI literacy within Computer Science education, ultimately leading to the establishment of a European teacher education monitoring system. Building upon these insights, the project will provide curriculum recommendations as well as a policy monitor for DL and AI, aimed at directly impacting relevant national and European stakeholders.

2 Data Literacy and Artificial Intelligence in School Education

The research landscape in AI has more than doubled since 2010, with certain areas like ethics experiencing a rapid surge in recent years [2]. In response to the rapid pace of technological advancements and resulting societal shifts, researchers, educators, and policymakers are increasingly acknowledging the importance of introducing students to concepts such as data collection, understanding, evaluation, and the critical application of AI technologies from an early stage [5]. Data Literacy can be defined as "*[...] the ability to collect, manage, evaluate, and apply data in a critical manner*" [4]. Efforts to integrate data and AI literacy into K-12 education have been ongoing for years, aiming to equip young learners with the skills needed to excel in a data-driven world [3]. However, the key to effectively integrate AI and Data Literacy (AI&DL) into school education lies in adequately preparing teachers as well as in the proper curriculum support. UNESCO's recent report highlighted that only 11 countries approved K-12 AI curricula by 2021 [6]. While international frameworks exist, such as OECD's AI policy observatory and the EU's DigComp/DigCompEdu, European countries vary in implementation and may have their own policies.

3 Development of Teacher Training: Methodology

To comprehend the scarcity of teacher training programs in AI&DL and to bridge this gap, we embraced an action research approach. Action research is iterative, involving cycles of planning, action, observation, feedback, and reflection. The project protocol contains three phases iterated three times throughout the project roadmap: The policy dialogue & building with a close integration of public authorities (A), the intervention with the field research and experimentation phase (B) and the evaluation on a micro (pedagogical), meso (organizational/structural) and macro (policy) level (C). The evaluation flows into the policy dialogue and thus guide the intervention of the next phase (see Figure 1). In this poster, we concentrate on the initial action research cycle on level B.

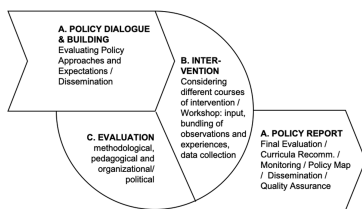


Fig. 1. Project lifecycle phases

Phase 1 - Grasping Practice: We studied the incorporation of AI&DL in school education and the availability of teacher training. Subsequently, we analyzed educational policies, engaging with policymakers and stakeholders in Germany, Austria, and Lithuania to discern factors relevant to teacher education in AI&DL. Phase 2 - Intentional Enhancements: Based on our findings, we devised a one-day AI&DL training program specifically tailored for secondary-level and primary-level. Phase 3 - Implementation and Observation: We executed the training program across Germany, Austria, and Lithuania, gauging its effectiveness through evaluation. Our approach remained reflective, fostering ongoing discussions within the research group and stakeholders.

When designing the teacher training program, we carefully considered the demands of policy requirements, insights gleaned from prior research in AI&DL within school education, and the TPACK model [1]. Our primary focus was on in-service teachers. In terms of content knowledge, which is the first facet of the TPACK model, we concentrated on core concepts such as rule-based AI and machine learning (ML) paradigms (including supervised, unsupervised, and reinforcement learning) and the essential data lifecycle. These concepts consistently emerge as pivotal themes across AI literacy and data literacy frameworks. Furthermore, we introduced educators to their national school curricula and AI&DL guidelines, informed by our policy research. Addressing pedagogical knowledge, the second aspect of the TPACK model, we structured the training around the didactic biplane method, a widely adopted approach in German-speaking regions for CS teacher training. Employing this technique, the facilitator assumes the role of a classroom teacher, while participating educators embody students, engaging with materials as they would in their own classrooms. Adhering to policy recommendations, our training incorporates unplugged resources and computer-based activities that teachers can seamlessly integrate into their teaching practices. Our selected computer-based activity for the data lifecycle employs *Orange3*, a user-friendly visual modeling tool that requires no programming skills, also *Google Teachable Machine*, an online tool which helps to better understand how AI based solutions can learn from data.

4 Conclusion

The TrainDL teacher training concept's evolution was shaped by close engagement with European policymakers, ensuring alignment with their perspectives. The formulated teacher training framework was subsequently put into action and quantitatively and qualitatively evaluated in Germany, Lithuania, and Austria. A one-day teacher training program had a notable positive impact on perceived and demonstrated AI competence. Across the three countries, quantitative findings indicated improved classroom integration of AI content. While DL enhancement varied, AI comprehension consistently improved. Teachers' viewpoints for incorporating AI&DL highlight a strong inclination towards integrating these topics into framework curricula. It is observed across all countries that formal curriculum anchoring is not a prerequisite for teaching these subjects. The prevailing consensus, also among the policymakers, is that AI&DL should form a part of teacher trainings. Additionally, there is a clear call to enhance and expand professional development opportunities in this domain. Our journey provides insights into multifaceted AI&DL teacher training, from stakeholder engagement to policy alignment, training effectiveness, and content preferences.

Acknowledgement

This work has been funded through the Erasmus+ KA3 Policy Experimentation project: "TrainDL - Teacher training for Data Literacy & Computer Science competences", 626145-EPP-1-2020-2-DE-EPPKA3-PI-POLICY.

References

1. Koehler, M., Mishra, P.: What is technological pedagogical content knowledge (TPACK)? *Contemporary issues in technology and teacher education* **9**(1), 60–70 (2009)
2. Lee, I., Zhang, H., Moore, K., Zhou, X., Perret, B., Cheng, Y., Zheng, R., Pu, G.: AI Book Club: An Innovative Professional Development Model for AI Education. In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. pp. 202–208 (2022)
3. Long, D., Magerko, B.: What is AI literacy? Competencies and design considerations. In: *Proceedings of the 2020 CHI conference on human factors in computing systems*. pp. 1–16 (2020)
4. Ridsdale, C., Rothwell, J., Smit, M., Ali-Hassan, H., Bliemel, M., Irvine, D., Kelley, D., Matwin, S., Wuetherick, B.: *Strategies and best practices for data literacy education: Knowledge synthesis report* (2015)
5. UNESCO: *Beijing Consensus on Artificial Intelligence and Education*. <https://unesdoc.unesco.org/ark:/48223/pf0000368303> (2021), access: 09/23
6. UNESCO: *K-12 AI curricula: a mapping of government-endorsed AI curricula*. <https://unesdoc.unesco.org/ark:/48223/pf0000380602> (2022), access: 09/23

Exploring Students' Preinstructional Mental Models of Machine Learning: Preliminary Findings

Erik Marx^{1,2}[0000-0002-5918-804X], Thimeo Leonhardt^{1,2}[0000-0003-4725-9776], Nadine Bergner³[0000-0003-3527-3204], and Clemens Witt¹[0009-0005-8160-4029]

¹ TUD Dresden University of Technology, Germany

² Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI)
Dresden/Leipzig, Germany

³ RWTH Aachen University, Germany

Abstract. This poster proposal provides insights into research on school students' conceptions of machine learning (ML) and highlights the effectiveness of interviews in exploring their mental models. In our study, we use semi-structured interviews to explore the mental models students develop prior to instruction on ML, outlining the advantages of this method for obtaining detailed insight. Eight interviews with German school students were conducted, revealing different perspectives. The preliminary findings indicate that some students imbue artificial intelligence (AI) with anthropomorphic qualities. Traditional concepts of computational thinking are also referenced, but often do not match the realities of ML. These findings contribute to computer science education by providing a nuanced understanding of school students' conceptions of AI and ML and highlighting the need for accurate education in the evolving landscape of AI and ML.

Keywords: Machine Learning · Mental Models · School Students' Conceptions · Preconceptions · Interview Study

1 Introduction

Considering the growing integration of artificial intelligence (AI) into students' daily lives, it becomes crucial for them to acquire a fundamental understanding of the functioning of this technology. Based on constructivist learning theories the assimilation of new knowledge occurs through the adaptation of existing knowledge structures [10, 2]. One concept that explains how individuals explain the world, solve problems, or form hypotheses is that of mental models [1, 2]. Mental models are cognitive representations of situations or domains that help in understanding, learning, reasoning, or predicting [1, 2]. In educational research, other terms are used that address students' preinstructional understanding such as preconceptions, alternative conceptions, or p-prims but with little consistency as to how these terms relate to each other [10]. We understand student conceptions and related terms as the observable phenomena of the use of internal mental

models. The study of mental models is necessary because they provide us with information about the learning process of students, for example, by explaining typical errors and thus enabling us to design better learning experiences [1].

Students' view on AI is influenced by a diverse array of sources. Media and in particular science fiction wield substantial influence over students' conceptualizations of AI, often leading to anthropomorphism of AI, thereby conflicting with accurate computational concepts [7]. Furthermore, in the field of computer science education (CSE) machine learning (ML) presents distinctive characteristics compared to conventional subjects within CSE. This contrast is elucidated by Tedre et al., who argue that ML introduces a paradigm shift, challenging established notions of computational thinking (CT). For instance, in the context of imperative programming, correctness predominantly revolves around syntactic accuracy or the generation of precise outputs. Conversely, within the domain of ML an element of probability is introduced, indicating the likelihood of correctness for each generated output [11].

The primary objective of this interview study is to delve into the preinstructional mental models that school students employ in their attempts to elucidate phenomena within ML. This investigation seeks to assess the potential implications of these mental models on the learning process and their potential conflicts with correct computational concepts of ML.

2 Related Works

Some works already exist that delve into school students' preconceptions about AI. Mertala et al. used a qualitative questionnaire to inquire middle-school students about their perceptions of how AI functions. The analysis primarily focuses on discerning how children conceptualize the type of technology AI represents and where AI finds application [9]. Kim et al. assessed middle school students' preconceptions about AI through video observations and learning artifacts, tracking their development across a summer camp [4]. Kreinsen & Schultz conducted interviews with students on their understanding of AI [5]. These endeavors share the commonality of encompassing a general assessment of school students' conceptions of AI, without concentrating on specific technologies like ML, which holds particular relevance for CSE research, given the fundamental differences between some concepts of ML and traditional CS concepts. Additionally, we identify a lack of utilization of interview-based data collection methods, despite their capacity to pose follow-up questions, thereby facilitating deeper understanding of students' mental models. With our investigation, we aim to narrow the focus onto the realm of ML, thereby exploring school students' preinstructional mental models concerning ML via an interview study.

3 Methodology

To assess students mental models, we opted for the traditional semi-structured interview, commonly used in mental model research [1]. A total of eight inter-

views with German school students were conducted with a duration between 43 and 69 minutes. Of the children interviewed, 6 were in eighth grade, 1 was in fifth grade, and 1 was in third grade. As Jones et al. found, interviewees' responses are more detailed and in-depth when they are directly exposed to the phenomena about which they are being questioned [3]. Thus, the students were introduced to two technologies to interact with during the interview: facial recognition on smartphones and ChatGPT. After each of these interactions, the participants were asked questions about how these technologies work. A semi-structured interview guide was created, outlining topics of interest derived from literature (such as correctness of AI [11]). Further detailed questions were formulated to facilitate deeper discussions. In the interview the interviewer also had flexibility to ask unlisted follow-up questions, aligning with the semi-structured interview approach.

For the analysis, interviews were transcribed from audio recordings and augmented with video footage to incorporate students' gestures and facial expressions, enhancing contextual understanding. Subsequently, the transcripts are to be analyzed using qualitative content analysis according to Kuckartz [6], taking into account Mayring & Fenzl's recommendations [8]. In accordance with [6], the analysis started with reading each interview and summarizing initial findings. The preliminary findings presented in section 4 are derived from this step. An inductive-deductive approach is planned post-initial text work to build the coding framework using the main categories from the interview guide. Open coding will refine the category system, followed by coding all interviews and analyzing thematic summaries.

4 Preliminary Findings and Implications

The initial analysis of the data demonstrates that the interviewed students provided a diverse range of explanations. Some students exhibited familiarity with ML concepts, such as neural networks, or possessed a basic comprehension of how facial recognition operates. In contrast, however, there were also more naive perspectives, like the notion that ChatGPT was simply some form of sophisticated search engine. Particularly when dealing with less familiar technical intricacies, students tend to anthropomorphize the technology, attributing human-like traits to it. For instance, one student characterizes AI as striving for self-improvement and perpetual learning. Another student draws a direct parallel between ChatGPT and humans, asserting that both make random decisions because humans are inherently "random" as well.

However, it's important to acknowledge that explanatory approaches rooted in traditional CT concepts also emerge. For instance, students are well aware of the pivotal role that data play and acknowledge its significance. However, a noteworthy observation permeating all the interviews, which has yet to be addressed in related work, is that most students assume data is stored and that AI makes decisions based on a comparison of input data with stored "training data"; this notion directly contradicts the actual functioning of ML systems. In

another instance, a student combines both anthropomorphic ideas and CT concepts. He speculates that ChatGPT was coded by developers with an initial set of handcrafted grammatical rules and vocabulary and subsequently independently scoured the Internet for new data to enhance its vocabulary.

The preliminary results presented herein bear relevance to research in CSE by offering an additional perspective on school students' conceptions of AI. In summary, it can be deduced that the conducted interview study proves effective as a methodology for assessing the various explanations put forth by students. The partially novel findings underscore the value of semi-structured interviews, as they allow to ask specific follow-up questions and thus to render students' mental models on the functionality of ML systems visible.

References

1. Gentner, D.: Mental Models, Psychology of. In: International Encyclopedia of the Social & Behavioral Sciences, pp. 9683–9687. Elsevier (2001). <https://doi.org/10.1016/B0-08-043076-7/01487-X>
2. Greca, I.M., Moreira, M.A.: Mental models, conceptual models, and modelling. *International Journal of Science Education* **22**(1), 1–11 (Jan 2000). <https://doi.org/10/btggz6>
3. Jones, N.A., Ross, H., Lynam, T., Perez, P.: Eliciting Mental Models: A Comparison of Interview Procedures in the Context of Natural Resource Management. *Ecology and Society* **19**(1) (2014)
4. Kim, K., Kwon, K., Ottenbreit-Leftwich, A., Bae, H., Glazewski, K.: Exploring middle school students' common naive conceptions of Artificial Intelligence concepts, and the evolution of these ideas. *Education and Information Technologies* (Jan 2023). <https://doi.org/10.1007/s10639-023-11600-3>
5. Kreinsen, M., Schulz, S.: Students' Conceptions of Artificial Intelligence. In: The 16th Workshop in Primary and Secondary Computing Education. pp. 1–2. ACM, Virtual Event Germany (Oct 2021). <https://doi.org/10/gqjsqq>
6. Kuckartz, U., Rädiker, S.: *Qualitative Content Analysis: Methods, Practice and Software*. SAGE Publications, Thousand Oaks, second edn. (2023)
7. Marx, E., Leonhardt, T., Bergner, N.: Brief Summary of Existing Research on Students' Conceptions of AI. In: Proceedings of the 17th Workshop in Primary and Secondary Computing Education. pp. 1–2. WiPSCE '22, Association for Computing Machinery, New York, NY, USA (Oct 2022). <https://doi.org/10/gq2p2n>
8. Mayring, P.: *Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution*. Klagenfurt (2014)
9. Mertala, P., Fagerlund, J., Calderon, O.: Finnish 5th and 6th grade students' pre-instructional conceptions of artificial intelligence (AI) and their implications for AI literacy education. *Computers and Education: Artificial Intelligence* **3**, 100095 (Jan 2022). <https://doi.org/10.1016/j.caeai.2022.100095>
10. Taber, K.S.: The Nature of Student Conceptions in Science. In: Taber, K.S., Akpan, B. (eds.) *Science Education*, pp. 119–131. New Directions in Mathematics and Science Education, SensePublishers, Rotterdam (2017). https://doi.org/10.1007/978-94-6300-749-8_9
11. Tedre, M., Denning, P., Toivonen, T.: CT 2.0. In: 21st Koli Calling International Conference on Computing Education Research. pp. 1–8. ACM, Joensuu Finland (Nov 2021). <https://doi.org/10/gnqv9f>

Teachers' Experience Regarding Digital Threats for Children and Teenagers

Julian Taupe^[0000–0002–9148–835X], Verena Knapp^[0009–0005–0982–0472], and
Andreas Bollin^[0000–0003–4031–5982]

Universität Klagenfurt, Universitätsstraße 65/67, 9020 Klagenfurt am Wörthersee,
Austria

Abstract. In a time embossed by increasing technology integration, digital security has become crucial, particularly for children and teenagers in primary and secondary school levels who extensively use digital media and devices on a daily basis. Austria's cybercrime incidents have almost been six-fold in the last ten years, highlighting the need for developing competencies in the field of digital security. This study focuses on teachers' experience in regards of threats pupils have to cope with. The study establishes a structured approach to address digital security concerns among children and teenagers. This informs teachers about potential threats and unveils possible competency gaps for pupils. An interesting finding is the ranking of threats from teachers' point of view based on their experience in reference to digital threats for children and teenagers.

Keywords: Digital Threats · Research in Informatics Education · Curriculum Gaps · Competencies for Children.

1 Introduction

Digital media and services are omnipresent in our society which has revolutionized the way we access information, interact, and communicate with others. These technologies have become a part of our daily routines and accompany many of us in many areas like education, profession or leisure. At the same time there is a significant increase in the frequency of individuals becoming involved into criminal cases within the digital area [1]. With increasing and early integration, however, potential threats to children and teenagers also become more relevant [2–6]. These threats range across various domains, such as inappropriate or illegal content, social environment and interactions, data and its protection, as well as potential addictive behavior. In order to determine the most significant threats, these need to be examined from different perspectives. This article focuses on taking a closer look at the perspective of teachers in regards of their personal experience with digital threats children and teenagers have to cope with.

2 Study regarding Teachers' Experience

In the course of this contribution, results of an empirical study with teachers in Austria are demonstrated. During the study, conducted between April and May 2023, teachers were asked about their experiences with regard to selected threats in the context of digital security among children and teenagers. In total 1.153 schools have been contacted to ask teachers for participation. Apart from the federal state of Salzburg, teachers from all over Austria took part in the study. The results reflect answers of 708 surveyed teachers of primary and secondary school levels. Among other parts, the questionnaire covered questions regarding threats in the digital realm. Based on their assessment, teachers were asked to rank a set of predefined threats within four groups based on the frequency with which young people are confronted with them. Furthermore, they were asked how they experienced children and teenagers being exposed to specific threats. In terms of experience, three options are distinguished. First-hand experience, which means that teachers have personally experienced or witnessed that children or teenagers have been confronted with digital threats. Indirect experience, meaning teachers at least know someone who experienced such cases. Finally, there is the possibility that teachers have not yet had such an experience or know a person who has.

3 Discussion

Existing publications already mention many threats and problem areas in the digital realm. These are already sufficiently documented, but they are scattered across several publications. What is missing, however, is a ranking of these threats in relation to the frequency with which children and teenagers are confronted with them. This ranking can be used, for example, to prioritize defining competencies. For this reason, this study was carried out to receive a sequence of threats in the digital context from teachers' point of view for the first time in Austria. Regarding the number of threats collected, they have been divided into four groups for easier ranking. In order to find suitable groups, threats were analyzed to determine any similarities. During the questionnaire teachers have been asked about their personal experience in regards to selected threats in reference to children and teenagers. The weighting of each threat is composed of teachers' first-hand and indirect experience. Table 1 illustrates the ranking of the threats according to teachers' first-hand experience.

Some potential sources of bias concerning the internal validity of the collected data have been considered. For instance, teachers may tend to pick their responses in accordance with societal expectations, possibly leading to an over-rated or underrated result for specific threat rating. Furthermore, teachers may have difficulties recalling certain occurrences or events, especially those that took place in the past. This could lead to distortions in responses respective to personal experiences with diverse threats. Furthermore, the distribution of teachers across gender conforms to the population, which strengthens representativeness.

The circumstance that the education directorate of the state of Salzburg did not permit teachers in Salzburg to participate in the study is not expected to be a substantial source of error.

Moreover, in regards of the external validity some considerations have been considered. First of all, the sample of teachers was randomly selected, encompassing schools as well as individual teachers, with no targeted selection. Also, the data indicates a corresponding distribution, particularly in regards of teachers' gender, aligning with the general population. Hence, the sample is representative and applicable to the target audience. It is worth noting that voluntary participation could introduce a bias, favoring teachers with specific opinion or experience. To minimize systematic errors, the online questionnaire enforced active response for all questions by not providing default values. Teachers were not forced to participate at a specific time and no rewards have been promised for taking part in the study.

Table 1. Threats ranked according to teachers' first-hand experience (rounded to two decimal places)

Threat	First-Hand Experience	Indirect Experience
Information Overload	47.32%	16.81%
Social Media Addiction	43.93%	30.37%
Gaming Addiction	40.54%	34.32%
Fake Information	38.42%	23.02%
Cyber Mobbing	36.16%	38.84%
Inappropriate Content	30.93%	32.20%
Computer/Data Damage	18.50%	20.20%
Illegal Content	17.37%	26.84%
Reputation	16.95%	29.38%
Copyright	15.82%	15.40%
Cyber Stalking	15.82%	30.65%
Data Breach	15.68%	19.63%
Violence/Crime Incitement	11.44%	23.31%
Injury Trivialization	10.17%	19.77%
Contact Strangers	8.62%	16.10%
Personal Privacy	8.19%	18.64%
Shopping Addiction	7.91%	19.21%
Purchase Illegal Substances	7.91%	15.11%
Happy Slapping	6.78%	18.08%

4 Conclusion and Outlook

The majority of existing publications mostly concentrate on digital threats, explaining their characteristics and implications. While a limited subset of these address the topic of digital competencies, their focus primarily relates to competencies required for using digital media, rather than encompassing strategies

for mitigating threats within the realm of digital security. In the context of educational curricula, regional differences become visible. Some countries follow centralized while others prefer decentralized approaches in regards of responsibilities for curricula. Consequently, a multitude of curricular frameworks result, often characterized by loosely defined requirements in regards to competencies in the area of digital security. These circumstances make it challenging when it comes to evaluating the extent to which competencies addressing digital threats are being considered. In any case, there is a need to catch up in this area, so children and teenagers can improve and develop competencies in the field of digital security. With this contribution, we want to comprehensively examine digital threats for children and teenagers and point out that teachers are aware that pupils are confronted with them. Our research findings confirm the presence of a multitude of digital threats that children and teenagers encounter. Moreover, this study is the first of a kind engaging Austrian educators in a discourse concerning threats within the realm of digital security. By disclosing their perspectives on the challenges confronting children and teenagers, a unique insight was created. The next step is to ask children and teenagers as well as parents about their experiences in subsequent studies. While this has already been partially done by previous studies, these mostly covered only a small part related to digital threats. The upcoming acquisition of complementary data is expected to unveil the major threats children and teenagers have to cope with from different perspectives. In the course of a dissertation, exploring this issue will involve investigating the relevant threats faced by children and teenagers across different age groups. Furthermore, the focus will be on defining competencies required to effectively encounter these threats.

References

1. Bundesministerium für Inneres, Bundeskriminalamt: Polizeiliche Kriminalstatistik 2022: Die Entwicklung der Kriminalität in Österreich (2023), https://bundeskriminalamt.at/501/files/2023/PKS_Broschuere_2022.pdf
2. Eichenberg, C., Auersperg, F.: Chancen und Risiken digitaler Medien für Kinder und Jugendliche: ein Ratgeber für Eltern und Pädagogen. Hogrefe, Göttingen, Germany, 1 edn. (2018)
3. Gasser, U., Cortesi, S., Gerlach, J.: Kinder und Jugendliche im Internet: Risiken und Interventionsmöglichkeiten. hep, Bern, Switzerland, 1 edn. (2012)
4. O’Keeffe, G.S., Clarke-Pearson, K., on Communications, C., Media: The Impact of Social Media on Children, Adolescents, and Families. *Pediatrics* **127**(4), 800–804 (04 2011). <https://doi.org/10.1542/peds.2011-0054>
5. Reid Chassiakos, Y.L., Radesky, J., Christakis, D., Moreno, M.A., Cross, C., COMMUNICATIONS, C.O., MEDIA, Hill, D., Ameenuddin, N., Hutchinson, J., Levine, A., Boyd, R., Mendelson, R., Swanson, W.S.: Children and Adolescents and Digital Media. *Pediatrics* **138**(5) (11 2016). <https://doi.org/10.1542/peds.2016-2593>, e20162593
6. Smahel, D., Machackova, H., Mascheroni, G., Dedkova, L., Staksrud, E., Ólafsson, K., Livingstone, S., Hasebrink, U.: EU Kids Online 2020: Survey results from 19 countries. EU Kids Online (2020). <https://doi.org/10.21953/lse.47fdeqj01ofo>

Exploring the Relationship between Digital Competences and Understanding of Informatics Education: A Study on Primary School Teachers

Gabrielė Stupurienė^[0000–0001–5577–1054]

Vilnius University, Universiteto str. 9, Lithuania
gabriele.stupuriene@mif.vu.lt

Abstract. This study investigates the relationship between primary school teachers' digital competencies and their comprehension of the purpose of Informatics education in primary school. The research collected data through online questionnaires from both in-service and pre-service teachers. The results reveal that while many experienced teachers needed help correctly identifying the objectives of Informatics education, a higher percentage of pre-service teachers demonstrated a better understanding. The study highlights a strong link between overall digital competencies and the grasp of Informatics education's purpose among pre-service teachers, underlining the need for targeted training programs.

Keywords: Informatics education · primary education · pre-service and in-service teachers · digital competences · SELFIE for TEACHERS tool.

1 Introduction

The Digital Education Action Plan (2021-2027) [1] identifies Informatics education as one of the priorities to improve digital skills and competences for digital transformation. Overall, Informatics education in primary and secondary schools is crucial for preparing young students for a rapidly evolving digital world and uncertain future [2]. In the renewed curriculum in Lithuania, the introduction of Informatics subjects will start in primary school [3]. The Informatics curriculum for primary schools includes six areas: algorithms and programming; digital content creation; data mining and information; technological problem solving; virtual communication and collaboration, and safe behavior [4].

The European Commission developed a tool called “SELFIE for TEACHERS” to foster the development of teachers' digital competence through self-reflection. The tool is based on the DigCompEdu Framework and is designed to help teachers reflect on and further use digital technologies [5]. The tool consists of 32 statements in six areas. It is an online tool to help primary and secondary teachers learn more about their digital competences and identify areas where they can develop further.

The research aims to determine if there is a relationship between primary school teachers' digital competencies and their understanding of the purpose of

Informatics education in primary education. This poster presents brief results from in-service and pre-service teachers who participated in the study to get a broader perspective.

2 Method and Participants

The quantitative data were collected via online questionnaires from June 2022 to February 2023. The Pearson correlation test was used to measure the strength of the linear relationship between two variables.

For this study, it was essential to find out how teachers perceive the subject of Informatics in primary education. In renewed curricula [3], the purpose of Informatics is defined in terms of five objectives: (1) to promote the safe and effective use of the various digital communication tools; (2) systematically develop computational thinking; (3) to apply programming knowledge in practice; (4) to improve students' digital skills; (5) to encourage the creative use of digital technologies to solve a wide range of problems. So, teachers were asked to answer the question: "What do you think is the purpose of Informatics education in primary education?". Teachers had to choose all five provided statements to show they understood the purpose of Informatics in primary education.

The "SELFIE for TEACHER" self-assessment tool was used to evaluate teachers' digital competences. Teachers were asked to provide their proficiency levels (A1, A2, B1, B2, C1, and C2) overall and in each of the six areas set with this tool. A1 means the lowest level, and C2 means the best level.

The study involved 17 future primary school teachers (3rd-year students) from Vilnius University. The average age was 22.29 years, with a range of ages from 21 to 30 years. All of them have an opportunity to do a traineeship: 59% in both public and private schools, 35% only in public schools, and 6% only in private schools.

Also, 72 primary education teachers working in primary education in different regions of Lithuania were involved. The average age was 47.78 years, with a range of ages from 22 to 66 years. 50% of teachers have experience teaching Informatics and computational thinking in formal and non-formal education. 36% of respondents are only in formal education, and 11% are only in non-formal education. Only 3% of respondents have no experience teaching Informatics at school.

3 Results

The results show that only 39% ($n=28$) of in-service teachers correctly chose all five statements describing the purpose of Informatics in primary education. 7% ($n=5$) of teachers didn't see the link to programming knowledge. 6% of teachers' perception of Informatics is only related to digital technologies. The same other 6% of teachers see no relation to computational thinking and programming.

Meanwhile, 65% (n=11) of the pre-service teachers chose all five statements correctly. 18% of students didn't see the relationship with programming knowledge and chose four statements out of 5.

It is also essential to show which statement describing the purpose of Informatics education must be clarified for teachers. Pre-service teachers and in-service teachers perceived the statement "to apply programming knowledge in practice" as the most related to Informatics education (Fig. 1). It means that during teacher training, we need to pay more attention to programming activities and show teachers how it can be integrated during the lessons. Only 69% of in-service teachers perceived the development of computational thinking skills as part of Informatics education. The results of the collected overall proficiency

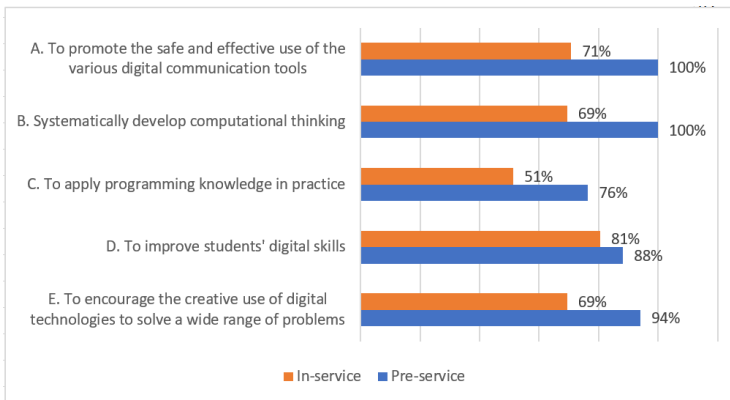


Fig. 1. Percentages of teachers who chose each statement.

levels of digital competences showed that pre-service teachers identified three main levels B1-C2. When comparing within the in-service teacher group, they identified to have main levels A2-B2.

When we compare results in six digital competencies areas, the results show that in the Professional engagement area, pre-service teachers indicated at least level A2, most of which achieved levels B1 and B2. In-service teachers indicated all levels, but mostly B1 (31%). In the second area (Digital Resources), pre-service teachers indicated at least level B1 (18%), and most of them level B2 (53%) as well as C1(24%). In-service teacher group, levels vary from A1 to C2, with most teachers with level A2 (27%), B1 (28%), and B2 (21%). In the areas of Teaching and learning (3rd) and Assessment (4th), the results of in-service teachers were quite similar, with most teachers achieving levels of A2-B2. Pre-service teachers mostly reached the same three levels of digital competences. In the area of Empowering learners, pre-service teachers achieved better results in levels B1-C2, and it can be assumed that they are more ready to engage students actively and include learners with different needs. In the area of Facilitating learners' digital competence, the main level of pre-service teachers was B1 (47%),

while in-service teachers reached mainly A2 (28%) and B1 (25%). Pre-service teachers are more confident in incorporating learning activities by using digital technologies.

In the group of pre-service teachers, the Pearson correlation coefficient of 0.497, observed at the 0.05 level, indicates a strong significant correlation [6] between the variables (level of overall digital competencies and understanding of the purpose of Informatics education in primary education). The Pearson correlation coefficient of 0.233, observed at the 0.05 level, indicates a relatively low degree of significant correlation between the variables in the group of in-service teachers.

4 Conclusion

The results highlight several important findings. Firstly, it was found that only part of in-service teachers (39%) correctly identified all five statements describing the purpose of Informatics education. Some teachers needed to recognize the link between Informatics education and programming knowledge, while others perceived it solely in terms of digital technologies. In contrast, a higher percentage (65%) of pre-service teachers correctly identified all five statements, indicating a potential gap between the training and readiness of future educators compared to their experienced counterparts. Additionally, the analysis of teachers' digital competences revealed variations in proficiency levels among both in-service and pre-service teachers. Pre-service teachers generally demonstrated higher levels of competence, suggesting their readiness to engage students with diverse needs using digital tools. Furthermore, the study established a strong and significant correlation between the overall level of digital competencies and understanding the purpose of Informatics education for pre-service teachers. This indicates that higher digital competency levels are associated with a more comprehensive grasp of the objectives of Informatics education. In contrast, in-service teachers' correlation between these variables was relatively low, potentially indicating a more diverse range of factors influencing their perception.

Acknowledgement

This project has received funding from European Social Fund (project No 09.3.3-LMT-K-712-23-0083) under a grant agreement with the Research Council of Lithuania (LMTLT).

References

1. The Digital Education Action Plan (2021-2027). <https://education.ec.europa.eu/focus-topics/digital-education/action-plan> Last accessed 20 Sep 2023

2. European Commission / EACEA / Eurydice. (2022). Informatics education at school in Europe. Eurydice report. Luxembourg: Publications Office of the European Union.
3. Stupurienė, G., Gülbahar, Y.: Informatics at Primary Education: Teachers' Motivation and Barriers in Lithuania and Turkey. LNCS, vol. 13488, pp. 27—39. Springer International Publishing (2022).
4. Dagienė, V., Jevsikova, T., Stupurienė, G., Juškevičienė, A.: Teaching computational thinking in primary schools: Worldwide trends and teachers' attitudes. *Computer Science and Information Systems* 19(1), 1–24 (2022).
5. Malagoli, C., Bocconi, S., Earp, J.: Contextual and Organisational Factors in the Development of Teachers' digital Competence. In: EDULEARN21 Proceedings, pp. 8240–8240 (2021).
6. Puth, M.-T., Neuhäuser, M., Ruxton, Graeme D.: Effective use of Pearson's product–moment correlation coefficient. *Animal Behaviour* 93, 183—189 (2014).

Teaching an Elective Course about Quantum Computing

Jihyun Kim¹, Chaeyeon Lee², Jisoo Song³, Chaeyoung Sim², and Seongbin Park²

¹ University of Pennsylvania, Philadelphia, PA, USA

² Korea University, Seoul, Korea

³ LoadIT, Seoul, Korea

jhkim10@seas.upenn.edu, lcysh@korea.ac.kr, jisoo@loadit.co.kr,
{simcy1024, hyperspace}@korea.ac.kr

Abstract. In this poster, we report our experience about teaching a one-semester elective course titled “Quantum computer and Computation” at a university in 2022. The course was intended to introduce 85 students from different disciplines ranging from liberal arts to engineering to the basic ideas about quantum computation and the power of quantum computing by exhibiting the existence of computational problems solvable efficiently by quantum algorithms, but not by classical algorithms. Especially, the emphasis was placed on intuitive explanations about the subjects in the course as much as possible because most students in the course had little or no knowledge of quantum physics. This course is offered annually and the objective of the course is to help students become quantum literate.

Keywords: quantum computing · quantum literacy · P vs NP.

1 Introduction

This poster is concerned with our experience about teaching an elective course about quantum computing at a university. The objective of the course was to provide students without background in quantum physics with basic ideas about quantum computers as well as simple quantum algorithms that exploit quantum mechanical properties. To motivate the power of quantum computers and the limits of classical computation, we also explained the idea about computational complexity, P versus NP question, and some of well-known classical algorithms.

As was argued in [1], we thought that knowledge of quantum physics would not be necessary to understand basic ideas about quantum computers especially since our goal was not to make experts, but to help students become quantum literate so that they understand basic quantum algorithms and claims made about applications of quantum technologies [2–4].

2 Ongoing and Current Work

The semester consisted of 16 weeks of lectures and the sequence of subjects addressed were as follows.

1. The class started with a motivation to quantum computing by mentioning the Moore's law, quantum supremacy, and quantum parallelism [5–7]. Then, core concepts such as a qubit, unitary transformations, superposition, entanglement, reversible computing etc. were explained. Out of these subjects, students had difficulty understanding entanglement and it was mentioned that Einstein did not believe in entanglement, either even though physicists believe that it is actually what is happening [8]. In addition, a concrete example that illustrates how entanglement can help cut down the number of steps for computation was explained [9]. This part of the course took about four weeks.
2. Then, a typical structure of a quantum algorithm was explained. It was mentioned that the description of a quantum algorithm is done by stational changes over time. While this is similar in classical algorithms, from the perspective of programmers or algorithm designers, it is different in that the description of a quantum algorithm is not done by a sequence of instructions, but by a sequence of unitary operations [10]. In addition to this, it was mentioned that the result of a quantum algorithm is obtained by measuring the state at the end. It took about a week for this part of the course.
3. For the following seven weeks or so, we explained both well-known quantum algorithms as well as classical algorithms.
 - (a) More specifically, we started with Deutsch's algorithm to point out that for some specific problem, quantum computation is clearly better than classical computation. Then, we explained Deutsch Jozsa algorithm that addresses a more general situation than the one addressed by Deutsch's algorithm. In addition to these, Grover's algorithm was explained in order to emphasize that the algorithm outperforms classical algorithms for the problem of unstructured database searching. Then, both Simon's algorithm and Shor's algorithm were explained by mentioning their similarity.
 - (b) In this part of the course, we also introduced computational problems that are believed to be difficult. These included the integer factorization problem, the graph isomorphism problem, etc. and it was mentioned that Shor's algorithm solves the integer factorization problem efficiently although as of yet we do not know whether classical algorithms could solve the problem efficiently or not.

This naturally led to the discussion of the P versus NP question and the NP completeness of many well-known computational problems. The graph isomorphism problem was mentioned because although many problems believed to be hard are known to be NP complete, it is one of few problems that are believed to be hard, but it is not known yet whether it is NP complete or not. In addition, the status of the integer

factorization problem is similar in that although it is believed to be hard, we do not know yet whether it is NP complete or not. At this stage, some classical algorithms that are relevant such as Miller's algorithm, Fast Fourier Transform algorithm, Euclid's algorithm, etc were explained.

4. For the remaining part of the semester, we addressed two views on analyzing algorithms. That is both of the worst case complexity and the average case complexity were explained and differences of these were mentioned. The reason that these were addressed lied in the following two facts. One is that many quantum algorithms are probabilistic by nature [11]. The other is that there are problems known to be hard in the sense that they are NP complete, but these problems exhibit phase transitions [12]. While the latter is not directly related to quantum computation, we thought that would be relevant because the problem that exhibited a phase transition phenomenon (in this case, it is the boolean satisfiability problem) has strange properties in that while it is believed to be hard, for a lot of instances of the problem they can be quickly answered correctly by classical algorithms. Yet, we do not know yet whether it could be solvable efficiently by any quantum algorithm.

3 Conclusion and Future Directions

In this poster, we reported a sequence of subjects that we taught in a course about quantum computing at a university. Students who enrolled in this course had little or no knowledge of quantum physics, but most students wrote positive feedbacks about the course in general, although a few students mentioned that the contents were too difficult to follow. It appeared that some students thought that the course was entirely for quantum computation, but there were actually subjects that were not directly related to quantum computation. We believe that it is important to point out that there are certain computational problems that can be solvable by quantum algorithms efficiently, but not by classical algorithms yet. And this inevitably brings up certain aspects of classical computation such as P vs NP question as well as the average case complexity. Currently, we are working on a way to organize the set of concepts and subjects used in the course in a more streamlined fashion in the course to be offered in the fall semester of this year. In addition, we are editing the course materials used in order to give lectures about quantum computing in a high school as a part of lecture series in a high school credit system.

Acknowledgements Seongbin Park is the corresponding author.

References

1. Recent progress in quantum algorithms, Dave Bacon, Wim van Dam, *Communications of the ACM*, February 2010, Vol. 53, No. 2

2. Hands-on Lab Skills Key for Quantum Jobs, Katherine Wright, 2020, <https://physics.aps.org/articles/pdf/10.1103/Physics.13.163>. Last accessed 11, June 2023
3. Quantum Physics Literacy Aimed at K12 and the General Public, Caterina Foti, Daria Anttila, Sabrina Maniscalco, Maria Luisa Chiofalo, *Universe*, 2021, 7(4)
4. The challenge and opportunities of quantum literacy for future education and trans-disciplinary problem-solving, *Research in science & technological education*, 2023, Vol. 41, No. 2
5. Technologies and Computing Paradigms: Beyond Moore's law?, Daniel Etiemble, <https://arxiv.org/pdf/2206.03201.pdf>. Last accessed 15, August 2023.
6. Quantum computing and the entanglement frontier, John Preskill, <https://arxiv.org/pdf/1203.5813.pdf>. Last accessed 15, August 2023.
7. An introduction to quantum computing for non-physicists, Eleanor Rieffel, Wolfgang Polak, *ACM Computing Surveys*, Vol. 32, No. 3, September 2000
8. Explorers of Quantum Entanglement Win 2022 Nobel Prize in Physics, Billings, October 4, 2022, <https://www.scientificamerican.com/article/explorers-of-quantum-entanglement-win-2022-nobel-prize-in-physics1/>. Last accessed 15, August 2023.
9. Demonstration of essentiality of entanglement in a Deutsch-like quantum algorithm, He-Liang Huang, Ashutosh K. Goswami, Wan-su Bao, Prasanta K. Panigrahi, *Science China, Physics, Mechanics & Astronomy*, June 2018, Vol. 61, No. 6
10. Quantum computing classical physics, David A. Meyer, *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, Vol. 360, No. 1792, 2002
11. Average case quantum lower bounds for computing the Boolean mean, A. Papageorgiou, *Journal of Complexity*, Vol. 20, No. 5, 2004
12. Computing Science: Can't Get No Satisfaction, Brian Hayes, *American Scientist*, Vol. 85, No. 2, 1997

Teaching Quantum Computing at a Middle School

Sunrim Lee^{1,2}, Yuri Kim², Soyeon Jeong², and Seongbin Park²

¹ Sehwa Girls Middle School, Seoul, Korea

² Korea University, Seoul, Korea

{tjsfla1207, fhrml1004, jj2709, hyperspace}@korea.ac.kr

Abstract. The field of quantum computing has been progressing rapidly recently and universities started to offer courses about quantum computing [1]. On the other hand, whether teaching quantum computing at a middle school can be helpful or not has not been explored yet [2]. We believe that it will be certainly helpful because early exposure to quantum computing will help students be quantum literate [3]. In this poster, we report a preliminary result that supports our hypothesis which asserts that teaching quantum computing at a middle school will help.

Keywords: quantum computing · quantum literacy · physical system.

1 Introduction

In this poster, we are concerned with whether it is appropriate to introduce quantum computing to middle school students. It is true that many people are aware of the importance of quantum computing [3] and well known companies such as IBM, Google, Intel etc put efforts to make quantum computers with many qubits, in which a qubit is the unit of information on a quantum computer [4]. However, it may be too difficult for middle school students to understand principles of quantum computation because it requires the knowledge about matrices and other mathematical concepts that students do not learn in schools.

To investigate whether it can help if we introduce middle school students to quantum computing, we organized an introduction course about quantum computing that consisted of two sessions of 45 minutes each in May 2023. The number of students participated in the course was 32 and based on the answers to questionnaires from the participating students at the end of the course, we came to believe that there certainly will be gains for students.

2 Current and Ongoing Work

In this section, we describe the sequence of subjects that were taught in the course. The first session focused on basic ideas about quantum computing and the subjects explained were as follows.

1. The session started with the general ideas about quantum technologies by mentioning differences between classical physics and quantum physics. Although students do not learn quantum physics in school, most students already heard of it because they knew the movie “Ant-man and the Wasp: Quantumania” [5]. It was mentioned that there is a world that we cannot experience. This naturally led to a discussion of differences of the two worlds which are the world that is describable by classical physics and the sub-atomic world that can be describable by quantum physics, respectively.

Then, Moore’s law [6] was mentioned in order to point out one of the reasons why we need a new kind of a computer. It took about 20 minutes or so for this part of the session and generally students did not have difficulty understanding the subjects explained.

2. For the remaining 25 minutes of the first session, a comparison between a classical computer and a quantum computer was made by pointing out a similarity and a difference. They are similar in that both are physical systems [7] whose operations are governed by physical laws. One of their differences is the unit of information used in both types of computers. While students are familiar with the notion of a bit, none of them heard of a qubit previously and they had hard time understanding it.

It was mentioned that the implementation of quantum computers requires a lot of qubits in general because as the number of qubits increases the number of different possibilities maintained by a quantum computer increases exponentially, which is not possible on a classical computer.

At this stage, the concept of superposition as well as entanglement were explained and there were questions from some students about why these phenomena occur and how they could be exploited for quantum computation.

The second session focused on hands-on experiences with simple quantum programs and the subjects addressed were as follows.

1. This session began with introducing different quantum operations that can be used for quantum computation. These included Hadamard operation, CNOT operation, and NOT operation.

Then, we used Azure quantum computing service [8] to run simple quantum programs. Students had to register for the service using Azure for students [9]. After registration, it was possible to run a sample program, “HelloWorld” written in Q# programming language [10].

While students did not have any experiences about quantum programming, the code was relatively straightforward and they could see the result of executing their first quantum program on their computers. It took about 20 minutes for this part of the session.

2. For the next 15 minutes, a second quantum program written in Q#, “Bell-State” was explained. Unlike the first program, the code was relatively complex and the teacher explained the code on a whiteboard. In addition, the

result was shown on the teacher's computer. This was a hardest part of the entire sessions and we thought that it would be better to use a different example to explain the idea of quantum entanglement.

3. For the remaining part of the session, we asked questions about whether they got interested in quantum computing as well as whether they had intentions to take a more comprehensive class about quantum computing. Most students responded positively to the first question while a few students mentioned that they would like to learn more by taking a comprehensive class if it is offered.

3 Conclusion and Future Directions

In this poster, we reported our experience about teaching quantum computing at a middle school. Before we gave an introduction course, we thought that not many students would be interested in knowing principles of quantum computing. On the contrary, what we found was that most students were interested in the subjects that were explained during the course. It is expected that computer science educators will play important roles to make quantum computing technologies accessible to all levels of audiences [11]. In addition, a quantum computing course can be a good example of STEM education [1]. Currently, we are investigating ways by which the same subjects can be introduced with more intuitive and simpler examples than what we used. For example, we may adapt the CHSH game [12] to explain the concept of entanglement as a game. In addition, we plan to explore how quantum computing fits into STEM education.

Acknowledgements Seongbin Park is the corresponding author.

References

1. Quantum Computing Is the Future, and Schools Need to Catch Up, Olivia Lanes, <https://www.scientificamerican.com/article/quantum-computing-is-the-future-and-schools-need-to-catch-up/> Last accessed 12 August, 2023.
2. Research Proposal: Exploring Quantum Informatics for Middle School Students, Giulia Paparo, ICER, 2022
3. The challenge and opportunities of quantum literacy for future education and transdisciplinary problem-solving, Laurentiu Nitaa, Laura Mazzoli Smithb, Nicholas Chancellor and Helen Cramman, *Research in Science & Technological Education*, Vol 41, No 2, 2023
4. Assessing the Quantum Computing Landscape, Advait Deshpande, *Communications of the ACM*, October 2022, Vol 65, No 10
5. Ant-man and the Wasp: Quantumania <http://https://www.imdb.com/title/tt10954600/> Last accessed 10 June, 2023.
6. Moore's law <https://www.intel.co.kr/content/www/kr/ko/history/museum-gordon-moore-law.html> Last accessed 10 June, 2023.

7. The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines, Paul Benioff, Journal of Statistical Physics, Vol 22, No 5, 1980
8. Azure quantum computing service <https://learn.microsoft.com/en-us/training/modules/get-started-azure-quantum/> Last accessed 10 June, 2023.
9. Azure for students <https://azure.microsoft.com/en-us/free/students/> Last accessed 10 June, 2023.
10. The Q# quantum programming language user guide <https://learn.microsoft.com/en-us/azure/quantum/user-guide/> Last accessed 13 August, 2023.
11. Exploring Quantum Reversibility with Young Learners, Diana Franklin, Jen Palmer, Wooring Jang, Elizabeth M. Lehman, Jasmine Marckwordt, ICER, 2020
12. A New Quantum Game Based on CHSH Game, Alan Bojić, JIOS, Vol 37, No 1, 2013

From Verbalization in Problem Solving on Computational Thinking Tasks to the Abstraction of Block Programming Concept under Scratch

Karima Sayeh^[0009-0002-6964-7155]

Al Awael School Group for Education and Learning, Algeria
Educational Museum of Mathematics and Computing
Lasmars@yahoo.fr

Abstract. Bebras challenge was introduced in Algeria since 2018, both of number of participants and schools are increasing, they use digital and pen and paper version, all student's categories are concerned by the challenge from 6 to 18 years old. Computational thinking is taught in the school one or two hours for each class during the week. The first-year students use only paper and pen activities. The second year, middle school students (11 years) participate to an experimentation, to prepare them in programming using Scratch, the teacher asks students after they find the solution to describe each (reasoning) and actions of the activity, we collect these steps on paper. This method is used by experimented persons which is usually automated step by step how they achieve the solution, they verbalize the solution. Most of the students find difficulties in writing their steps even the solution is correct. In this paper we analyse students' verbalization and we highlight the impact of this action on Scratch block. We used the concept of scheme of Vergnaud to analyse students' activity.

Keywords: Computational thinking, verbalization, scheme

1. Introduction

Bebras is an international initiative aiming to promote Informatics (Computer Science, or Computing) and computational thinking among school students at all ages. Participants are usually supervised by teachers who may integrate the Bebras challenge in their teaching activities. In Algeria, many schools attend the challenge even they don't teach computer science.

The experiment take place in middle school in Algeria, It should be noted that teaching computer science in Algerian curriculum is only present in secondary school education for two years. However, the mathematics curriculum in middle school presents activities in Excel. To prepare students to use the digital technologies, the school has introduced computer science in all classes using one hour per week.

Since 2018, the content of the courses was changed and was focused on Bebras tasks to introduce computer science concepts.

2. For scheme to verbalization in Bebras computational thinking problem solving

In our experiment, we use the concept of Vergnaud's scheme. Schemes are mental structures that evolve as students acquire new knowledge and skills. Applying this theory, we examine how students approach computational thinking problems, what schemas they use, how those schemas evolve, and how teachers can guide them to develop more advanced schemas. This allows a better understanding of how students learn and solve computational thinking problems. The principle of verbalization in problem solving related to computational thinking describes out loud between students then writing each reasoning (procedure, goals and action of the students' activity) it allows to concretize thoughts, actions and reasoning.

According to Vergnaud (1991) conceptual field theory, characterizes situations as tasks "any complex situation can be analyzed as a combination of tasks". Any solvable problem can be defined as an initial situation with a goal to achieve. Vergnaud (1991) describes "scheme" concept and encompasses it around its constituents: its goal, sub-goals and anticipations, rules of action for taking information and controls, operational invariants (concept-in-act and theorem-in-act) and possibilities of inferences, which he generalizes through the organization of the activity.

3. Verbalization

The student (in computational thinking) describes aloud then transcribes each reasoning (procedures, goals, problems) and action of his activity. This technique makes it possible to concretize thoughts, actions and reasoning. For its implementation, we consider verbalization in activity where student expresses out loud each reasoning and action carried out during his activity (real or remembered) then a cooperative verbalization where two groups compare their intermediate transcriptions to refine transcription of the solution. Faced with a task or problem solving in computational thinking, and to help students making their verbalization, we use some questions: Verbalize thoughts (what to do, what is/are the goals), verbalize his actions (how to do, what are the procedures), verbalize and justify his actions (why his actions and not others, what about the evaluation and the solution). (See table 1).

Task name	Description : be brief,- use short sentences
Verbalize your thoughts (what to do, what are the goals?)	
Verbalize your actions (how to do, what are the procedures) "	

Verbalize and justify your actions (why these actions and not others? What about the evaluation of the solution?) "	
---	--

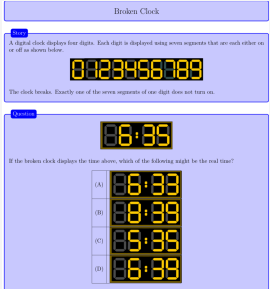
Table 1: Model table to guide students in their verbalizations
--

4. Methodology

A class of fourteen students divided into two groups took part in the experiment during a term (one hour per week). The students are asked to solve Bebras tasks by performing their written verbalization at the same time. All classes solve the same tasks. In the second step, two groups from the class confront and discuss their verbalization to refine it. The students follow a canvas pre-established by their teacher, however they are free to fill it or not. For each task, the teacher collects three paper where the verbalization is refined. Institutionalization is conducted by the teacher using student transcriptions, he often proceeds to modify their transcription by conditions or by changing the initial state of the solution. Students discover by themselves the concepts of scratch block programming. In the experiment students, try to write on paper their steps to choose the correct solution, we can see that this step start just after they find the solution, they do that at first orally, discuss together and valid the solution.

This task allows teacher to introduce algorithmic concepts, notion of variables, affectation, loops, and condition. The finality of the verbalisation is to make a bloc program under SCRATCH. All Bebras tasks are a multiple choice questionnaire (MCQ), the experience has shown that students often explain easily their steps for choosing the right solution but they do not present steps to eliminate the other options. The teacher then guide them with the opposite approach, by asking them for the same task: "Which of these answers are wrong and why?" ". This approach encouraged the students to provide a global solution to the task.

Task name	Description : be brief,- use short sentences
Verbalize your thoughts (what to do, what are the goals?)	على والعثور الصحيحة العصا على العثور منا نطلب الصحيحة الساعة We ask us to find the correct stick and to find the correct hour

<p>Verbalize your actions (how to do, what are the procedures) "</p>	<p>برقم رقم ذلك من التحقق سنحاول We'll try to check number by number</p>
<p>Verbalize and justify your actions (why these actions and not others? What about the evaluation of the solution?) "</p> 	<p>إذا كانت العصا المفقودة في الرقم 6 ، فستعطي 8 لكننا نجد 08:33 أو 8:39 والتي لا تساوي 6:35 لذلك لا يعمل، تتغير ربما نختار من بين 3 ونجد 6:95 أوه!!!!، هذا خطأ مرة أخرى نتنقل بعد ذلك إلى الرقم 5 ، حيث يمكننا تجربة ، 9 أو 6 ، ويكون الرقم 9 هو الصحيح</p> <p>If the missing stick is in number 6, it results 8 but our findings are 8:33 or 8:39 which is not equal to 6:35 therefore it doesn't work, we change.</p> <p>We might choose 3 this will result in 6:95 ohhh!! it also not correct we will then choose 5 and we will try 9 or 6 this will result in 9 which is correct</p>
<p>Table 1: Model table to guide students in their verbalizations</p>	

5. Conclusion

The experimentation of the verbalization of the solution to approach the programming with the Scratch blocks was initially a difficult task for the students, the cooperative work by discussing the solutions between peers however made this work quite motivating, the students reached at the end to present fine, abbreviated transcripts and even sometimes to use the same vocabulary in the different tasks. Institutionalization using student's verbalization favors access to the abstraction of the blocks of Scratch.

References

1. Vergnaud, G. (1991). La théorie des champs conceptuels. Recherches en didactique des mathématiques 10 (2), 133-170.
2. Kovacs B., Gaunet F., Briffault X., Les techniques d'analyse de l'activité pour l'IHM, Hermes, Lavoisier, Paris, 2004, 123-129.

From Tree to Forest: Determining the Probability of Scoring a Goal in Football Games

Jan Hendrik Krone and Johannes Fischer

Technische Universität Dortmund, 44227 Dortmund, Germany

hendrik.krone@tu-dortmund.de

johannes.fischer@cs.tu-dortmund.de

Abstract. More and more teaching tools and materials on the topic of decision trees are being developed, since they play a central role in the field of artificial intelligence. We developed teaching materials that are based on existing ones, which we extend with the Random Forest algorithm [1]. This has the advantage that the algorithm can be embedded in a group work. We used the calculation of goal probabilities in football as an example, which many students are familiar with from television and video games.

Keywords: Decision Tree · Random Forest · Machine Learning · Football · Soccer · Lower Secondary School

1 Introduction



Fig. 1: Example of two playing cards. On the left for a goal and on the right for no goal.

We present a learning approach using playing cards (see fig. 1)¹ and videos of real football scenes. The general motivating approach is to let the students

¹ The material can be downloaded at the following link: <https://tu-dortmund.sciebo.de/s/xtkh1ZMtQ2kFK1X>

watch football videos, stop the video just before the striker's shot and use the *Random Forest* algorithm to predict the outcome, which is verified afterwards by continuing the video. Decision trees have been previously learned with the ID3 algorithm [4], using the playing cards containing training data from real football scenes. The selection of a context from the students' everyday life brings several advantages, such as motivation and interest. Additionally, a context-based approach makes it easier to understand abstract concepts such as algorithms [3].

2 Theoretical Background

Expected Goals: For the English Premier League, the statistics company OPTA has published a first model of expected goals in 2012 to calculate goal probabilities. For our lesson, we use the features *distance* (0-100), *position goalkeeper* (0-5), *pressure* (0-11), *player strength* (0-100), *goalkeeper strength* (0-100), *angle* (1-179 degrees), *speed* (0 km/h-35 km/h), and the binary feature *head/foot* (see fig. 2).

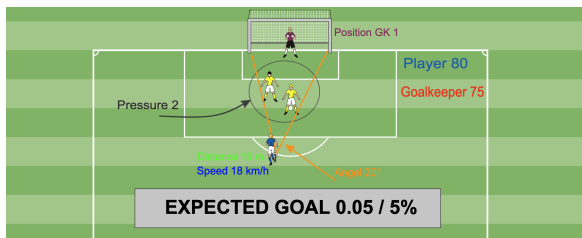


Fig. 2: Visualization of features that are relevant for the goal probability.

ID3 and Random Forest: The teaching unit is based on the *ID3* (Iterative Dichotomizer 3) algorithm to construct a decision tree. Then the *Random Forest* algorithm combines multiple trees.

ID3 is an algorithm to construct a simple decision tree [4]. The main idea is to choose a random training set and form a decision tree that correctly classifies all objects. To form a decision tree, one has to choose a feature for the root of the tree. All features are tested for their information content, and the best feature is picked as the root. Then the process is repeated with each new node.

The idea of the Random Forest algorithm [1] is to create multiple trees and then make a prediction by a majority vote. When initializing each tree, random features are used for its creation [2].

3 Implementation in school

3.1 One Feature / One Dimension

The first lesson starts with some football scenes to motivate the students. The teacher stops the video just before a player shoots. The students are asked about the outcome of the game scene and for their reasoning. The features from the explanations are collected and discussed. This results in the set of features, which is present on the cards (see section 1).

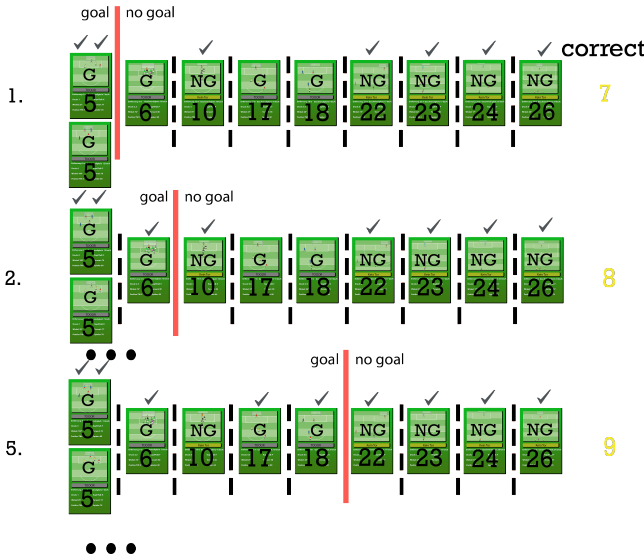


Fig. 3: Test of the feature *distance* (number on the card) for the information content. Correctly classified cards are indicated by check marks. The best dividing line is between distances 18 and 22.

When selecting the first feature in the decision tree, the first step is to find the best feature. How this process is carried out in the lesson is shown in fig. 3. The students each draw five cards from the *goal* deck and five cards from the *no goal* deck. Each group sort different cards features, seeking optimal dividing lines by exploring all options, counting correct classifications. They repeat this for all positions until all dividing lines are tested. Finally, the students compare their solutions and the best feature can be determined. This provides an opportunity to talk about the concepts of *overfitting* and *underfitting* for the first time.

3.2 Two Features / Two Dimensions

To counteract the limitation of a single feature, multiple features are used. For the extension, the students need to be able to represent two dimensions. They

create a coordinate system to plot the data. Each student gets two random features, which are used for the axes. With the help of the two-dimensional representation, it is now relatively easy to construct the decision tree (see fig. 4).

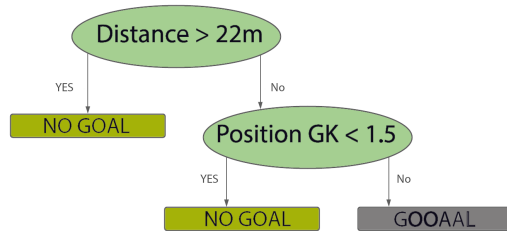


Fig. 4: Example of a possible decision tree with two features.

3.3 Random Forest

In the third part we try to improve the method by using predictions with a probability, instead of classifications. For this we use a group process to analyze the goal scenes. The decision trees created in the previous part are used again. All decision trees are evaluated simultaneously and the results are collected on the board. This collection forms a probability distribution. This method helps to predict a lot of game scenes correctly. Other scenes where goals are scored from far away or where strikers look particularly unlucky cannot be predicted correctly.

4 Future work

The units have been tested several times with different groups of lower secondary students. The students and the teachers consistently gave positive feedback. Teaching ML with the context of football is a suitable method. Next we will evaluate the effectiveness of the materials on students.

References

1. Breiman, L.: Random forests. *Machine Learning* **45**, 5–32 (2001)
2. Hastie, T., Tibshirani, R., Friedman, J.: *The Elements of Statistical Learning*. Springer New York (2009)
3. Nijenhuis-Voogt, J., Bayram-Jacobs, D., Meijer, P.C., Barendsen, E.: Omnipresent yet elusive: Teachers' views on contexts for teaching algorithms in secondary education. *Computer Science Education* **31**(1), 30–59 (2020)
4. Quinlan, J.R.: Induction of decision trees. *Machine Learning* **1**(1), 81–106 (1986)

Workshops

List of Workshops

Interactive Workshop on Teaching Materials for AI literacy

Megumi Iwata, Jari Laru, and Kati Mäkitalo

Teach AI to K12 by Manipulating a Learning Robot and Visualizing Deep Learning

Marie Martin, Morgane Chevalier, and Thomas Deneux

Reverse Engineering for Beginners in Programming Using micro:bit and Sphero Bolt

Corinna Mößlacher, Kevin Wiltschnig, and Marcell Anderlinger

Teaching Science with BBC micro:bit

Martin Cápay and Magdaléna Bellayová

Experience and Express Contextual Games for Computational Thinking and Multiple Perceptions

Ju-Ling Shih, Valentina Dagienė, and George Ghinea

Computer Science Competitions in Switzerland: Introduction and How to Make Use of Them in Schools

Susanne Datzko-Thut and Charlotte Knierim

From Modelling To Your Own 3D Printed Object

Katharina Brugger, Daniel Dobernig, and Melanie Bostjancic

Explore Computer Magic Using the Oxocards

Thomas Garaio

Discover Diverse Pedagogical Activities, With or Without Robots

Sophia Reyes Mury and Seraina Betschart