On the Role of Unplugged Programming in K-12 Education

Niklas Humble[1], Peter Mozelius[2], Lisa Sällvin[3]
[1]Department of Computer and System Science, Mid Sweden University, Östersund, Sweden
[2]Department of Computer and System Science, Mid Sweden University, Östersund, Sweden
[3]Department of Information Systems and Technology, Mid Sweden University, Sundsvall, Sweden
niklas.humble@miun.se
peter.mozelius@miun.se
lisa.sallvin@miun.se

Abstract: The integration of programming in K-12 setting is a global phenomenon with different implementations in different countries. In Sweden this is a rapid process where programming should be a part of K-12 mathematic and technology with an implementation during 2018 and 2019. The time frame has been narrowly defined, but there are few directives considering which types of programming that should be used. Three main programming types are textual programming, block programming and unplugged programming, this study has a focus on unplugged programming. The research question to answer was: Which are K-12 teachers attitudes on the role of unplugged programming in education?

The research study has been a qualitative cross-sectional study with the aim to collect teachers' attitudes towards unplugged programming halfway through their introductory programming course. Cross-sectional study is an approach to capture snapshots of an ongoing process at a given point in time. Data were collected from discussions and online postings during a workshop in the above-mentioned programming course. Participants postings have been grouped into categories in a content analysis based on the frequency of occurrence and relevance for answering the research question.

Findings show that most teachers see a benefit of unplugged programming as a means to learn the fundamental programming concepts in their teaching and learning activities. However, there are different opinions on when this unplugged introduction should occur. Some teachers also pointed out that unplugged programming could be used as an alternative to block programming and textual programming when the digital environment lacks or fails. Conclusions are that unplugged activities are a valuable complement to block programming and textual programming, but teachers have different opinions on the optimum age group for unplugged programming activities. The recommendations for K-12 teachers is to seriously consider the unplugged complement, both for pedagogical reasons and as a never-failing analogue backup.

Keywords: Unplugged programming, Block programming, Textual programming, K-12 education, Teacher professional development

1. Introduction

A global phenomenon during recent years have been the integration of programming in K-12 education, where the levels and time plans for the implementation vary between different countries (Balanskat & Engelhardt, 2015). Some of the expectations for this integration are that it should equip students with skills such as self-efficacy, problem solving and reasoning skills that can be beneficial for other subjects, and not only computational thinking skills (Psycharis & Kallia, 2017; Duncan & Bell, 2015). The Swedish government approved a new curriculum for K-12 education in mars 2017 to be implemented during 2018. This contained explicit directives for mathematics and technology to work with programming, algorithms, problem solving, and controlling physical artefacts; and with mentions of programming and digital competence as interdisciplinary traits. (Heintz et al, 2017) Even though the first mentioned time frame for the implementation have passed, many teachers still perceive a lack of concrete guidelines concerning from their schoolboards and the Swedish National Agency for Education (Humble & Mozelius, 2019).

The traditional way of learning and applying programming is trough textual programming, which could be carried out in programming languages such as Python or Java. However, two other types of programming have gained in popularity in educational settings: block programming and unplugged programming. Block programming can be conducted through the programming tool Scratch, while unplugged programming is programming without

the use of a computer. (Faber et al, 2017; Wohl, Porter & Clinch, 2015; Bell et al, 2009) Both block programming and unplugged programming are also mentioned at the website for the Swedish National Agency for Education as recommendations for K-12 teachers that want to involve programming in their teaching and learning activities (Swedish National Agency for Education, 2019).

Research on programming education has traditionally had a focus on programming in higher education, and since the implementation of programming in K-12 settings recently has started, there are relatively few studies in this area. Earlier studies on K-12 programming mainly have a focus on textual programming and block programming. In this study the focus was on unplugged programming, in a strive to answer the question: *Which are K-12 teachers attitudes on the role of unplugged programming in education?*

2. Extended background

In this section the three main types of programming, textual programming, block programming and unplugged programming, are presented with research related to educational contexts. Lastly, a brief summary of research concerning computational thinking in education is presented.

2.1 Textual programming

A computer processor can only execute binary instructions, and writing such instructions is difficult as well as time consuming for humans. To facilitate for programmers the first assembly language was developed at the Cambridge University in the 1940s where binary instructions were aligned to mnemonics. Sometimes, but not that often, computer programs are still written in an assembly language, but the vast majority of programs are today written in high-level languages where one single instruction in a programming language such as Python can correspond to tens or hundreds of machine instructions. (Gaddis, 2011)

The first high-level programming language FORTRAN was created and introduced by IBM in the 1950s, a language with textual programming interface close to modern programming languages like Java and Python. Imperative textual programming is carried out with combinations of statements, selections, iterations, constants and variables together as textual code in a file that later is checked by a compiler or an interpreter (Erwig & Meyer, 1995). Like in assembly languages, reading and writing code can be hard in traditional textual languages like C, C++ or Perl. In the early 1990s new programming languages such as Java and Python emphasised the importance of high readability (Lutz, 2001).

Python is a modern programming language with high readability and high writability designed and developed by a team led by Guido Van Rossum. The language is multi-paradigm in the sense that the Python language, seen as a snake, could change its imperative skin to an object-oriented one, or use the built-in features that supports functional or aspect-oriented programming (Lutz, 2001; Van Rossum, 2007). Python's dynamic flexibility in combination with its high readability and high writability makes the language an interesting candidate for a multi-purpose programming tool in K-12 settings. However, for the younger age groups block programming and unplugged programming are interesting alternatives.

2.2 Block programming

Block programming can be described as a graphical or visual representation of a programming code, for example with the use of graphical icons or blocks that can be combined (Lavonen et al, 2003). The development of block programming is by many considered to start with the LISP-LOGO programming language, that was developed to be an easier and more visual alternative to its textual predecessor LISP, in its use of graphical commands (Jehng & Chan, 1998). Perhaps the most widespread block programming tool used in K-12 settings is Scratch (Lye & Koh, 2018), an environment that was developed by the Lifelong Kindergarten research group at MIT Media Lab. The Scratch environment has a similar approach to programming as how we build things with LEGO bricks (Brennan & Resnick, 2012; Resnick et al, 2009). A strength mentioned about Scratch is that it includes key factors for a broad participation among both boys and girls in that it combines programming with music, storytelling and art (Maloney et al, 2010; Rusk et al, 2008).

Research suggests that block programming, with its relatively low threshold compared to textual programming, can be used to teach students the basics of computer programming, while still maintaining the opportunity to

support large and complex programming projects (Shute, Sun & Asbell-Clarke, 2017; Tundjungsari, 2016; Resnick et al., 2009). Further, studies indicate that an introduction to programming through block programming, can enhance students' performance in later programming assignments, providing a wider understanding for the place of the code in the software development cycle (Topalli & Cagiltay, 2018). Block programming tools, such as Scratch, also have a clear connection to the concept of computational thinking, since it is designed to create interactive multimedia products and through that facilitate engagement in, and development of, computational thinking skills (Lye & Koh, 2018). Block programming tools have also been used to develop frameworks for assessment of computational thinking skills (Brennan & Resnick, 2012).

2.3 Unplugged programming

Unplugged programming can be described as programming without the use of a computer (Aranda & Ferguson, 2018; Faber et al, 2017; Wohl, Porter & Clinch, 2015; Bell et al, 2009) and was first developed at the University of Canterbury in New Zealand, providing teaching material for computer science in an unplugged environment (Aranda & Ferguson, 2018; Wohl, Porter & Clinch, 2015; CSUnplugged, 2019). Unplugged programming can take various form, from boardgames (Tsarava, Moeller & Ninaus, 2018; Jagušt et al, 2018) to controlling each other or something with commands or written instructions (Aranda & Ferguson, 2018; Miller et al, 2018; Faber et al, 2017; Wohl, Porter & Clinch, 2015). An advantage mentioned about the unplugged approach, compared to the more traditional use of digital devices, is that it can be especially important for schools with low technical resources or lack of stable access to the internet or electricity (Brackmann et al, 2017).

Research surrounding unplugged programming suggests that it can be used to teach computing concept to both students and teachers (Bell & Vahrenhold, 2018; Curzon et al, 2014), as well as some aspects of computational thinking (Tsarava, Moeller & Ninaus, 2018; Brackmann et al, 2017). Unplugged activities are also used to teach basic programming and robotic manipulation to students (Miller et al, 2018; AlAmer et al, 2015). However, research suggests that there is a need for further investigations of the benefits and drawbacks of unplugged activities long-term impact on computational thinking skills, and when digital devices need to be introduced (Brackmann et al, 2017; AlAmer et al, 2015). There is also research indicating that certain types of unplugged activities have no impact on students perceived understanding of computational thinking, or on their attitudes towards computer science (Feaster et al, 2011).

2.4 Computational thinking

Computational thinking is a recurrent concept in research about programming education and its potential effect on students learning. However, research indicates that there are some uncertainties about what computational thinking entails (Shute, Sun & Asbell-Clarke, 2017; Román-González, Pérez-González, & Jiménez-Fernández, 2017; Weintrop et al, 2016; Brennan & Resnick, 2012). The concept of computational thinking stems back to the work of Papert (1980, 1991) and have its basis in the idea of constructionism (Aranda & Ferguson, 2018; Shute, Sun & Asbell-Clarke, 2017). The term was coined by Wing (2006), who described computational thinking as a way of using the fundamentals of computer science to solve problems, design systems, and understand human behaviour (Shute, Sun & Asbell-Clarke, 2017; Chen et al, 2017; Wing, 2006).

Brennan and Resnick (2012) have developed a framework for assessment of computational thinking where it is suggested that computational thinking stretch over computational concepts, computational practices, and computational perspectives as its three key dimensions. A more recent study by Shute, Sun and Asbell-Clarke (2017) suggests the following definition “The conceptual foundation required to solve problems effectively and efficiently (i.e, algorithmically, with or without the assistance of computers) with solutions that are reusable in different context”. Based on this definition and on previous research they have constructed a model for computational thinking encompassing the following aspects: decomposition, abstraction, algorithms, debugging, iteration, and generalisation (Shute, Sun & Asbell-Clarke, 2017). Considering the development of computational thinking skills, studies have indicated that computational thinking is not necessarily developed through the teaching of programming, and seems to need more explicit teaching strategies to support the development of computational thinking (Duncan & Bell, 2015).

3. The programming course

The course referred to in this study was held in Mid Sweden region for mathematics and technology teachers in K-12 education. It was given during twenty weeks in spring 2019, at 25% study pace corresponding to 7.5 ECTS. The participants had little or no prior experience in programming or how programming can be used as a tool in their education.

The course was divided in five sections: 'Programming in school, why, what and how?'. 'The fundamental building blocks of programming', 'Didactics for Technology and Mathematics', 'Didactics for programming education' and 'Project work' (Mozelius, 2018). General aspects of digitalisation and computational thinking of K-12 education were presented and discussed, before introducing the fundamentals of programming with textual programming in Python and block programming in Scratch. These two different programming types were then used in parallel throughout the course to show similarities and differences, strengths and weaknesses when used in mathematics and technology education. Unplugged programming was later introduced in the section for programming didactics.

Each section combined face-to-face meetings with lectures and interactive workshops, followed by online learning organised in the Moodle virtual learning platform. The participants were encouraged to create local study groups to collaborate and share ideas during the course.

4. Methodology

The research strategy was a qualitative cross-sectional study with the idea of capturing K-12 teachers' attitudes towards unplugged programming. An identified disadvantage with cross-sectional studies are that they only depict the situation at the actual chosen time (Bryman, 2016:690). On the other hand, this type of studies could be useful for identifying phenomena that later can be followed-up with more detailed and more longitudinal studies (Mann, 2003). Cross-sectional studies are often carried out with a quantitative design, but there are also many examples of qualitative cross-sectional studies in contemporary research (Rosemann & Szecsenyi, 2004; Moja et al., 2014; Keleş, Kavas & Yalım, 2018).

In this cross-sectional snapshot the teacher attitudes were collected halfway through an introductory programming course for K-12 teachers in mathematics and technology during the 2019 spring semester. Data were gathered during a workshop session in the course where teachers had group discussions on advantages and disadvantages of unplugged programming. Before the group discussions, a video of an unplugged programming lesson (Code.org, 2015) was shown and each teacher posted a brief summary of their individual thoughts on the subject. The activity was based on the questions: "Do you see a place for unplugged programming and computational thinking in your teaching activities?", and "What knowledge in programming would you like the students to acquire before they reach your level of education?". The video and their individual online postings were later used as the starting point for the group discussions.

The course participants online postings were analysed with content analysis for systematically examination of occurrence in the data and to identify relevant topics (Bryman, 2016:283; Drisko & Maschi, 2015:25-26). Findings from the analysis were further gathered and structured in a spread sheet document with the help of inductive coding (Driskp & Maschi, 2015:43). Results from the analysis were later compared to the observations of the workshop discussions.

5. Results and discussions

A majority in the online postings includes ideas on how to use unplugged programming in teaching and learning activities. Corresponding to findings presented in previous research, the most common fields of use mentioned were: as an introduction to other kinds of programming or computational thinking, and as a compliment to other types of programming activities (Bell & Vahrenhold, 2018; Tsarava, Moeller & Ninaus, 2018; Brackmann et al, 2017; Miller et al, 2018; AlAmer et al, 2015; Curzon et al, 2014). Some more concrete examples from the postings are: to explain a mathematical concept, to show and explain a though process, how to break down problems in smaller parts, and how to better write and structure code.

Some teachers also mention in their online postings that unplugged programming probably might constitute a majority of their programming activities in the classroom, due to the technical limitations at their schools. This

was further discussed and elaborated later during the workshop. This is coherent with the strength about unplugged programming as a way for schools with low technical resources to still get started with programming activities (Brackmann et al, 2017). However, it could also be argued that this is a sign of inequality between schools in their technical ability to make the integration of programming successful.

Duncan and Bell (2015) mention that it is not clear that unplugged activities have the desired impact on students understanding and attitudes. This was an aspect of unplugged programming that was discussed during the workshop, and mentioned in some of the online posting, as reason for not using unplugged programming in the classroom. Two examples of what was discussed, and mentioned in the online postings, were that the benefits of unplugged programming activities were unclear, and that unplugged programming possibly is more suited for younger students with no, or little, experience in programming or computers.

Regarding what programming knowledge and skills teachers would like their students to have when they reach their level of education (grade 7-12), the most common aspects discussed during the workshop and mentioned in the online postings corresponds with aspects of computational thinking (Shute, Sun & Asbell-Clarke, 2017; Brennan & Resnick, 2012). For example, that the students should have basic knowledge of programming concepts, computational thinking, problem solving, and an understanding of the instruction-structure. Some also mentioned that they would prefer if students had some prior experience in programming, for example in block programming. On the other hand, some teachers mentioned that they would prefer that students had no prior experience in programming, to better match the current level of teachers' proficiency in the field.

6. Conclusion

The conclusion of this study is that the participating K-12 teachers are generally positive towards the use of unplugged programming in education. There is, however, an uncertainty among the teachers about when and to what extent unplugged programming should be used, which corresponds with existing research in the field. An unexpected but interesting finding, was that many teachers discussed the possibility of using unplugged programming as a back-up, or even as the main programming activity, in their teaching activities due to the lack of technological resources in their schools.

Considering the need of further research on the long-term benefits of unplugged programming, mentioned in previous research, authors' recommendation is that unplugged programming should be used together with other kinds of programming and computational thinking activities, and not as an isolated activity. This have the potential of making unplugged activities a valuable compliment in the classroom, as an introductory step to block programming and textual programming. Finally, unplugged programming could also be used as a never-failing analogue back-up in case of technical problems or lack of digital equipment.

7. Future research

Since the integration of programming in K-12 education is affecting teachers to a great extent, a next interesting step would be to combine unplugged programming with textual programming and block programming in future research. Such a study could further, and to a greater extent, examine similarities and differences in teacher attitudes towards programming in K-12 education relating to the type of programming used.

References

AlAmer, R. A., Al-Doweesh, W. A., Al-Khalifa, H. S., & Al-Razgan, M. S. (2015, October). Programming unplugged: bridging CS unplugged activities gap for learning key programming concepts. In *2015 Fifth International Conference on e-Learning (econf)* (pp. 97-103). IEEE.

Aranda, G., & Ferguson, J. P. (2018). Unplugged Programming: The future of teaching computational thinking?. *Pedagogika*, *68*(3), 279-292.

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

Bell, T., & Vahrenhold, J. (2018). CS Unplugged—How Is It Used, and Does It Work?. In *Adventures Between Lower Bounds and Higher Altitudes* (pp. 497-521). Springer, Cham.

Balanskat, A., & Engelhardt, K. (2015). Computing our future computer programming and coding-priorities, school curricula and initiatives across Europe. Brussels, Belgium: European Schoolnet.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 65-72). ACM.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (Vol. 1, p. 25).

Bryman, A. (2016). *Social research methods*. Oxford university press.

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162-175.

Code.org (2015). *Unplugged Lessons in Action – Graph Paper Programming [Video File]*. Retrieved from: https://youtu.be/vBUtejDNvrs [Accessed 2019, June 4]

CSUnplugged (2019). *Computer Science Without a Computer*. Retrieved from: https://csunplugged.org/en/ [Accessed 2019, June 3]

Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014, November). Introducing teachers to computational thinking using unplugged storytelling. In *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 89-92). ACM.

Drisko, J. W., & Maschi, T. (2015). *Content analysis*. Pocket Guides to Social Work Research Methods.

Duncan, C., & Bell, T. (2015, November). A pilot computer science and programming course for primary school students. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 39-48). ACM.

Erwig, M., & Meyer, B. (1995). Heterogeneous visual languages-integrating visual and textual programming. In *Proceedings of Symposium on Visual Languages* (pp. 318-325). IEEE.

Faber, H. H., Wierdsma, M. D., Doornbos, R. P., van der Ven, J. S., & de Vette, K. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network*, *12*, 13-24.

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011, June). Teaching CS unplugged in the high school (with limited success). In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 248-252). ACM.

Gaddis, T. (2011). *Starting out with Python*. Addison-Wesley Publishing Company.

Heintz, F., Mannila, L., Nordén, L. Å., Parnes, P., & Regnell, B. (2017). Introducing programming and digital competence in Swedish K-9 education. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (pp. 117-128). Springer, Cham.

Humble, N. & Mozelius, P. (2019). *Teacher perception of obstacles and opportunities in the integration of programming in K-12 settings*. In proceedings of EDULEARN 2019.

Jagušt, T., Krzic, A. S., Gledec, G., Grgić, M., & Bojic, I. (2018, October). Exploring Different Unplugged Game-like Activities for Teaching Computational Thinking. In *2018 IEEE Frontiers in Education Conference (FIE)* (pp. 1-5). IEEE.

Jehng, J. C. J., & Chan, T. W. (1998). Designing computer support for collaborative visual learning in the domain of computer programming. *Computers in human behavior*, *14*(3), 429-448.

Keleş, Ş., Kavas, M. V., & Yalım, N. Y. (2018). LGBT+ individuals' perceptions of healthcare services in Turkey: A cross-sectional qualitative study. *Journal of bioethical inquiry*, *15*(4), 497-509.

Lavonen, J. M., Meisalo, V. P., Lattu, M., & Sutinen, E. (2003). Concretising the programming task: a case study in a secondary school. *Computers & Education*, *40*(2), 115-135.

Lutz, M. (2001). *Programming python*. " O'Reilly Media, Inc.".

Lye, S. Y., & Koh, J. H. L. (2018). Case Studies of Elementary Children's Engagement in Computational Thinking Through Scratch Programming. In *Computational Thinking in the STEM Disciplines* (pp. 227-251). Springer, Cham.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 16.

Mann, C. J. (2003). Observational research methods. Research design II: cohort, cross sectional, and case-control studies. *Emergency medicine journal*, *20*(1), 54-60.

Miller, B., Kirn, A., Anderson, M., Major, J. C., Feil-Seifer, D., & Jurkiewicz, M. (2018, October). Unplugged Robotics to Increase K-12 Students' Engineering Interest and Attitudes. In *2018 IEEE Frontiers in Education Conference (FIE)* (pp. 1-5). IEEE.

Moja, L., Liberati, E. G., Galuppo, L., Gorli, M., Maraldi, M., Nanni, O., ... & Vaona, A. (2014). Barriers and facilitators to the uptake of computerized clinical decision support systems in specialty hospitals: protocol for a qualitative cross-sectional study. *Implementation Science*, *9*(1), 105.

Mozelius, P. (2018). Teaching The Teachers To Program: On Course Design and Didactic Concepts. In *11th annual International Conference of Education, Research and Innovation*(Vol. 11). IATED.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Papert, S. (1991). Situating Constructionism. I. Harel, & S. Papert (Eds), Constructionism. Norwood.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*(5), 583-602.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. B. (2009). Scratch: Programming for all. *Commun. Acm*, *52*(11), 60-67.

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678-691.

Rosemann, T., & Szecsenyi, J. (2004). General practitioners' attitudes towards research in primary care: qualitative results of a cross sectional study. *BMC Family Practice*, *5*(1), 31.

Rusk, N., Resnick, M., Berg, R., & Pezalla-Granlund, M. (2008). New pathways into robotics: Strategies for broadening participation. *Journal of Science Education and Technology*, *17*(1), 59-69.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

Swedish National Agency for Education / Skolverket (2019, May 23). *Getting started with programming / Kom igång med programmering.* Retrieved from: https://www.skolverket.se/skolutveckling/inspiration-och-stod-i-arbetet/stod-i-arbetet/kom-igang-med-programmering [Accessed 2019, June 3]

Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, *120*, 64-74.

Tsarava, K., Moeller, K., & Ninaus, M. (2018). Training Computational Thinking through board games: The case of Crabs & Turtles. *International Journal of Serious Games*, *5*(2), 25-44.

Tundjungsari, V. (2016, February). E-learning model for teaching programming language for secondary school students in Indonesia. In *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)* (pp. 262-266). IEEE.

Van Rossum, G. (2007). Python Programming Language. In *USENIX annual technical conference* (Vol. 41, p. 36).

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127-147.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wohl, B., Porter, B., & Clinch, S. (2015, November). Teaching Computer Science to 5-7 year-olds: An initial study with Scratch, Cubelets and unplugged computing. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 55-60). ACM.