The use of Programming Tools in Teaching and Learning Material by K-12 Teachers

Niklas Humble
Mid Sweden University, Östersund, Sweden
niklas.humble@miun.se
DOI: 10.34190/EEL.21.117

Abstract: The integration of programming in K-12 education (kindergarten to grade 12) can be viewed worldwide. Expected outcomes of the integration is that students will develop skills in problem solving, creativity, logical thinking, reasoning, and computational thinking. Previous research has identified several challenges for the integration of programming, such as lack of guidance, motivation, time and competence. The aim of this study is to examine the use of programming tools in teaching and learning material by K-12 teachers. The teaching and learning material analysed in this study were collected from a website (Lektion.se) where K-12 teachers can share their teaching and learning material with each other. A document review process, inspired by a systematic literature review process, was used to select the teaching and learning material to be included in the study. The selected teaching and learning material were later analysed through content analysis with deductive and inductive coding. In the study, four types of programming tools can be found in the analysed teaching and learning material: textual programming tools, block programming tools, tangible programming tools, and unplugged programming. The findings of the study indicate potential relationships between the use of different programming tools and school subjects and student grades. These findings can be drawn upon by teachers and other stakeholders in the decision on what programming tools to integrate in classroom practice and how they are to be used.

Keywords: programming tool, programming, K-12, teaching material, learning material

1. Introduction

A worldwide trend in K-12 education (kindergarten - grade 12) is the integration of computer science and programming in the school practise and curriculum (Balanskat & Engelhardt, 2015; Rubio et al., 2015; Lindberg, Laine & Haaranen, 2019). Two frequently mentioned reasons for this integration are the future shortage of computing and informatics professionals; and that coding skills could help students understand the digitalised society (Balanskat & Engelhardt, 2015; Enders et al., 2019; Lindberg, Laine & Haaranen, 2019).

The integration, and learning, of programming is expected to help students develop useful skills, such as computational thinking, logical thinking, problem solving and creativity (Balanskat & Engelhardt, 2015; Zhang & Nouri, 2019). Previous research has pointed out challenges for the integration, such as lack of technical resources, motivation, time, guidance, competence and high-quality professional development courses for teachers (Yadav et al., 2016; Falkner, Vivian & Williams, 2018; Humble, Mozelius & Sällvin, 2020).

Sweden is one of the countries that are in the process of integrating programming in K-12 education. A new curriculum was presented by the Swedish government in 2017, which contained mentions of programming and relating concepts, both as interdisciplinary traits and in the subjects of mathematics and technology (Heintz et al., 2017). The aim of this study is to examine the use of programming tools in teaching and learning material by K-12 teachers. This work has been guided by the following research questions:

- RQ1) Which programming tools are used in the teaching and learning material?
- RQ2) How are the programming tools used in the teaching and learning material?
- RQ3) What relationship can be found between the programming tools in the teaching and learning material and school subjects and student grades?

2. Theoretical background

In a literature review study by Garneli, Giannakos and Chorianopoulos (2015), the results show that there are mainly three types of programming tools used in K-12 setting: textual programming, block (or visual) programming, and tangible programming. These three types of programming tools can also be found in a study by Wyffels, Martens and Lemmens (2014), together with a fourth programming approach that uses no computer. The activity of programming without a computer is mentioned in other studies (Heintz et al., 2017; Aranda & Ferguson, 2018) under the name of unplugged programming. The four types of programming tools are well established in previous research and will be presented in separate sub-headings below.

2.1 Textual programming

Textual programming can be described as an authentic programming technology due to its long connection to professional software design (Erwig & Meyer, 1995; Garneli, Giannakos & Chorianopoulos, 2015). Textual programming tools are characterised by the use of text editors to write and modify the code (Chen et al., 2019). Although textual programming tools are mainly used in professional context and with older students, there are examples with younger students. In a study by Tsukamoto et al. (2015), textual programming is introduced to primary school students with the motivation that 1) it is more practical to use programming tools that are similar to those used for real application development, 2) textual programming tools could be easier for primary school students than initially thought, and 3) there are existing tools for textual programming that are both easy and attractive.

Some of the textual programming tools mentioned in previous research are: Cobol, FORTRAN, Python, Java, Action Script, Arduino Integrated Development Environments, and Processing (Erwig & Meyer, 1995; Garneli, Giannakos & Chorianopoulos, 2015; Tsukamoto et al., 2015; Lindberg, Laine & Haaranen, 2019). The strong position of textual programming tools in professional contexts does not necessarily depend on it being a better approach than other alternatives. There are strong economic and personal reasons in software design to stay in the well-established, since a lot of investments have been put into it. (Erwig & Meyer, 1995) A challenge with textual programming tools is that they can be both hard to learn and difficult to teach, even with well-chosen teaching and learning materials, since many struggles with the syntax of textual programming (Garneli, Giannakos & Chorianopoulos, 2015; Yukselturk & Altiok, 2017; Lindberg, Laine & Haaranen, 2019).

An opportunity with using textual programming tools, especially with older and more advanced students, is that the professional and authentic experience could be inspiring (Garneli, Giannakos & Chorianopoulos, 2015; Lindberg, Laine & Haaranen, 2019). It is suggested in previous research that if students want to further develop their programming skills, they sooner or later need to shift to a textual programming tool (Tsukamoto et al., 2015). Another opportunity with textual programming tools mentioned in previous research, is that younger students, in primary school, can understand and use textual programming if effort is put into making the interaction fun and engaging (Tsukamoto et al., 2015).

2.2 Block programming

Block programming tools often uses representations of the code, such as graphical icons or blocks, to build programs in a similar way that LEGO bricks are used (Brennan & Resnick, 2012; Costa & Miranda, 2017). Some of the block programming tools mentioned in previous research are: Alice, Blocky, Code.org, Dwengo blocks, Greenfoot IDE, Modkit, Scratch, Squeak Etoys, S4A, and Viscuit (Wyffels, Martens & Lemmens, 2014; Garneli, Giannakos & Chorianopoulos, 2015; Wohl, Porter & Clinch, 2015; Tsukamoto et al., 2015; Costa & Miranda, 2017; Yukselturk & Altiok, 2017; Lindberg, Laine & Haaranen, 2019; Durak, 2020).

A challenge with block programming tools is that they could require more time to learn and use in educational setting, since teachers have to learn both technical and pedagogical aspects of the tools (Meerbaum-Salant, Armoni & Ben-Ari, 2013; Garneli, Giannakos & Chorianopoulos, 2015). A potential challenge is that block programming tools can produce a greater focus on the tools, rather than the programming (Wohl, Porter & Clinch, 2015). The use of block programming tools might therefore require a carefully planned and explained approach (Garneli, Giannakos & Chorianopoulos, 2015). Another challenge with block programming tools is that they might not provide advanced students with enough opportunity to develop their programming skills (Tsukamoto et al., 2015). Since block programming tools are not widely used in professional context and it is not certain that students' knowledge in block programming will transfer to textual programming tools, this could provide yet another challenge (Tsukamoto et al., 2015; Weintrop & Wilensky, 2019).

An opportunity with block programming tools is that they could provide an engaging introduction to computing since many students find the tools absorbing (Garneli, Giannakos & Chorianopoulos, 2015; Sáez-López, Román-González, & Vázquez-Cano, 2016; Durak, 2020). This could potentially support a broader appeal towards programming among students and not only the boys (Kong, Chiu & Lai, 2018). Previous research also mentions that some of the strengths with one of the more popular block programming tools, Scratch, is that it combines programming with other fields, such as music, art, games and storytelling (Maloney et al., 2010; Topalli & Cagiltay, 2018). This, together with its use of computer devices (such as a mouse), enables block programming tools for a broad participation among students with, and without, behavioural or learning disabilities (Maloney

et al., 2010; Garneli, Giannakos & Chorianopoulos, 2015); and could support the view of programming as something more than lines of code (Topalli & Cagiltay, 2018).

2.3 Unplugged programming

Unplugged programming is programming without the use of a computer (Heintz et al., 2017; Aranda & Ferguson, 2018). In the study by Wyffels, Martens and Lemmens (2014), the students give their teacher instructions to perform different tasks, and by doing so, learn basic concepts in programming. Wohl, Porter and Clinch (2015) describe it as an activity where the algorithms are enacted by the students themselves. Unplugged programming is also exemplified in previous research as boardgames and instructions to control another human (Tsarava, Moeller & Ninaus, 2018; Aranda & Ferguson, 2018).

There have been some sceptical voices raised in previous research, stated that more research is needed to determine to what extent unplugged programming can be used before the effectiveness decreases and computing devices need to be introduced (AlAmer et al., 2015; Brackmann et al., 2017). In a study by Feaster et al. (2011) the results indicate that the unplugged approach had no effect on high school students' attitudes towards computer science or perceived understanding of the concepts taught. In a study by Bell and Vahrenhold (2018) it is concluded that the unplugged approach should not be used isolated in education, but linked to current technology.

Although there are challenges associated with unplugged programming, previous research also suggests that it can serve as an engaging introduction to the field of computer science and programming, especially for younger students (Brackmann et al., 2017; Bell & Vahrenhold, 2018). Studies suggest that unplugged programming serve as a good approach to generate high understanding of the basic concept of logical predictions, algorithms and debugging (Wohl, Porter & Clinch, 2015). Perhaps the most important opportunity of unplugged programming is its easy access and low threshold for getting started, since it requires no computing devices; which could be especially important for schools and regions with a lack of technical resources (Brackmann et al., 2017).

2.4 Tangible programming

Tangible programming can be described as an activity where tangible objects are manipulated directly, for example through building or controlling physical objects (Horn, Crouser & Bers, 2012; Garneli, Giannakos & Chorianopoulos, 2015; Manches & Plowman, 2017). Tangible programming tools are often similar to other kinds of programming tools, such as textual and block programming, but incorporates physical objects to represent the programming instead of words and pictures (Horn & Jacob, 2007). Tangible programming is often developed to resemble stacking of blocks or building with LEGO (Wohl, Porter & Clinch, 2015; Hamilton et al., 2020). Some of the tools mentioned in previous research are: Arduino, Bee-Bot, Blue-Bot, LEGO-Mindstorms, Robo-blocks, Tern, and Quetzal (Horn & Jacob, 2007; Garneli, Giannakos & Chorianopoulos, 2015; Hamilton et al., 2020).

Previous research suggest that younger students can find tangible programming both engaging and easy to use (Horn, Crouser & Bers, 2012; Garneli, Giannakos & Chorianopoulos, 2015; Wohl, Porter & Clinch, 2015). However, previous research has also suggested that a challenge with tangible programming tools is that some of the older, and the more curious younger, students might prefer a graphical interface (Horn, Crouser & Bers, 2012; Garneli, Giannakos & Chorianopoulos, 2015). Researchers have therefore explored a hybrid interface between a tangible and graphical approach and found that younger students seem to be able to move fluidly between the two (Horn, Crouser & Bers, 2012).

An opportunity with tangible programming tools is that they could make programming attractive and enjoyable, especially among girls, by the promotion of collaboration, interaction, active learning, exploration, and artistic creation with light, sound and motion (Rusk et al., 2008; Horn, Crouser & Bers, 2012; Garneli, Giannakos & Chorianopoulos, 2015; Rubio et al., 2015). This could also make tangible programming appropriate for activities and discussions in the whole-class (Horn, Crouser & Bers, 2012). Another opportunity with tangible programming tools is that they could be more engaging for the students, this is suggested by previous research where tangible programming have been compared to block and unplugged programming (Wohl, Porter & Clinch, 2015).

3. Method

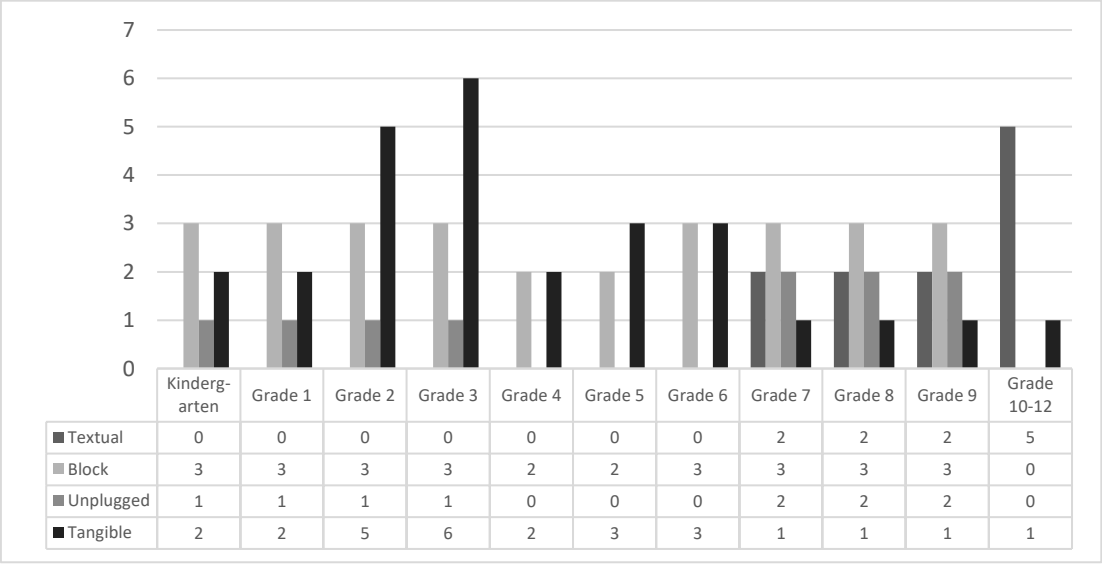
This study has been conducted as a document review with teaching and learning material as the data source. Bowen (2009) describes the document review as a method for identifying relevant and meaningful passages in a text. In this study it has been used to identify the use of programming tools in teaching and learning materials by K-12 teachers. The data in this study consists of written teaching and learning material from a Swedish website for K-12 teachers (Lektion.se), collected during late 2019. At the website, users can share their teaching and learning material to a database by creating an account and posting it. Other users can see the post, but to download and comment on the material they also need to create an account.

The selection process in this study was inspired by Tondeur et al. (2017) systematic literature review. In the first step, potentially relevant teaching and learning material was identified by conducting a database search on the following keywords in Swedish and English: programming, program, code, coding, script. This produced a list of 332 unique and potentially relevant teaching and learning material. The second step consisted of screening the title and summary of each teaching and learning material to judge the relevance for the study. This decreased the list of potentially relevant teaching and learning materials to 62. The third and final step consisted of thoroughly analysing each post as a whole (title, summary and the attached teaching and learning material) to determine the relevance for the study. This decreased the list of teaching and learning materials to 26, published between 2008 and 2019, which were chosen to be included in the study.

The selected teaching and learning materials have been analysed through content analysis. According to Bryman (2016) "[content analysis] seeks to quantify content in terms of predetermined categories and in a systematic and replicable manner" (2016, p. 283). A mixture of deductive and inductive coding has been used in the process of analysis. Deductive coding was used to structure the findings in the categories described in *2 Theoretical background*, and to connect the use of the tools to school subjects and student grades. Inductive coding was used to identify themes of use for the programming tools in the analysed material.

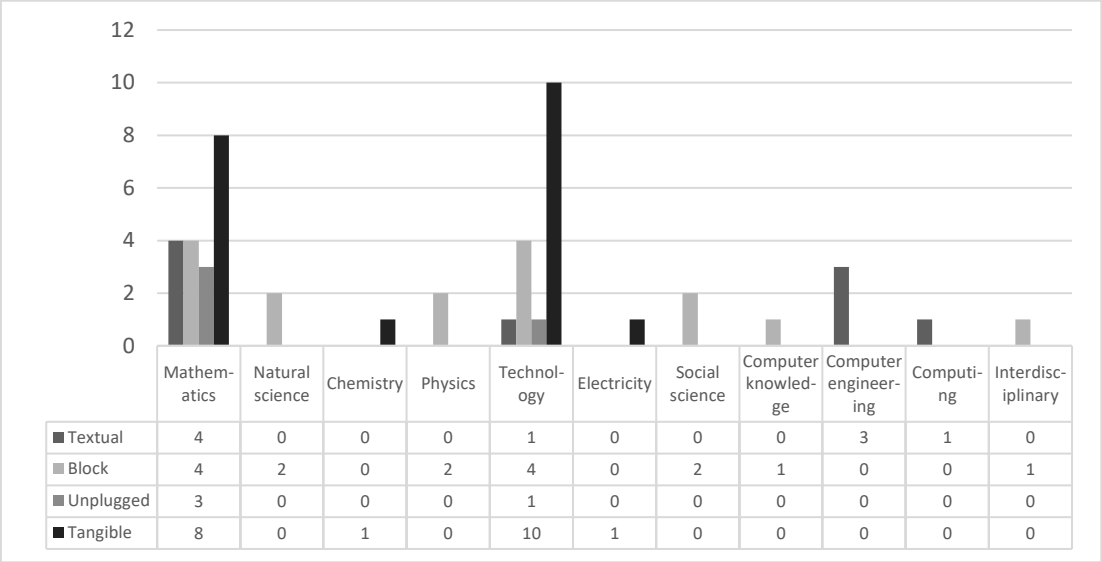
4. Results

The results of which programming tools are used in the material, which student grades (Figure 1) and school subjects (Figure 2) they relate to, and how they are being used, are presented in sub-headings according to the four types of programming tools outlined in *2 Theoretical background*.



	Kinderg-arten	Grade 1	Grade 2	Grade 3	Grade 4	Grade 5	Grade 6	Grade 7	Grade 8	Grade 9	Grade 10-12
Textual	0	0	0	0	0	0	0	2	2	2	5
Block	3	3	3	3	2	2	3	3	3	3	0
Unplugged	1	1	1	1	0	0	0	2	2	2	0
Tangible	2	2	5	6	2	3	3	1	1	1	1

Figure 1: Distribution of tool usage over student grades



	Mathem-atics	Natural science	Chemistry	Physics	Technol-ogy	Electricity	Social science	Computer knowled-ge	Computer engineer-ing	Computi-ng	Interdisc-iplinary
Textual	4	0	0	0	1	0	0	0	3	1	0
Block	4	2	0	2	4	0	2	1	0	0	1
Unplugged	3	0	0	0	1	0	0	0	0	0	0
Tangible	8	0	1	0	10	1	0	0	0	0	0

Figure 2: Distribution of tool usage over school subjects

4.1 Textual programming

Textual programming tools can be found in 7 of the 26 selected materials. The programming tools mentioned in the material are (parenthesis for the number): Java (2), Python (2), Kojo (1), Excel (1), Octave (1) and Visual Basic (1). The most common student grades for the tools to be used in are Grade 10-12 (Figure 1) and the most common school subject is mathematics (Figure 2).

There are two main usage of textual programming tools in the material: as a tool for other subjects (mathematics and technology), and to learn programming. As a tool, the material give examples of how textual programming can be used to more efficiently solve complex calculations (Example 1), and visualise geometry and steering of objects (Example 2).

> *"This lesson has the aim to show how Python programming can be used to check if a random integer is a prime or not. The assignment should be solved with a loop that checks all the integers up to the chosen integer or an appropriate span of integers."*
>
> ***Example 1**. Check prime with programming.*
>
> *"Use programming to draw an equilateral triangle. With the help of the Turtle module write code that draws an equilateral triangle. Write the code in three different ways to achieve the same result."*
>
> ***Example 2**. Programming in geometry.*

The material give examples of how textual programming tools can be used to learn the fundamentals of programming (Example 3). Some programming concepts referred to in the material are: algorithms, classes, comparisons, compilation, conversions, data types, functions, graphical user interface, methods, modules, objects, procedures, pseudo code, and variables.

> *"There are eight data types built into Java: byte, short, int, long, float, double, Boolean, and char. Everything else are classes."*
>
> ***Example 3**. Data types in programming.*

4.2 Block programming

Block programming tools can be found in 7 of the 26 selected materials. The programming tools mentioned in the material are (parenthesis for the number): Scratch (3), Code.org (2), RoboLogic LE (1), and Scratch Jr (1). The use of block programming tools is evenly spread out over the grades, with the exception of Grade 10-12 that have no usage (Figure 1). The most common subjects are mathematics and technology (Figure 2). All of the described uses in the material are centred around controlling objects in a virtual environment (Example 4). The most obvious relation to a school subject is therefore technology.

> *"Download the application 'RobotLogic LE' in AppStore. When you open RoboLogic LE, you will get the assignment to create a program that controls the robot. You use the visual commands to program your robot to execute different movements; move forward, turn left/right, jump."*
>
> ***Example 4**. Steering and technology.*

The material also touches upon concepts such as pair programming and coordinates, and is often set in a context to visualise another subject or field, such as natural science or social science. In that sense, the tools are used in an interdisciplinary way (Example 5). Some of the material relates to basic concepts in programming, such as: commands, events, functions, iteration, and variables.

> *"Program a lake [with Scratch] where one or more fishes swim back and forth. How does your lake look beneath the surface? Maybe there is an object floating on the surface."*
>
> ***Example 5**. Water and density.*

4.3 Unplugged programming

Unplugged programming can be found in 3 of the 26 selected materials. Pen and paper are used in all the material. The materials are aimed to the youngest and some of the older students (Figure 1) and mainly towards the subject of mathematics (Figure 2). The material does not explicit touch upon any concepts in programming but is centred on the practise of giving instructions, either oral or via drawings. In the material, unplugged programming is combined with either geometry or steering imagined objects in a game-like way (Example 6).

> *"Santa Claus wants to reach his sleigh. He has to collect all the Christmas gifts on the way but he does not want to walk into a Christmas tree. Draw your own Christmas gifts and trees [on the playfield]. Ask a friend to draw arrows in the column on the side."*
>
> ***Example 6**. Christmas programming.*

In one of the posts, unplugged programming is used as an introduction to textual programming (Example 7). This material is directed to students in grade 7, 8 and 9 and the subject of mathematics.

> *"The task: 1- Draw an equilateral triangle and a square with the help of a ruler, protractor, paper and a pen. 2- Use programming to draw the same geometric figures [...]. 3- Compare the two methods, what advantages and disadvantages do they have?"*
>
> ***Example 7**. Drawing geometry with and without a computer.*

4.4 Tangible programming

Tangible programming tools can be found in 11 of the 26 selected materials, which makes it the most common tool. The programming tools mentioned in the material are (parenthesis for the number): Blue-Bot (7), Bee-Bot (1), Arduino (1), programmable logic controller (1), and 1 unclear (defined only as 'the robot'). The tools are most commonly used in grade 3 (Figure 1) and the subjects of technology and mathematics (Figure 2).

Most of the use in the materials are centred around controlling physical objects, which make the subject of technology the most obvious relationship. However, much of the material directed towards the lower grades relates the controlling of objects to mathematics (Example 8).

> *"Your assignment is to program the bot [robot] to walk on the numbers in order of size. Order the numbers by size by doing the calculation tasks below. Then program the bot to walk on the answers in order of size and to pause on each answer before moving on to the next. Good Luck!"*
>
> ***Example 8**. Robots and mathematics.*

A common use is to get a robot to move over a printed carpet and stop on mathematical symbols and numbers. Most of the material does not explicit touch upon programming concepts, with some exceptions in the materials directed toward older students (Example 9). Some programming concepts that the materials relate to are: error handling, functions, iteration, and selection.

> *"important parts of a program: [...] curly brackets. [...] for. [...] if. […] case. [...] Comments. [...] Function. [...] random(). [...]"*
>
> ***Example 9**. Robot programming.*

5. Discussion

Concerning which programming tools are used in the analysed material, all categories mentioned in *2 Theoretical background* can be found. The most common are tangible programming tools, which were mentioned in 11 of the 26 analysed materials. Followed by textual and block programming tools, which were mentioned in 7 of the 26 analysed materials each. Lastly, unplugged programming was mentioned in 3 of the 26 analysed materials. The reason for tangible programming tools being the most common in the material could be the approach to programming as part of the outside world. As stated in previous research, the connection to collaboration, active learning, exploration, creation, sound and motion in tangible programming could provide an engaging and attractive entrance to programming for students (Rusk et al., 2008; Horn, Crouser & Bers, 2012; Garneli, Giannakos & Chorianopoulos, 2015; Rubio et al., 2015).

Textual and block programming tools have a greater focus on being used as tools for other subjects in the materials, which differentiates them from unplugged and tangible programming tools. Textual programming tools are primarily centred around performing complex calculations and learning programming with older students, while a more versatile and interdisciplinary use can be found in the materials on block programming tools. The difference could be explained by the programming tools history and development, where textual programming tools often are described as authentic and professional tools (Erwig & Meyer, 1995; Garneli, Giannakos & Chorianopoulos, 2015). Block programming tools does not have the same relationship to professional development, but are primarily used in educational settings. This can explain the more versatile use of block programming tools in the materials, which is also noted as one of the strengths with block programming in previous research (Maloney et al., 2010; Brennan & Resnick, 2012; Topalli & Cagiltay, 2018).

Regarding the programming tools relation to school subjects and student grade, the most common relationships are to mathematics (17 of 26) and technology (15 of 26); and the second (9 of 26) and third (10 of 26) grade. The relationship to school subjects, this is in line with the integration in Swedish K-12 education where the curriculum has specific mentions of programming in the subjects of mathematics and technology (Heintz et al., 2017). The clearest relationship to student grades can be found between textual programming tools and the upper grades, while the other programming tools have a more scattered distribution. This might not be that surprising since previous research has stated that textual programming tools' high threshold and connection to professional development could make them more suitable for older students (Erwig & Meyer, 1995; Garneli, Giannakos & Chorianopoulos, 2015; Lindberg, Laine & Haaranen, 2019).

6. Conclusion

This study has shown that there is a possibility for a variety of programming tools to be used in K-12 education; and that there are potential relationships between different types of programming tools, student grades and school subjects. This can be drawn upon by teachers and other stakeholders in the decision on what programming tools to be integrated in classroom practice, and how they are to be used. The study indicates that block programming tools and tangible programming tools are appropriate to use in activities and school subjects, in most student grades, where emphasis is put on controlling an object. For example, the subject of technology, or visualising an aspect of a school subject through the controlling of an object, for example storytelling and mathematics.

The results of the study further indicate that textual programming tools are the most specialised among the four types of programming tools, in regard to use, student grades and school subjects. This makes the textual programming tools appropriate for activities that are centred on the programming practice, for example programming visualisations of geometry or conducting mathematical calculations. Since textual programming tools have well established higher threshold for novice programmers, it could also be appropriate to use these tools with students and in settings where these difficulties can be addressed. The high threshold of textual programming tools could also, potentially, be reduced by combining the tools with an unplugged programming introduction. Since unplugged programming have a focus on giving clear instructions (without the use of a computer), this could make the students better prepared for the more complex textual programming tools.

7. Limitations and future research

Although a structured design was used in this study, there are limitations. The main limitation is the number of included teaching and learning materials in the study. This is especially obvious in the category of unplugged programming. Future research should focus on widen and deepen the analysis of which programming tools are

used and how they are used in K-12 education. This could be done through an aggregated use of multiple sources for gathering teaching and learning materials. It could also be done by combining the research with other methods for data collection, such as interviews and classroom observations.

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.

Balanskat, A., & Engelhardt, K. (2015). Computing our future: Computer programming and coding. *Priorities, school curricula and initiatives across Europe: European Schoolnet (EUN Partnership AIBSL), Brussels, Belgium*.

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.

Bowen, G. A. (2009). Document analysis as a qualitative research method. *Qualitative research journal*, *9*(2), 27.

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).

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, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education*, *29*(1), 23-48.

Costa, J. M., & Miranda, G. L. (2017). Relation between Alice software and programming learning: A systematic review of the literature and meta-analysis. *British Journal of Educational Technology*, *48*(6), 1464-1474.

Durak, H. Y. (2020). The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. *Technology, Knowledge and Learning*, *25*(1), 179-195.

Enders, T., Hediger, V., Hieronimus, S., Kirchherr, J. W., Klier, J., Schubert, J., & Winde, M. (2019). Future Skills: Six Approaches to Close the Skill Gap. *World Government Summit*.

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

Falkner, K., Vivian, R., & Williams, S. A. (2018). An ecosystem approach to teacher professional development within computer science. *Computer Science Education*, *28*(4), 303-344.

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).

Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015, March). Computing education in K-12 schools: A review of the literature. In *2015 IEEE Global Engineering Education Conference (EDUCON)* (pp. 543-551). IEEE.

Hamilton, M., Clarke-Midura, J., Shumway, J. F., & Lee, V. R. (2020). An Emerging Technology Report on Computational Toys in Early Childhood. *Technology, Knowledge and Learning*, *25*(1), 213-224.

Heintz, F., Mannila, L., Nordén, L. Å., Parnes, P., & Regnell, B. (2017, November). 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.

Horn, M. S., & Jacob, R. J. (2007, February). Designing tangible programming languages for classroom use. In *Proceedings of the 1st international conference on Tangible and embedded interaction* (pp. 159-162).

Horn, M. S., Crouser, R. J., & Bers, M. U. (2012). Tangible interaction and learning: the case for a hybrid approach. *Personal and Ubiquitous Computing*, *16*(4), 379-389.

Humble, N., Mozelius, P., & Sällvin, L. (2020). Remaking and reinforcing mathematics and technology with programming–teacher perceptions of challenges, opportunities and tools in K-12 settings. *The International Journal of Information and Learning Technology*.

Kong, S. C., Chiu, M. M., & Lai, M. (2018). A study of primary school students' interest, collaboration attitude, and programming empowerment in computational thinking education. *Computers & Education*, *127*, 178-189.

Lindberg, R. S., Laine, T. H., & Haaranen, L. (2019). Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. *British Journal of Educational Technology*, *50*(4), 1979-1995.

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), 1-15.

Manches, A., & Plowman, L. (2017). Computing education in children's early years: A call for debate. *British Journal of Educational Technology*, *48*(1), 191-201.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & Angel, P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education*, *82*, 409-420.

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.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129-141.

Tondeur, J., Van Braak, J., Ertmer, P. A., & Ottenbreit-Leftwich, A. (2017). Understanding the relationship between teachers' pedagogical beliefs and technology use in education: a systematic review of qualitative evidence. *Educational Technology Research and Development*, *65*(3), 555-575.

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.

Tsukamoto, H., Takemura, Y., Nagumo, H., Ikeda, I., Monden, A., & Matsumoto, K. I. (2015, October). Programming education for primary school children using a textual programming language. In *2015 IEEE Frontiers in Education Conference (FIE)* (pp. 1-7). IEEE.

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, *142*, 103646.

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).

Wyffels, F., Martens, B., & Lemmens, S. (2014, November). Starting from scratch: Experimenting with computer science in flemish secondary education. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 12-15).

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, *26*(4), 235-254.

Yukselturk, E., & Altiok, S. (2017). An investigation of the effects of programming with Scratch on the preservice IT teachers' self-efficacy perceptions and attitudes towards computer programming. *British Journal of Educational Technology*, *48*(3), 789-801.

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607.