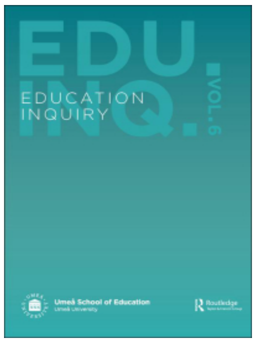

Education Inquiry

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/zedu20

# Development of computational thinking, digital competence and 21st century skills when learning programming in K-9

Jalal Nouri, Lechen Zhang, Linda Mannila & Eva Norén

**To cite this article:** Jalal Nouri, Lechen Zhang, Linda Mannila & Eva Norén (2020) Development of computational thinking, digital competence and 21st century skills when learning programming in K-9, Education Inquiry, 11:1, 1-17, DOI: 10.1080/20004508.2019.1627844

**To link to this article:** https://doi.org/10.1080/20004508.2019.1627844

EDUCATION INQUIRY
2020, VOL. 11, NO. 1, 1–17
https://doi.org/10.1080/20004508.2019.1627844

OPEN ACCESS Check for updates

# Development of computational thinking, digital competence and 21st century skills when learning programming in K-9

Jalal Nouri [a], Lechen Zhang [a], Linda Mannila[b] and Eva Norén [c]

[a]Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden; [b]Department of Computer and Information Science, Linköping University, Linköping, Sweden; [c]Department of Mathematics and Science Education, Stockholm University, Stockholm, Sweden

**ABSTRACT**

Teachers around the world have started teaching programming at the K-9 level, some due to the formal introduction of programming in the national curriculum, others without such pressure and on their own initiative. In this study, we attempted to understand which skills – both CT-related and general – are developed among pupils in the process of working with programming in schools. To do so, we interviewed 19 Swedish teachers who had been teaching programming for a couple of years on their own initiative. The teachers were selected based on their experience in teaching programming. Our thematic analysis of these interviews shed light on what skills teachers perceive pupils develop when programming. This led us to identify three themes related to CT skills and five themes related to general skills. The CT skills identified corresponded well with and were thus thematically structured according to the dimensions of CT proposed in the framework of Brennan and Resnick, namely computational concepts, computational practices and computational perspectives. In addition to the CT skills, our thematic analysis also resulted in the identification of general skills related to digital competency and 21st century skills, namely cognitive skills and attitudes, language skills, collaborative skills and attitudes and creative problem-solving skills and attitudes.

**KEYWORDS**
Computational thinking; programming; k-9; digital competency; 21st century skills

## Introduction

Computer science (CS) in general and programming in particular have traditionally been considered areas relevant to information technology (IT) professionals only. During recent years, however, this view has changed. At the current rate of digitalization, software and technology are playing an increasing role in almost all areas of society and every aspect of life. This raises a need to understand how the digital world works, as well as what opportunities and risks it brings, just as we learn about the physical world. Consequently, in the last decade, we have witnessed an active discussion concerning the role of CS and programming for everyone (e.g. Informatics Europe and ACM Europe, 2015). As a result, an increasing number of countries have introduced or

**CONTACT** Jalal Nouri jalal@dsv.su.se Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden

are in the process of introducing CS or elements thereof in their school curricula. The way in which the new content is introduced varies. Some countries, such as England, have introduced CS as a subject of its own, while others, such as Finland and Sweden (Skolverket, 2017), have taken an interdisciplinary approach, building on the concept of digital competence. Regardless of the approach, one topic – programming – has received particular attention in the debate.

Programming lies behind all the digital solutions, software and systems that we use. To understand the digital world, one thus needs to have some idea about what programming is. Programming is also a means of creating something new in the digital world, solving problems and implementing ideas. Skolverket (2017) defines programming first and foremost as a problem-solving process, but also emphasizes that programming should be considered from a broad perspective, including creativity, simulation and democratic dimensions. A common way of capturing the broad view of programming is through the umbrella term *computational thinking* (CT), which was initially coined by Papert (1980), but won new ground in the 2000s through the work of Wing (2006). Wing argues that CT is a set of general problem-solving skills based on computer science that allows computers to be used effectively in the process of problem solving. Several models have been proposed to operationalize CT (see "Computational Thinking" below).

While programming has previously been part of early age education, its addition to curricula today means new content being taught by most – if not all – teachers. As such, many questions arise related to what to teach and how to teach it in terms of developing the skills and competencies aimed for. Although there is some literature on teaching and learning programming in a school context, the overwhelming majority of research has been conducted at the university level (Heintz, Mannila, & Färnqvist's, 2016) and the use of programming to teach thinking skills in K-12 was not extensively reported (Lye & Koh, 2014). Furthermore, the existing research on K-12 levels does not have a strong focus on teachers, for instance how teachers integrate CT in education (Sjoberg, Risberg, Nouri, Noren & Zhang, 2019) or how teachers perceive CT (Sands, Yadav, & Good, 2018).

The research focusing on pupils development of CT have focused on the misconceptions and the difficulties that the novice programming learners encounter (e.g. Grover & Basu, 2017; Žanko, Mladenović, & Boljat, 2018). Researchers have also identified a few threshold concepts (Sanders & McCartney, 2016) that tend to be particularly troublesome for many novices. For instance, Carter (2015) indicated that students find it difficult to detect and handle errors in their programs. Studies have also shown that beginning programmers face difficulties in utilizing variables (Fields, Vasudevan, & Kafai, 2015; Hermans & Aivaloglou, 2017).

All in all, the research on teaching and learning programming at the university level is quite extensive, which is understandable as programming has been taught at the university level for many decades. Programming as part of K-9 education is, however, still relatively new and the research base is only beginning to form. In a review of CT in K-12 education, Grover and Pea (2013) point out that most of the research at lower levels of education has so far focused on definitions and tools for supporting CT, leaving large gaps when it comes to empirical studies. While some of the results from studies on programming among university level students might also be applicable, at least in some respects, to lower levels of education, we cannot be sure without conducting the research. At the same time, it is not clear that questions of interest at the university level are those we need to ask at lower levels of education. Consequently, we cannot take university level results for granted, but empirical studies in the classroom could deliver such findings (Grover & Pea, 2013).

In this paper, we aim to contribute to filling the research gap by investigating teachers' views concerning the CT skills pupils in K-9 education develop when programming. To accomplish this, we interviewed 19 teachers who had taught programming for at least one year. Our primary research question was as follows:

*RQ: From the teachers' perspective, what skills do pupils obtain when engaging in programming activities?*

The rest of the paper is organized as follows. First, we present the background to our work in more detail, after which we discuss our study settings and the methods used. Next, we present the results, which are then related to previous research. We conclude the paper with some final remarks and ideas for future work.

## Computational thinking

It is considered necessary to teach CT outside the area of computer science and scholars argue for doing so as early as in kindergarten (Fessakis, Gouli, & Mavroudi, 2013; Sullivan & Bers, 2016). Seymour Papert originally coined the term CT in his book "Mindstorms: Children, *computers, and powerful ideas*" in 1980. Papert referred to CT primarily as the relationship between programming and thinking skills. He believed that students' constructions, undertaken through programming with LOGO, could facilitate their procedural thinking across multiple disciplines. Unlike Papert, many definitions of CT in the 21st century emphasize concepts that are commonly involved when programming or part of computer science. Jeanette Wing, the researcher who brought CT back to public attention in 2006, refreshed the definition of CT with tremendous influence (3,467 citations upon the drafting of this article). Wing (2006) defines CT as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p. 33). She argues that CT, "just like reading, writing, and arithmetic, should be added to every child's analytical ability" (p. 33; see also Resnick, 1987). Later, Cuny, Snyder, and Wing (2010) updated the definition, stating "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (p. 1). In both cases, Wing articulated a vision that everyone can benefit from learning to use the principles, concepts and approaches common to computer science.

Some research in recent years has focused on issues related to teaching and learning the skills, concepts and practices relevant to CT (Grover & Pea, 2013). As noted above, many countries have taken actions to include CT in their curricula, for example, the new subject "Computing" in England (Department for Education, 2013), a new curriculum with a strong focus on CT in Australia (Australian Curriculum Assessment and Reporting Authority, 2015) and digital competence in Finland (Opetushallitus, 2014) and Sweden (Skolverket, 2017). Despite the rising recognition of CT, "there is little agreement on what CT encompasses" (Brennan & Resnick, 2012). The discussion around CT is at such an early stage that it is far from explaining what CT is (Kalelioglu, Gülbahar, & Kukul, 2016). The lack of a universal definition (Weintrop et al., 2016) is challenging when wanting to integrate CT within education. As Selby and Woolard (2013) pointed out, the design of a consistent curriculum and appropriate assessment is out of the question without knowing what CT entails. "How to teach" is pointless without knowing "what to teach". Hence, this study aimed to investigate what skills can young pupils learn through programming.

Based on the core concepts of CT provided by computer scientists such as Wing, several definitions have emerged for what CT means in K-12 education (Voogt, Fisser, Good, Mishra, & Yadav, 2015). These definitions of CT usually offer a list of concepts and skills that both support and differ from each other, whose most common elements are abstraction, algorithms, data, problem decomposition, parallelism, debugging & testing and control structure (Rose, Habgood, & Jay, 2017).

In parallel to efforts to operationalize CT, for instance, in the form of frameworks, the development of suitable tools can aid in supporting the development of CT in K-12 contexts (Voogt et al., 2015). Scratch, developed at the MIT Media Lab starting in 2007, is one among many popular programming languages. Brennan and Resnick's (2012) framework (see Table 1) provides a theoretical foothold to this visual block programming language. This bond between Scratch and Brennan and Resnick's (2012) framework is the first reason for adopting their framework to be the guideline regarding the content of CT skills in K-12 education. Secondly, their framework is considered as providing "a wide coverage of CT" (Kong, 2016, p. 379). Many curricula emphasize the learning of basic CT through mastering what Brennan and Resnick (2012) term "computational thinking concepts" (Falloon, 2016). Their framework has also been widely discussed and laid out as a basis for previous empirical and theoretical studies (e.g. Lye & Koh, 2014). More importantly, "Brennan and Resnick's framework concentrates primarily on the sort of knowledge used, and how it is used, when students create code rather than general thinking skills" (Falloon, 2016, p. 578).

**Table 1.** The three dimensions of CT (Brennan & Resnick, 2012).

| | Definition |
|---|---|
| Computational concepts | The concepts designers employ as they program:<br>• Sequences (S)<br>• Loops (L)<br>• Events (E)<br>• Parallelism (P)<br>• Conditionals (C)<br>• Operators (O)<br>• Data (D) |
| Computational practices | The practices designers develop as they program:<br>• Being incremental and iterative (I)<br>• Testing and debugging (T)<br>• Reusing and remixing (R)<br>• Abstracting and modularizing (A) |
| Computational perspectives | The perspectives designers form about the world around them and about themselves:<br>• Expressing (Exp)<br>• Connecting (Con)<br>• Questioning (Que) |

Brennan and Resnick's (2012) framework categorizes CT into three dimensions:

- The *concepts* designers engage with in programming, i.e. sequences, loops, parallelism, events, conditionals, operators and data;
- The *practices* designers develop as they engage with the concepts, i.e. being incremental and iterative, testing and debugging, reusing and remixing, abstracting and modularizing;
- The *perspectives* designers form about the world around them and about themselves, i.e. expressing, connecting, questioning.

## Methodology

### *Context and participants*

For some years now, many Swedish teachers in compulsory schooling have explored programming with their pupils, supported by many national and global initiatives (for instance the Maker movement). In this study, we reached out to 19 teachers who took part in a national research project that was initiated in light of the introduction of programming in Swedish basic education. The research project involved around 15 schools and approximately 150 teachers. Using an online questionnaire, we selected the teachers who considered themselves most experienced in teaching programming in kindergarten, specifically grades 1–9. In total, we selected 19 teachers who worked in nine schools located in different municipalities in Sweden. Six of these teachers taught grades 1–3 in kindergarten, seven taught grades 4–6 and the remaining six taught grades 7–9.

The teachers had approximately two years of experience in having programming as part of their teaching (minimum 1 year, maximum 4 years). The teachers worked with programming on their own initiative before the formal introduction of programming in the Swedish curriculum. In a previous article (Nouri, Norén, & Skog, 2018), we concluded that the teachers used four didactic strategies when teaching programming, namely analogue programming, programming with RoBots (BeeBots and BlueBots) and Apps, block programming and text-based programming. These four didactic strategies follow a line of progression and can also be considered programming levels. The majority of the teachers had experience with the three first strategies/levels, while only a few teachers had worked with text-based programming. Thus, when asking about the skills pupils develop, the answers, to a great extent, reflect the impact of the first three didactic strategies/levels.

### *Data collection*

Two of the researchers undertook the interviews. Each lasted approximately one hour and they were conducted at the school of the respective teacher or online using video conferencing tools. Guidelines for the interview were developed, focusing mainly on two themes (in addition to a set of background questions):

(1) Didactic practices and strategies employed when teaching programming, including a focus on the tools, languages and environments used.
(2) Pupils' knowledge and skills development when working with programming in K-12 education.

The questions we asked the respondents regarding the second theme were partly informed by the framework of Brennan and Resnick (2012). That is, we specifically asked the teachers if they could see if pupils developed skills related to computational/programming concepts and gave examples such as loops and variables. We also asked what type of new practices that pupils learn when working with programming in school.

The interviews were recorded with the consent of the interviewees and were later fully transcribed.

### Data analysis

When the interviews had been transcribed, we used thematic analysis procedures (Braun & Clarke, 2006) to analyse the data. Thematic analysis is proposed as a flexible method for identifying, analysing and reporting patterns (i.e. themes) within data (Braun & Clarke, 2006). The transcripts were coded based on two schemes: one focusing on the didactic strategies used in teaching programming and the other one with the aim of identifying knowledge and skills that pupils develop as they engage in programming activities. In this paper, we report on the latter coding. We used the following six-phase procedure based on Braun and Clarke (2006):

(1) Familiarizing oneself with the data
(2) Generating initial codes
(3) Searching for themes
(4) Reviewing themes
(5) Defining and naming themes
(6) Producing the report

The thematic analysis resulted in the identification of a total of 7 themes and 25 categories of skills that K-9 pupils develop when learning to program, according to the teachers interviewed.

## Results

Our analysis led us to identify a number of themes. Three of the themes identified during the analysis corresponded well with the dimensions in Brennan and Resnick (2012) framework, namely computational concepts, computational practices and computational perspectives; these were thus named in the same way (Table 1). The remaining four themes identified were as follows: (1) cognitive skills and attitudes, (2) language skills, (3) creative problem-solving skills and attitudes and (4) collaborative skills and attitudes (see Table 2). These were not found in the framework developed by Brennan and Resnick (2012).

Table 2. Themes and categories of general skills and attitudes developed when learning programming.

| Cognitive skills and attitudes | Language skills | Creative problem-solving skills and attitudes | Collaborative skills and attitudes |
|---|---|---|---|
| Logical thinking skills | Programming language comprehension | Problem-solving skills | Collaborative problem-solving skills |
| Symbolic/abstract thinking skills | Syntax (programming) | Creativity/fantasy | Pedagogic communication skills (being able to explain) |
| Accuracy | Compressing code | Remixing | |
| Courage to make mistakes and test things | Vocabulary | | |
| Patience/persistence | | | |

## *Computational concepts developed when learning programming*

According to the teachers, when working with *analogue programming*, pupils develop an understanding of computational concepts such as sequences, stepwise instructions, loops and conditionals. The following interview excerpt describes an analogue programming activity:

> *"When I want to advance things, I let the pupils individually write the entire code for the Lego figure's route to the goal using symbols they choose themselves. The only thing I give them is two kinds of statements they must use. 'If' statements: if the Lego figure reaches an obstacle, it should turn in another direction. And 'loops', so for instance, go straight forward, loop 10 times. When they have written the code, they should test it and refine it and re-test it and so on until the code does what it is supposed to do."* (Interviewee 1)

The set of computational concepts developed working with *RoBots* such as BeeBot or BlueBot is somewhat smaller as it is limited to sequences and stepwise instructions. However, these types of activities are often enacted by the teachers when introducing programming to the youngest pupils to – as one of the interviewees expressed it – *"fosters pupils' feeling and understanding of stepwise instructions and the idea that they actually can program a robot … it makes them motivated to work with programming I think"* (Interviewee).

Finally, when asking specifically about the computational concepts explored and developed by pupils when working with *block programming* languages such as Scratch, or in the few cases of *text-based programming*, all of the computational concepts listed in Table 3 were mentioned. Among these concepts, those most frequently mentioned by the teachers were sequences, loops and conditionals. Nevertheless, a few examples of variable use were mentioned, for instance, pupils constructing Scratch games and storing scores in a variable or creating Celsius to Fahrenheit converters that store the temperature in a variable.

Relating back to the computational concepts in the framework of Brennan and Resnick (2012), the concepts of parallelism and operators were not at all emphasized by the teachers in the interviews. In general, we could conclude that a majority of the teachers did not have the language to express the formal name of the computational concepts, albeit they explored them with their pupils. Some of these teachers were not

Table 3. CT skills developed.

| Computational concepts | Computational practices | Computational perspectives |
|---|---|---|
| Sequences, algorithms and stepwise instructions | Abstracting and modularizing | Expressing oneself |
| Loops | Reusing code | Connecting by sharing and building on others' work |
| Events | Testing and debugging | |
| Data (variables) | Being incremental and iterative | |
| Conditionals | | |

aware of the computational concepts at all and did not know that they were important in programming languages.

## *Computational practice skills developed when learning programming*

Regarding the practices pupils develop when they engage in programming activities and work with computational concepts, the teachers mentioned three practices: (1) *abstracting and modularizing*; (2) *reusing code*; (3) *testing and debugging*. The first, abstracting and modularizing, was mainly emphasized by the few teachers who worked with text-based programming. For instance, one teacher explained how he approached the concept of functions and abstraction:

> *"I'm just showing them … okay instead of writing a big loop, try to write functions that make things, one called measure and one called right and one called left, and then you can call them when needed. They can do this."* (Interviewee 2)

While it is quite common to modularize code in Scratch by creating different scripts for different kinds of events and behaviors, this was not highlighted by any of the teachers. However, the practice of reusing code was quite commonly referred to and was associated with all didactic strategies/levels except for analogue programming. One teacher emphasized the development of this practice quite clearly:

> *"Well they start to understand quite soon that it is okay to build on others' code. Actually that coding is all about reusing code instead of reinventing the wheel. So when working with BlueBots, they are recoding each other's codes and building on them and they continue with this in Scratch also; they kind of remix code all the time."* (Interviewee 3)

Some teachers viewed code reuse as necessary for pupils who needed more support and for pupils slower than others to grasp how to program themselves.

Finally, testing and debugging code is also common practice. It is explored and developed when employing all didactic strategies/levels from, for instance, doing analogue programming with pupils giving each other instructions to text-based programming in JavaScript. Several of the teachers expressed and emphasized a positive attitude towards the development of this practice, claiming, for instance, that *"they learn a lot by making mistakes and correcting them"* and *"they dare to try more things and thus learn more things when it is standard procedure so to say to make mistakes, debug and correct things iteratively"*. As can be seen from the last excerpt, the practice of working incrementally and iteratively is also mentioned.

### *Computational perspective skills developed when learning programming*

Computational perspective is the final dimension in Brennan and Resnick (2012) framework. When analysing the interviews, we found two such perspectives, namely *expressing oneself* and *connecting by sharing and building on others' work*. With regard to the former, teachers, especially those working with Scratch, emphasized that programming education is one of the school platforms that allows pupils to express themselves creatively:

> *"It's fantastic to see all these creations they make [Scratch] and how they reflect themselves and things that they value. It's like looking at a painting and trying to understand the painter's intentions and identity, what he or she expresses with the painting. It's the same here; they are expressing themselves in different ways in Scratch."* (Interviewee 4)

Sharing and building on work made by others was part of many teachers' strategies; they let their pupils remix a program the teachers had prepared themselves, modify programs available online or even programs made by their peers. In doing so, the pupils *"build an attitude that sharing is caring when doing programming and that programming is all about collaborating and working with codes others have made"* (Interviewee 5).

### *General skills and attitudes developed when learning programming*

In addition to the CT skills previously mentioned, teachers also found that teaching programming supports the development of some general skills and attitudes. Our thematic analysis of these skills and attitudes resulted in the identification of four dimensions, namely *cognitive skills and attitudes, language skills, collaborative skills and attitudes* and *creative problem-solving skills and attitudes* (see Table 3).

#### *Cognitive skills and attitudes*

The teachers highlighted some general cognitive skills and attitudes. For instance, *logical thinking skills* were commonly highlighted as an outcome of teaching programming, regardless of the didactic strategy employed. Many teachers described logical thinking as an integral aspect of programming. One of them stated:

> *"For me, programming is one way of logical thinking, so whether they are giving each other instructions in an analogue programming activity, programming the BeeBots or creating a game in Scratch, they are doing it in logical steps and through logical thinking. It makes them more logical I think. They are becoming robots themselves in a way* [laughter]*."* (Interviewee 4)

The teachers also emphasized that working with programming entails working with symbols and the development of *symbolic/abstract thinking skills*. Interviewee 4 highlighted the progressive work with symbols and abstract thinking:

> *"I also believe that they [pupils] learn to work with symbols when they are introduced to analogue programming with paper and pen. Then we work much more with symbols when we work with navigation, for example when we program the BeeBots or play LightBot. And the blocks we use in Scratch are kind of more advanced symbols that do different things. Next year, we will program MicroBits using a real programming language and that is completely symbolic and abstract. So yes, abstract thinking also, I think."* (Interviewee 4)

Besides the aforementioned skills, the teachers also emphasized that teaching programming results in pupils being *"trained to work and value accuracy because if something very small is wrong in the code, it will not work"* (Interviewee 5). Furthermore, several teachers shed light on the development of two interrelated skills/attitudes, namely *"courage to make mistakes and test things"* and the development of *patience* and greater *persistence* among pupils in the pursuit of their goals when working with programming, as clearly emphasized by one of the teachers:

> *"I sometimes hear or they tell me that they give up too often … the youngsters today … I do not really agree, because here they definitely do not give up. And that's something we've seen during social science lessons. When they are going to search for facts and it may be the case that they say I can not find this. Okay, then we're thinking about how we do programming, think that you are in the programming class. How did you do then? Then I did this or that* [pretends to be a pupil]. *But you can try to test it during social science* [teacher replies]. *Good idea!* [teacher replies as a pupil]. *Then you see that a light has been lit."* (Interviewee 5)

The above excerpt illustrates how a teacher perceives and fosters a transfer effect; the skill or attitude employed and acquired when learning programming is also used by pupils in other subjects. However, as two teachers emphasized, we have to be aware that not all children adapt or easily develop patience when working with programming. Instead, they might give up quickly if support is not provided by classmates or the teacher.

### *Language skills*

Programming, mainly when using blocks or text-based instructions, means working with language and hence results, as highlighted by the teachers, in the development of different kinds of language-related skills. For instance, a couple of teachers underscored that pupils develop *programming language comprehension skills*. For example, *"pupils develop the ability to read and understand code"* in specific programming languages, as well as being able to *"understand that there are different programming languages"* and *"get a feeling"* for their individual characteristics, their similarities and dissimilarities. The latter is expressed by one of the teachers in the following excerpt:

> *"I believe that they [pupils] get a feeling for the similarities between Scratch code and Python code and also the differences between the two ways of coding. I try to help them see that at least. I can say: Do you remember how you repeated the instruction in Scratch? In Python, you write 'a' for loop and you write it in text but it is the same thing."* (Interviewee 5)

Then, related to language comprehension, teachers also mentioned that the pupils develop writing skills and more specifically *syntactic skills* (i.e. being able to program/code with correct syntax) and the skill to *write code effectively* (i.e. being able, for instance, to write more compact code). Besides these programming language-related skills, several teachers also mentioned that the pupils develop their English vocabulary, which was appreciated both by the teachers and the pupils themselves.

### *Creative problem-solving skills and attitudes*

This theme included three categories: *problem-solving skills; creativity/fantasy*; skills for *remixing code*. The relation between problem-solving and programming was quite clear among the teachers. Indeed, all of them emphasized that teaching programming supports pupils' development of problem-solving skills. When asking about the specific problem-solving skills that are developed, a majority of the teachers mentioned the ability to understand the problem, to divide a problem into sub-problems/parts and to use debugging as a strategy to support the problem-solving process. A couple of teachers also emphasized the creative aspect of programming and problem solving, for instance mentioning reasoning skills regarding "thinking outside of the box" and that programming is strongly associated with creativity and fantasy:

> *"The fun thing with programming, yes at least when we use Scratch for example, is that it's so creative on so many levels. They [pupils] practise thinking outside of the box, they practise using their fantasy to create stories or games, yeah the fact that they create things that they imagine and that they can play or interact with the creations, see them come to life, that is quite cool to see."* (Interviewee 7)

Some teachers mentioned that younger pupils also develop creative problem-solving skills while engaging in, for example, programming Bluebots *"even though it is not needed according to the course syllabus"*. Another creative problem-solving skill mentioned was the ability to *"creatively remix existing code into something meaningful"*, as put by one of the teachers, which is likely a logical consequence of the didactic strategy to let pupils learn programming by modifying and remixing code; a strategy employed by many of the teachers.

### *Collaborative skills and attitudes*

Our analysis led us to identify three collaborative skills and attitudes that are developed by pupils according to the teachers interviewed, namely *collaborative problem-solving skills, pedagogic communication skills* and *sharing and building on others' work*. When interviewing the teachers, it became evident that a majority of them, in fact, all of them at some point, utilized the strategy of letting pupils work collaboratively on programming tasks of various kinds and with different tools and environments. Consequently, several of the teachers highlighted collaborative problem-solving skills as an outcome of the collaborative activities. When elaborating on and specifying these skills, the teachers mentioned, for instance, the ability to divide and work on different parts of a programming project, being able to assemble different parts into a whole together and being able to communicate effectively during the process. Several of the teachers explained the motivation for the strategy to focus on peer collaboration in the classroom by pointing to their own limited knowledge of both programming and how to teach programming. Several of them also acknowledged that pupils sometimes had more knowledge and acted as supportive pedagogues in the classroom; consequently, as explained by the following teacher, they developed pedagogic communication skills:

> *"I don't have any formal training in programming and honestly my knowledge is quite limited, but I am interested in the subject and try to make it work and it kind of works I would say and much because that the pupils learn together and help me in the classroom. And I learn together with them. There is, for example, a child who has a parent who is a programmer and his knowledge of programming in Scratch is much richer than mine so he helps me a lot in the classroom, explaining things to me and to his peers, and other pupils do that too. It's quite fascinating actually that they participate in teaching the subject and they practise providing explanations that help others. It gives them more confidence … yeah they are shining at times as heroes."* (Interviewee 6)

In the above excerpt, the teacher highlights more explicitly that pupils develop their ability to explain things to peers and teachers pedagogically.

## Discussion

In recent years, teachers around the world have started teaching programming at the K-9 level, some due to the formal introduction of programming in the national curriculum, others without such pressure and on their own initiative. In this study, we attempted to understand which skills – both CT-related and general – are developed in the process. To do so, we interviewed 19 teachers who had been teaching programming for a couple of years on their own initiative and because of interest in the topic. The teachers were selected based on their experience in teaching programming. Our thematic analysis of these interviews shed light on what skills teachers feel pupils develop when programming. This led us to identify three themes related to CT skills and five themes related to general skills.

### *Computational thinking skills*

The CT skills identified corresponded well with and were thus thematically structured according to the dimensions/themes of CT proposed in the framework of Brennan and Resnick (2012), namely *computational concepts, computational practices* and *computational perspectives*. For instance, the teachers highlighted that pupils develop an understanding of fundamental computational concepts, such as algorithms, variables, loops and conditionals, when exposed to activities on all four programming levels, i.e. analogue programming, programming with RoBots, block programming and text-based programming (Nouri et al., 2018). According to the teachers, these activities also support pupils in developing skills related to fundamental computational practices, such as abstracting/modularizing, reusing code and debugging code, as well as skills related to computational perspectives, such as expressing oneself and connecting with others by sharing and building on others' work.

However, some skills presented in the Brennan and Resnick (2012) framework were not highlighted by the teachers, such as those related to the computational concepts of events, parallelism and operators. One of the reasons for this, according to our observations and teachers' comments, is that the teachers lack knowledge about these concepts in the first place. Indeed, due to the lack of formal education in programming, a majority of the teachers were not familiar with basic programming concepts such as conditionals and data (variables), albeit they had worked with these concepts and with the support of our interview questions could conclude that pupils develop skills associated with these concepts.

Based on this, some conclusions can be drawn. First, by working with the didactic strategies mentioned by the teachers, i.e. analogue programming, programming with RoBots, block programming and text-based programming, enacted in the way the teachers reported (see Nouri et al., 2018), pupils actually develop meaningful CT skills that the teachers either might not possess fully themselves or that they are not aware of. We, therefore, with some certainty, determine that there is a rather symmetric knowledge relation, at least in comparison with traditional subjects, between teachers and pupils regarding the CT skills developed. Second, we conclude that, due to their limited knowledge and to the fact that the formulations in the Swedish curriculum do not contain requirements of teaching CT as an isolated subject, teachers' didactic strategies lack an explicit focus on the development of CT skills among the pupils and therefore also focused assessment of these skills. A first step forward would be to develop teachers' CT skills through professional development programmes, such as those in which many countries, Sweden included, have invested and planned for. Albeit teachers are evidently creating meaningful programming experiences for pupils without suitable professional development, teachers' current knowledge and skill levels together with the current formulation of the Swedish curriculum, will put boundaries on how programming instruction is didactically orchestrated and consequently the skills and knowledge pupils might be able to develop in the classroom.

## General skills and attitudes developed and the relation to digital competency and 21st century skills

In addition to the CT skills, our thematic analysis of the interviews conducted also resulted in the identification of four themes concerning general skills and attitudes, namely *cognitive skills and attitudes, language skills, collaborative skills and attitudes* and *creative problem-solving skills and attitudes*. In contrast to CT skills, these themes and categories of skills and attitudes were much easier for the teachers to identify and verbalize. Some of these deserve more discussion. For instance, creative problem-solving skills were emphasized by all the teachers interviewed as a central outcome of programming education. While problem-solving skills have been practised in schools before in subjects such as mathematics, programming instruction seems to create more opportunities for pupils to engage playfully in creative problem-solving activities. As teachers over time and with increased experience – and through professional development initiatives – become more aware of the creative problem-solving processes that are involved in programming practices, they will become even more equipped to foster and enhance such skills in their pupils.

Partly due to the integral collaborative aspects of programming practices and partly due to teachers' limited knowledge, pupils also develop skills in solving problems collaboratively and pedagogically communicating and helping their peers (and teachers), as indicated in the interviews. This development is interesting as it sheds light on pupils' participation in the teaching activities of the topic and suggests a shift in the roles and agencies of teachers and pupils in the classroom. It remains to be seen what happens with their respective roles when teachers' knowledge and skill levels increase.

Interestingly, collaborative problem-solving skills, programming language literacy, creative problem-solving skills and the skills to use digital means for self-expression are also central elements of 21st century skills (Ahonen & Kinnunen, 2015; Ananiadou & Claro, 2009: Dede, 2009; Griffin & Care, 2015; Van Laar, van Deursen, van Dijk, & de Haan, 2017) and digital competence (Ala-Mutka, 2011; Erstad, 2008; Eshet, 2004). For instance, in the conceptual framework of Ala-Mutka (2011), digital competence includes advanced skills for the creation of material, problem solving, collaboration and innovation – skills that correspond well with the themes of skills we identified in this study. When looking more closely at the specific digital competences mentioned in the literature, we find, for example, the specific skill of creatively remixing, which in our thematic analysis is a subcategory of the theme creative problem-solving skills. Erstad (2008) describes this skill category as an "essential part of digital literacy" that represents a "change in our schools today, from knowledge development being based on predefined content in school books and the reproduction of knowledge provided by the teacher, towards a situation where students take available content and create something new, something not predefined" (p. 178). Also, elements such as collaboration and perseverance are listed as approaches integral to CT in the Barefoot CAS framework (https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking/), which presents CT as a collection of six concepts and five approaches.

Thus, we conclude that programming education in K-9, when considering teachers' assessment of it, not only develops CT skills that to a great extent have been the primary objective when introducing programming curricula (Brennan & Resnick, 2012; Heintz, Mannila, Nygårds, Parnes, & Regnell, 2015), but also fosters skills and attitudes of more general character that are strongly associated with 21st century skills and digital competence/literacy, which have not been explicitly aimed for. We do, however, need to keep in mind that the skills identified in this study reflect: (1) the didactic practices and strategies used by the teachers; (2) the teachers' limited knowledge of programming and the teaching/didactics of programming (including CT); (3) that the Swedish curriculum primarily view programming as an instrument for learning other subjects such as mathematics, and not as an subject to be learned and formally assessed; which have the consequence (4) that the majority of the teachers have not actively and deliberately worked towards fostering the skills and attitudes identified. Thus, with time and as teachers' knowledge of programming and the didactics of programming increases, future work will be required that adds skills to the list identified, based on both teachers' perceptions and direct assessments of pupils' knowledge and skill development when learning programming in K-9.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## Notes on contributors

*Jalal Nouri* is an associate professor at the Department of Computer and System Sciences, Stockholm University. Nouri conducts research in the field of technology enhanced learning and have a particular interest in computing education, learning analytics and digital competency.

*Lechen Zhang* is a PhD student at the Department of Computer and System Sciences, Stockholm University. Zhang's research focus on the development of computational thinking in K-9 education.

*Linda Mannila* is a senior lecturer at the Department of Computer and Information Science, Linköping University. Mannila's research interests are focused on computing education and questions related to digital competency.

*Eva Norén* is a senior lecturer at the Department of Mathematics and Science Education, Stockholm University. Noréns interests are focused on mathematics education and programming within the frame of mathematics education.

## ORCID

Jalal Nouri http://orcid.org/0000-0002-9942-8730
Lechen Zhang http://orcid.org/0000-0002-4216-4463
Eva Norén http://orcid.org/0000-0002-6099-7426

## References

Ahonen, A. K., & Kinnunen, P. (2015). How do students value the importance of twenty-first century skills? *Scandinavian Journal of Educational Research*, *59*(4), 395–412.

Ala-Mutka, K. (2011). *Mapping digital competence: Towards a conceptual understanding*. Sevilla: Institute for Prospective Technological Studies.

Ananiadou, K., & Claro, M. (2009). *21st century skills and competences for new millennium learners in OECD Countries* (OECD Education Working Papers, No. 41). Paris: OECD Publishing.

Australian Curriculum Assessment and Reporting Authority. (2015). *The new Australian Curriculum: pathway to ICT success*. Retrieved from http://docs.acara.edu.au/resources/20151201_Response_to_ICT_article.pdf

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101.

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

Carter, E. (2015). Its debug: Practical results. *Journal of Computing Sciences in Colleges*, *30*(3), 9–15.

Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress. Retrieved from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Dede, C. (2009). *Comparing frameworks for "21st century skills"*. Boston, MA: Harvard Graduate School of Education.

Erstad, O. (2008). The digital literacy in schools: The Norwegian experience. *Italian Journal of Technology*, *16*(1), 4.

Eshet, Y. (2004). Digital literacy: A conceptual framework for survival skills in the digital era. *Journal of Educational Multimedia and Hypermedia*, *13*(1), 93–106.

Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning*, *32*(6), 576–593.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97.

Fields, D., Vasudevan, V., & Kafai, Y. B. (2015). The programmers' collective: Fostering participatory culture by making music videos in a high school Scratch coding workshop. *Interactive Learning Environments*, *23*(5), 613–633.

Griffin, P., & Care, E. (2015). *Assessment and teaching of 21st century skills – Methods and approach*. Dordrecht, Heidelberg, New York, London: Springer.

Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 267–272). Seattle, Washington: ACM.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

Heintz, F., Mannila, L., & Färnqvist, T. (2016, October). A review of models for introducing computational thinking, computer science and computing in K-12 education. In *Frontiers in Education* Conference *(FIE)* 2016 (pp. 1–9). Pennsylvania, US: IEEE.

Heintz, F., Mannila, L., Nygårds, K., Parnes, P., & Regnell, B. (2015). Computing at school in Sweden - Experiences from introducing computer science within existing subjects. In *8th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives I Informatics in Schools. Curricula, Competences, and Competitions/Lecture Notes in Computer Science and General Issues, 9378* (pp. 118–130). Saint-Petersburg, Russia.

Hermans, F., & Aivaloglou, E. (2017, November). To scratch or not to scratch? A controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 49–56). Nijmegen, The Netherlands: ACM.

Informatics Europe and ACM Europe. (2015). *Informatics in education: Europe cannot afford to miss the boat*. Report of the Joint Informatics Europe and ACM Europe Working Group on Informatics Education. Retrieved from http://europe.acm.org/iereport/ACMandIEreport.pdf

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*(3), 583.

Kong, S. C. (2016). A framework of curriculum design for computational thinking development in K-12 education. *Journal of Computers in Education*, *3*(4), 377–394.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61.

Nouri, J., Norén, E., & Skog, K. (2018). Didactical strategies employed by teachers when teaching programming in k-9 education. In Didactical strategies employed by teachers when teaching programming in k-9 education Technology, Education and Development Conference. pp. 7983-7989. doi:10.21125/inted.2018.1915

Opetushallitus 2014. Perusopetuksen opetussuunnitelman perusteet 2014. Määräykset ja ohjeet 2014:96. Helsinki

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

Resnick, L. B. (1987). *Education and learning to think*. Washington, DC: National Academy Press.

Rose, S., Habgood, J., & Jay, T. (2017). An exploration of the role of visual programming tools in the development of young children's computational thinking. *Electronic Journal of E-Learning*, *15*(4), 297–309.

Sanders, K., & McCartney, R. (2016). Threshold concepts in computing: Past, present, and future. *Proceedings of the 16th Koli Calling international conference on computing education research*, Finland.

Sands, P., Yadav, A., & Good, J. (2018). Computational thinking in K-12: In-service teacher perceptions of computational thinking. In Khine M. (Ed.). *Computational thinking in the STEM disciplines* (pp. 151–164). Cham: Springer.

Selby, C., & Woollard, J. (2013). Computational thinking: The developing definition. Retrieved from http://eprints.soton.ac.uk/356481

Sjöberg, C., Risberg, T., Nouri, J., Noren, E., & Zhang, L. (2019). A lesson study on programming as an instrument to learn mathematics and social science in primary school. In Proceedings of the 13th International Technology, Education and Development Conference (pp.2230–223) 11–13 March, 2019. Valencia, Spain.

Skolverket. (2017). *Få syn på digitaliseringen på grundskolenivå*. Commentary material. Retrieved from https://www.skolverket.se/publikationer?id=3783

Sullivan, A., & Bers, M. U. (2016). Robotics in the early childhood classroom: Learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade. *International Journal of Technology and Design Education*, *26*(1), 3–20.

Van Laar, E., van Deursen, A. J., van Dijk, J. A., & de Haan, J. (2017). The relation between 21st-century skills and digital skills: A systematic literature review. *Computers in Human Behavior*, *72*, 577–588.

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*(4), 715–728.

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.

Žanko, Ž., Mladenović, M., & Boljat, I. (2019). Misconceptions about variables at the K-12 level. Educ Inf Technol 24 (2), 1251–1268. DOI: https://doi.org/10.1007/s10639-018-9824-11–18.