Renate Andersen
Oslo Metropolitan University

DOI: https://doi.org/10.5617/adno.9169

Block-based programming and computational thinking in a collaborative setting: A case study of integrating programming into a maths subject

Abstract

Block-based programming and computational thinking (CT) have undergone a revival in K–12 education, and the article addresses the following research question: In what ways can block-based programming be integrated into a maths subject, and what are the implications for CT? The empirical data presented in this article are derived from a design-based research (DBR) project consisting of four interventions each year over a duration of two years. The article reflects data from 43 pupils aged 12–16 years who participated in the interventions and used MakeCode (a block-based programming language) with micro:bit to create solutions for maths tasks assigned by the teachers. The pupils met three hours per week for 16 weeks during two semesters each year. Data were collected using video recordings of Zoom meetings. A thematic analysis was performed in the first rounds of analysing the data to complete an overview of the entire data set, screening for common topics. Subsequently, interaction analysis was used to analyse select parts of the data in detail. The main findings in this article are as follows: 1) Integrating block-based programming into a maths subject enabled active and collaborative learning, 2) integrating programming into a maths subject enabled the development of CT, and 3) using block-based programming facilitated learning of maths.

Keywords: block-based programming, computational thinking, computer-supported collaborative learning, computational thinking in K–12 education

Blokkbasert programmering og algoritmisk tenkning i en samarbeidslæringskontekst: En case-studie av programmering integrert i et matematikkfag

Sammendrag

Blokkbasert programmering og algoritmisk tenkning har fått økt interesse i skolen. Følgende forskningsspørsmål tas opp i artikkelen: På hvilke måter kan blokkbasert programmering integreres i et matematikkfag, og hva er implikasjonene for algoritmisk tenkning? De empiriske dataene som presenteres i denne artikkelen er hentet fra et designbasert forskningsprosjekt bestående av fire intervensjoner hvert år over en varighet på to år. Denne artikkelen reflekterer data fra 43 elever i alderen 12–16 år som deltok i intervensjonene og brukte MakeCode (et blokkbasert programmeringsspråk) med micro:bit for å lage løsninger for matematikkoppgaver gitt av lærerne. Elevene

møttes tre timer per uke i 16 uker i løpet av to semestre hvert år. Data ble samlet inn ved hjelp av videoopptak av Zoom-møter. En tematisk analyse ble utført i de første rundene med å analysere dataene for å få en oversikt over hele datasettet. Deretter ble interaksjonsanalyse brukt for å analysere utvalgte deler av dataene i detalj. Hovedfunnene i artikkelen er som følger: 1) Integrering av blokkbasert programmering i et matematikkfag muliggjorde aktiv samarbeidslæring, 2) integrering av programmering i et matematikkfag muliggjorde utviklingen av algoritmisk tenkning, og 3) bruk av blokkbasert programmering la til rette for læring av matematikk.

Nøkkelord: blokkbasert programmering, algoritmisk tenkning, datastøttet samarbeidslæring, algoritmisk tenkning i skolen

Introduction

Programming and computational thinking (CT) are regarded as fundamental skills in the 21st century (Wing, 2006; Weintrop et al., 2016; Boccconi et al., 2016; Yadav et al., 2017; Mohaghegh & McCauley, 2016; Zhang & Nouri, 2019). Inculcating pupils with skills in CT and programming is important in order to prepare them for future work and enable them to reflect on challenges and opportunities created by the technology we use. Due to our increasingly information-based and technology-rich society it is important to ensure that students develop CT at the K–12 education level. K–12 education refers to pupils in school grades prior to college. It is necessary to provide teachers with knowledge about CT and how to incorporate it into their teaching (Yadav et al., 2014), making it crucial to conduct research that focuses on how to integrate CT and programming into K–12 education. In the Nordic countries, it has been suggested that CT and programming should be part of a more comprehensive 21st century skills approach (Bocconi et al., 2016). 21st century skills are defined as encompassing creativity, critical thinking, and problem solving, as well as other skills students should develop to act as creators of knowledge rather than passive consumers in the classroom (Gretter & Yadav, 2016).

CT was first coined by Papert (1980) as the relationship between programming and thinking skills. Building on this, Wing (2006) defined CT as solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science. However, there is not one agreed-upon definition of CT. Shute et al. (2017) define CT as the conceptual ingredient required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are applicable in different contexts. In a school context, CT is viewed as extending computer science principles to other disciplines to help break down the elements of any problem, determine their relationship to each other, and devise algorithms to arrive at an automated solution (Kafai & Burke, 2013). What all these definitions have in common is that they perceive CT as transcending programming; it also

entails problem-solving processes. CT can be regarded as an investigative and problem-solving method that utilises computer science concepts, tools, and techniques in science, technology, engineering, and mathematics (STEM) (Lee et al., 2020). This article adopts the view of CT as encompassing both a programming perspective and a problem-solving perspective. The connection between programming and CT is that through programming, pupils develop computational thinking. Lye and Koh (2014) underscore that during the process of programming, pupils are exposed to CT, which involves the use of computer concepts.

CT has been included in school curricula in many countries. In recent years, the Nordic countries have all introduced CT into their curricula; however, different countries have interpreted and implemented CT in different ways (Bocconi et al., 2016; Heintz et al., 2016; Zhang & Nouri, 2019). Bocconi et al. (2016) emphasise that CT and programming have been advocated by educational stakeholders as skills that all people should learn, as fundamental as numeracy and literacy. Although CT and related concepts (programming, coding, algorithmic thinking) have received increasing attention in the educational field, there is a dearth of research on the successful integration of CT in compulsory education, and there are a plethora of unresolved issues and challenges (Bocconi et al., 2016). Programming skills have become important core competencies for 21st century skills, particularly in education; as a result, many countries have recognised that programming needs to be integrated into school curricula to equip pupils with these skills, such as problem-solving and logical thinking (Forsström & Kaufmann, 2018). In recent years there has been a new interest in reviving learning and teaching programming at all K–12 levels (Kafai & Burke, 2013). The focus in this article is on exploring CT as an integrated part of a school subject. In autumn 2020, Norway implemented a new curriculum for K–12 education wherein programming and CT were integrated into selected subjects: maths, natural science, music, and arts and crafts. This warrants research into the implications of introducing CT and programming into the new curriculum to uncover how it works in practice.

The integration of CT into STEM subjects is relatively new, and CT is regarded as having the potential to deepen STEM learning by enabling pupils to act as young scientists through engagement in authentic STEM practices (Lee et al., 2020). Consequently, there is a need for research on the implications hereof. The research question addressed in this article is:

> *In what ways can block-based programming be integrated into a maths subject, and what are the implications for computational thinking?*

The focus of this article is K–12 education. A maths subject is the context for the study, which explores the processes that emerge during block-based programming and the implications for computational thinking.

Related literature

This section presents literature relevant to CT in K–12 education, block-based programming and CT in K–12 education, and computer-supported collaborative learning (CSCL) and CT.

Studies related to CT in K–12 education

Several researchers have elucidated what CT entails and suggested frameworks for how to understand it (Weintrop et al., 2016; Heintz et al., 2016; Ioannidou et al., 2011; Brennan & Resnick, 2012). Other studies have reported how CT can be facilitated using block-based programming languages (Dwyer et al., 2015; Moreno-León et al., 2016), and some studies have focused on the integration of CT in STEM subjects (Lye & Koh, 2014; Lee et al., 2020). Scrutinising how CT transcends programming and exploring how to redefine it as a thinking process is an increasing development (Lodi & Martini, 2021; Lee et al., 2020). Several reviews provide overviews of CT in K–12 education (Grover & Pea, 2013; Shute et al., 2017; Heintz et al., 2016; Lye & Koh, 2014) and in K–9 education (Nouri et al., 2020; Heintz & Mannila, 2018; Flórez et al., 2017). Several studies focus on exploring how CT can be integrated in K–12 STEM education (Kong et al., 2019; Lee et al., 2020), but there is a lack of empirical case studies exploring how CT can be integrated into K–12 education and what the implications are.

Studies related to block-based programming and CT in K–12 education

Block-based programming is a visual programming language that takes the form of dragging and dropping programming instructions together as it uses a programming-primitive-as-a-puzzle-piece metaphor to provide visual cues to the user as to how and where commands may be used (Weintrop, 2019). One of the advances of block-based programming is that it reduces pupils' challenges in learning the language syntax when programming (compared to text-based programming) and renders programming more accessible to novices (Sengupta et al., 2013). Some studies have reported how CT can be facilitated through block-based programming (Weintrop, 2019; Sengupta et al., 2013). In a study by Weintrop et al. (2016), the researchers developed a taxonomy of CT practices for maths and science consisting of data practices (collecting, creating, manipulating, analysing, and visualising data), modelling and simulation practices, computational problem-solving practices, and systems thinking practices. Sengupta et al. (2013) emphasise that arguments favouring the integration of CT and programming into K–12 STEM curricula have been suggested; they present a critical review of educational computing and propose a set of guidelines for designing learning environments focusing on science topics that foster the development of CT. The focus in this article is on block-based programming, since the data are drawn from the social interactions and collaborations between pupils when using the block-based programming language MakeCode.

Studies related to CSCL and CT

Computer-supported collaborative learning (CSCL) is an interdisciplinary research field which focuses on studying how people together can learn with the help of a computer; it emphasises the interplay between learning and technology (Stahl et al., 2006). In CSCL, the group is the unit of analysis. The underlying principle in CSCL is that collaboration is primarily conceptualised as a process of shared meaning construction, and the meaning making processes do not merely express mental representations of the individuals but also an interactional achievement (Stahl et al., 2006). CSCL studies are often divided between focusing on physical co-located studies or distributed studies. The focus here is on distributed CSCL, as the context for the empirical data derives from a distributed online setting. In one empirical study, being in a distributed CSCL context, the term "mutual development" was defined as a joint collaboration process wherein multiple stakeholders with different backgrounds co-created shared artifacts (Andersen & Mørch, 2009). In another study, the focus was on mutual development and the co-creation of artifacts at different levels of participation, as well as investigating collaborative knowledge creation processes mediated by online technologies (Andersen, 2019). Kafai (2016) underlines that CT and programming are social creative practices, and they offer a context involving making applications for others wherein sharing and collaboration are the premise. This article follows along these by framing CT and programming in a collaborative learning context, emphasising the group as a unit of analysis.

There are few research studies that focus on CT, programming, and CSCL. One study investigates how students are motivated to learn programming concepts in a CSCL setting (Serrano-Cámara et al., 2014); another study investigates how collaborative learning through pair programming can improve pupils' computational thinking skills (Echeverría et al., 2019). The present article uses CSCL in combination with CT and programming to explore and understand how pupils interact and collaborate when using block-based programming as an integrated part of learning maths when working in small groups. According to Stahl (2007), there is a need for empirical analysis of how meaning is constructed in small-group interactions. He underlines that meaning is created and shared through processes of interaction, communication, and coordination.

In summary, this review demonstrates that there is substantial research that defines, discusses, and presents frameworks for understanding CT in K–12 education. However, there are few empirical studies exploring how CT and programming are integrated into STEM subjects in K–12 education. Moreno-León et al. (2016) underscored this when stating that there is a lack of empirical studies that investigate how learning to program at an early age affects other school subjects. There is a need for research regarding the implications of introducing CT into K–12 education. This article addresses this gap by contributing with empirical research concerning how pupils collaborate in small groups when solving tasks

using block-based programming as an integrated component of a maths subject; it also examines the implications for CT.

Conceptual framework

The conceptual framework consists of two different research perspectives: CT and computer-supported collaborative learning (CSCL). The CT perspective is used to highlight the programming and CT elements, whereas CSCL is used to bring the collaborative components in these settings to the foreground. These perspectives are combined because, when used in combination, they provide a richer data analysis. From the CT perspective, Brennan and Resnick's (2012) CT framework is used, comprising: computational concept, computational practice, and conceptual perspective. From the CSCL perspective (Stahl, 2007), the concepts knowledge sharing, meaning making, and group cognition are presented. The connection between these two perspectives is that they represent different approaches to exploring the programming processes, and they are useful in combination because they complement each other by gaining insights into both the CT processes and the collaborative learning processes emerging in the programming. This provides a group interaction perspective on the programming processes and a group perspective on CT in K–12 education, which is novel. See Table 1 for an overview of the conceptual framework. The analytical concepts presented in the conceptual framework are used to analyse the empirical data.

Table 1. The conceptual framework of the article

Analytical concepts from CT (Brennan & Resnick, 2012)	**Analytical concepts from CSCL** (Stahl, 2007)
Computational concepts (sequences, loops, parallelism, events, conditionals, operators, and data)	*Knowledge sharing* (individual shares information with group)
Computational practices (being incremental and iterative, testing and debugging, reusing and remixing, abstracting and modularising)	*Meaning making* (collaborative construction of shared meaning)
Computational perspective (expressing, connecting, questioning)	*Group cognition* (a gradually evolving result of the group discourse)

Brennan and Resnick's framework of categorising CT (2012) divides it into three dimensions: a) the *concepts* designers engage with in programming, i.e., sequences, loops, parallelism, events, conditionals, operators, and data; b) the *practices* designers develop as they engage with the concepts, i.e., being incremental and iterative, testing and debugging, reusing and remixing, abstracting, and modularising; and c) the *perspectives* designers form about the world around them and about themselves, i.e., expressing, connecting, and questioning. This means that the computational concepts are concepts the programmers use, the computational practices are the problem-solving processes that emerge in the programming process, and the computational perspective concerns how the programmers and

learners perceive the world and themselves in it, including the technological aspect. From the perspective of CSCL, the concepts *knowledge sharing*, *meaning making*, and *group cognition* are presented. *Knowledge sharing* is when individuals communicate what they already know to the others and when the participants offer alternative views related to the discussion (Stahl, 2007). *Meaning making* is defined as shared meaning construction, in which *shared group meaning* is collaboratively created by the entire group; it is interactively achieved in a discourse and may not originate from any particular individual (Stahl, 2007). Processes of establishing shared meaning involve a process of *group cognition* of the shared construction of a meaning; this emerges through the participants building on each other's utterances, in which the individual group members have to interpret the meaning from their own personal perspectives, display their understanding of the meaning, and affirm that meaning as shared (Stahl, 2007).

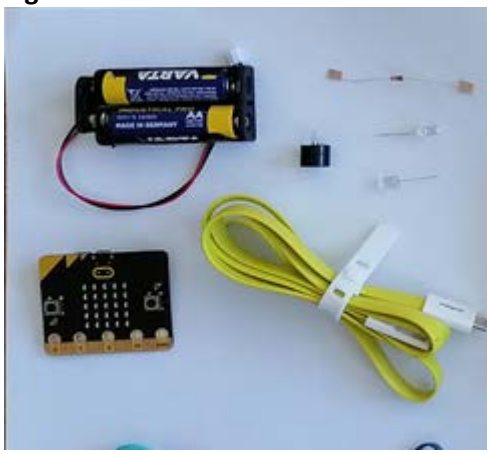
Methods

Research design

The empirical data in this article originate from an ongoing research project where the applied overall research design is *design-based research* (DBR). DBR is a research methodology used in an educational context that seeks to increase the impact, transfer, and translation of educational research into improved practice – also underlining the need for theory building and the development of design principles that guide and inform the practice (Barab & Squire, 2004). In DBR, teachers, researchers, and other participants design and create educational interventions through collaboration. DBR is also often referred to as "educational design research" which focuses on the iterative development of solutions to practical and complex educational problems in addition to offering the context for empirical investigation, which also may produce theoretical knowledge that can inform others (McKenney & Reeves, 2018). The research project focuses on creating and testing a series of eight learning designs that were used as interventions in classes consisting of gifted pupils. They focus on learning block-based programming as an integrated part of their course subject (maths). In total, four iterations each year were conducted over a two-year period. In each semester, two interventions are tested; they are evaluated based on feedback, and as such, the research design and the interventions are iteratively evolving. This means that the learning designs were modified and revised after each intervention as a result of feedback from the teachers and pupils. In sum, the interventions (learning designs) gradually evolved. The pupils met three hours per week for 16 weeks during two semesters each year. This reflects two interventions each semester (two semesters per year) which lasted for 8 weeks over a 2 year period. We followed the same pupils for one year at a time, which means we have two sets of different pupils in the dataset. The data used in this article are from the same year and therefore the

same set of pupils. In total, up to 200 gifted pupils between 12 and 16 years old participated in the project. They are divided into classes with around 20 participants in each class. This article reflects data from 43 of these pupils. The competence goals which are used as a premise for designing the learning designs in the intervention are 3 to 5 years above the pupils' age. Each class is taught by a high school teacher, and there are six participating high school teachers in the research project. Each of the interventions is organised to consume around three hours per week, with a duration of 8 weeks. The data presented in this article are derived from the second intervention, in which the topic of the learning design was probability. The interventions were structured as follows: First, the teacher introduced the topic for the intervention, and the teacher then divided the class into groups consisting of 3 to 4 pupils in different breakout rooms in Zoom[1]. The pupils used MakeCode as a block-based programming language and the subsequent micro:bit (a block-based programming language with similar syntax and structure as Scratch, see Figure 1) to solve the maths tasks they were assigned. In order to facilitate collaboration among the pupils, the teachers encouraged the pupils to share their screen in Zoom so that the others in the group could see their partial solution. This gave the students a common ground for further discussion regarding how to solve the task assigned by the teacher. As a result, the group was the unit of analysis.

Figure 1. A screenshot of the micro:bit the pupils use when solving the tasks in assignments



Selection of participants

Gifted pupils were selected as participants in the research project due to their need for differentiated learning in school. After receiving nominations from teachers and parents, the participants applied for participation in the research project.

Methods for collecting the data

Data were collected through video recordings of the pupils' screens using Zoom. As a result, a virtual ethnography approach was used, which included participant

[1] Due to the pandemic Covid 19 in 2020–2021. Zoom is a video communications app that allows participants to set up video conferencing (Zoom, 2021).

observation. Virtual ethnography is a method used for analysing social interactions in online contexts, and a technique for collecting data is to engage in the social interactions when collecting data (Hine, 2000). Participant observation implies that the pupils were sometimes asked to elaborate upon what they were doing during their group work. In total, there are 70 hours of screen recordings of interaction data between different pupils. All of the participants in the article have been anonymised, meaning that their names are fictive. The research project was reported to and approved by the National Center for Data services in Norway in order to reassure that the research project follows ethical guidelines and data protection laws.

Methods for analysing the data

A combination of thematic analysis (Braun & Clarke, 2006) and interaction analysis (Jordan & Henderson, 1995) was applied for analysing the empirical data to ensure a richer analysis, from an overview and from a more detailed examination of the data, respectively. Thematic analysis is defined as a qualitative method for systematically identifying and organising patterns of meanings across a data set. This enables the researcher to see and make sense of collective or shared meaning and experiences (Braun & Clarke, 2006). Examples of the thematic codes that emerged when screening the data are as follows: knowledge sharing, programming, computational concepts, and collaboration. Interaction analysis is defined as a method for the empirical investigation of interactions between human beings and the objects in their environments, including speech, non-verbal actions, and the use of artifacts (Jordan & Henderson, 1995). Consequently, the data were analysed using a two-step approach: 1) Thematic analysis provided an overview of the data set when screening the whole data set, looking for thematic codes and emerging patterns. 2) Interaction analysis was employed when zooming in on specific data and analysing it in detail.

Validity

A typical validity issue is anecdotilism, meaning that the researcher only presents a few well-chosen examples (Silverman, 2005). To overcome the issue of anecdotilism, a focus was placed on ensuring that the selected thematic codes were representative for at least 3 to 5 data excerpts to underscore that the thematic code represents a common issue in the data set and not merely a deviant case or a one-time issue.

Empirical data and analysis

Two data extracts representing different aspects of the social interactions emerging when pupils use block-based programming to solve maths tasks are presented: 1) programming the micro:bit to calculate probability, and 2) discussing how to fix a code that is not working via collaborative debugging.

Extract 1: Programming the micro:bit to calculate probability

Contextualising the extract

In the extract below, the pupils are assigned the task to program the micro:bit to create a dice and calculate the probability of getting the number "6" on the dice after rolling the dice a given number of times (for example, 10,000 dice rolls). The extract is derived from an intervention focusing on maths and consists of three pupils who collaborate on creating the code. The extract stems from the start of the group session, immediately after they have started discussing how to create the code.

Table 2. Data extract 1 (translated into English by the author)

Turn	Actor	Utterance	Analytical concept
1	Lucas	Now I will show you and try to explain what I did. When "Button A" is pressed we create a variable which we define as "denominator" – agree? Yes, then, we declare the variable "denominator" to 0 because then we have a starting point for that variable. Next, we create a loop which is repeated 10,000 times, and in connection to this we create another variable which we define as "the dice roll" and then we also declare it to 0. Next, we use a maths operator block which chooses a random number between 1–6. Did you also do it like this?	Knowledge sharing, computational concepts (loop and variable)
2	James	Yes, I guess we have the same code	
3	Noah	What result did you get as the probability?	Meaning-making (turns 3–10)
4	James	Wait a minute	
5	Lucas	0.16 or 0.17	
6	Noah	If the roll of the dice becomes 6, then I get the probability to be 1.05 percent.	
7	Lucas	Then it just is very unlikely (that the dice will roll and becomes 6). How many times did you program it to be repeated (that the dice would roll)?	Computational perspective, computational concept
8	Noah	1,500 times	
9	Lucas	Oh yes, ok.	
10	Lucas	Then, we go to the logics blocks and we take this block (if-else block) if the dice is rolled 1,500 times, then we increase the variable "denominator" with 1, like this. Have you also done it like this? And then "show number" (block) divided by the number of repetitions, which is 10,000, like this. Is this how you have done it?	Knowledge sharing, Computational concepts (conditionals, operators)
11	James	Yes, just that I have not programmed the last part yet. I have not calculated the probability in percent yet since I thought I did not know how to do it. Now, I will try to program the percent calculations as well, so maybe it will turn out to be correct.	Group cognition
12	Lucas	Shall we see, here we get (running the code in micro:bit) 0.17 – Yes!	Knowledge sharing, see Figures 2 and 3

Analysing the extract

The extract starts with Lucas in turn 1, explaining to the group how he programmed the code that calculates the probability of rolling the number "6" on the dice after 10,000 repetitions. This utterance has similarities to knowledge sharing (Stahl, 2007). Knowledge sharing is defined by Stahl (2007) as when individuals communicate what they already know to the others. In turn 1, we can also see instances of what Brennan and Resnick (2012) refer to as computational concepts. Examples of the computational concepts are "variable" and "loop." In turn 2, James responds that he has programmed the same code as Lucas in turn 1. However, Noah interrupts in turn 3 and wants to know the answer to the task (what is the probability?). This question triggers a discussion with similarities to a meaning-making process, wherein the different utterances by the pupils build on each other and forge a shared meaning construction (Stahl, 2007). In turns 3 through 10, the pupils discuss and build on each other's utterances, and a common understanding of how to program the code emerges. In processes of meaning-making, there is an uptake of the other pupils' utterances by posting a subsequent one, and through this continuation, the interactions and collaboration processes advance forward (Stahl, 2007). As a result, in turn 5, Lucas states his result when he runs the code and tests out how the micro:bit can be used as a calculator. Lucas gets a probability of 0.16 or 0.17, which is correct. However, in turn 6, Noah gets another result that is too low. In turn 10, there are several instances of computational concepts, such as conditionals (when talking about the "if-else block") and operators, and the pupils discuss the computational concepts. Conditionality is a key computational concept defined as the ability to make decisions based on certain conditions, which supports the expression of multiple outcomes (Brennan & Resnick, 2012). Additionally, when saying "show number" (block) divided by the number of repetitions, 10 is an example of using the operator as a computational concept. "Operator" is also a computational concept defined as providing support for mathematical, logical, and string expressions, enabling the programmer to perform numeric and string manipulations (Brennan & Resnick, 2012). In turn 11, James states that he has learned a new method of programming the percent calculations. Together, the students have discussed and formed a new solution, which they have created in collaboration during the social interaction; this is an indication of group cognition. Group cognition is defined by Stahl (2007) as learning that is constituted of the interactions between the participants. Finally, in turn 12, Lucas happily exclaims "yes!", stating that he has created a code that is working and shares knowledge of this.

Figure 2 is a screenshot of the program code they created (as referred to in turn 12). It depicts how the micro:bit can calculate the probability of getting the number "6" on the dice roll when rolling the dice 10,000 times. Figure 3 is a screenshot of the micro:bit when it runs the code in Figure 2. Figure 3 illustrates how the number "0.17" is visualised and printed on the micro:bit screen when running the code.

Figure 2. Screenshot of the programmed code in extract 1 (turn 12)

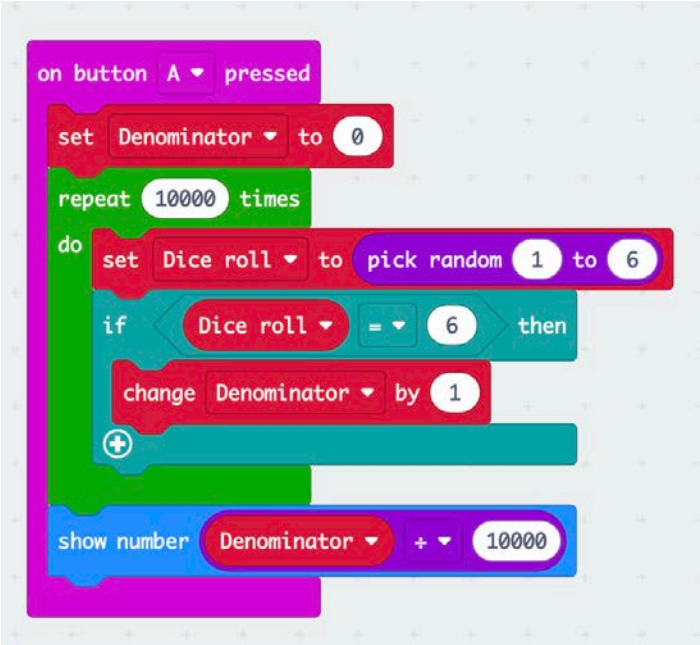
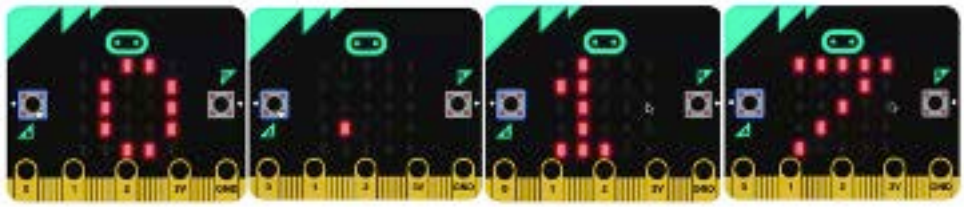


Figure 3. Screenshot of the micro:bit and the result of running the code in Figure 2



Extract 2: Discussing how to fix a code that is not working – collaborative debugging

Contextualising the extract

This extract stems from the same discussion as extract 1 above and presents the final part of the discussion. The focus is on three pupils who collaborate on how to help James find a solution for how to fix his code, which is not working.

Table 3. Data extract 2 (translated into English by the author)

Turn	Actor	Utterance	Analytical concept
1	James	Oh! It is not working Lucas!	
2	Lucas	What?! Did the code turn out to be the same as this? (pointing to his screen)	
3	Noah	Yes, I have the same (code) as that	
4	Lucas	Is it working?	
5	Noah	Yes, it is working.	
6	James	I must have overlooked a small detail. I will set all my variables to the exact same (parameters) as you have.	Computational concept (parameter, variable), computational practice (debugging and testing)
7	Lucas	Yes, but do not set so high numbers here then (because if the number of repetitions is high it takes longer time to test the code) – set it to only 10,000 (repetitions in the loop) (instead of 100,000)	Computational concept (repetition, parameter, loop)

8	James	Yes, that is what I am trying to do. Okey, then I am changing the variable “denominator”	
9	Lucas	Have you tried to change the number down here (pointing at the maths block in MakeCode) to divide – so that it divides here?	Computational concept (testing and debugging), knowledge sharing
10	James	Eh, yes. (clicking on the computer) now I have! Hehe but it still does not work now either	
11	Noah	James, have you made sure to set the parameter in your if-loop so that it says “if-dice roll-then” and not “if denominator-then”?	Meaning-making (turns 11–23), computational concept (loop, conditionals)
12	James	Yes. I must have made some small mistake	
13	Lucas	You must not change the variable set “denominator”, you must not write "change dice roll" you must write "set dice roll to" on where you have the purple (block), no, not down there, a little further up, you need to change it	Meaning making, computational concept (variable)
14	James	Ahh! (frustrated)	
15	Lucas	Yes, is it (the code) working now? 0.003… I do not know if it is completely correct ...	Meaning-making
16	James	Have I done everything right now?	
17	Lucas	Now it actually looks pretty good, does it not?	
18	James	Yeah ... 0.0 ... no! Now it showed something else (visualised on the micro:bit screen)	
19	Lucas	0.00 but that is a completely wrong number!	
20	Lucas	Do you have too many zeros? Shall we see—one-two-three	
21	James	Do I have too many zeros?	
22	Lucas	No, it is the right number of zeros	
23	James	Can you see what is wrong?	
24	Noah	James, I told you it is because you have…	Group cognition
25	Lucas	=he has the right variables in the right places (In the code) – eh no!	Group cognition, computational concept (variable)
26	Noah	=no! “denominator”! so it should not be the denominator – change it (the variable) to “dice roll” where it says “denominator” – it is the second highest variable – the purple (block)	Group cognition, computational concept (variable)
27	James	=the variable second from the top (referring to a drop-down button) – the purple (block)	
28	Noah	Yes! There it should say “roll the dice”	
29	Lucas	Oh, that's what I messed with earlier when I switched (the variable block)	
30	James	Try now! Yey! (clapping)	
31	Lucas	Yey!	
32	Noah	Yey!	

Analysing the extract

In turn 1, James exclaims that his code is not working, and Lucas (in turn 2) asks James if he has created the same code as him. Noah replies, in turn 3, that he has

a similar code as Lucas. Next, Lucas asks Noah (in turn 4) if his code is working and he responds it does (turn 5). In turn 6, James underlines that his code is still not working and that he will attempt to set all parameters as the same as the other codes in the group. Also in turn 6, James uses the computational concepts "parameter" and "variable" when suggesting a strategy for how to debug and figure out how to make the code to work. When stating that he will set all his variables to the exact same as his peers, he suggests a debugging strategy as part of a computational practice (Brennan & Resnick, 2012). Computational practice is defined as the processes wherein one is moving beyond what they are learning to how they are learning. In these processes, several practices were identified, such as being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularising (Brennan & Resnick, 2012).

In turn 7, Lucas follows up James' debugging approach (setting the same parameters as the others in the group) but emphasises that he should not set excessively high parameters on the repetitions of the loop since it takes a longer time to the test the code the higher the parameters are set. This is an example of a debugging strategy which is in line with Brennan & Resnick (2012), underlining that when programming code, it rarely works just as imagined; it is therefore necessary to develop strategies for dealing with and anticipating problems. Testing and debugging are components of computational practices when programming code. In turn 8, James confirms that he will follow the advice from his peers. In turn 9, Lucas suggests changing the variable, which is an example of how the pupils use computational concepts (Brennan & Resnick, 2012) as part of their discussion and social interactions. Turn 9 also contains a component of knowledge sharing (Stahl, 2007) when Lucas provides information about changing the variable. Nevertheless, the code is still not working. This triggers a meaning-making process in turns 11 through 23 when Lucas and Noah suggest different ideas for how to fix the code and make it work. The pupils build on each other's statements, such as in turn 11: "James, have you made sure to set the parameter in your if-loop …" Lucas responds in turn 13, saying "you must not change the variable". Meaning-making is a social and collaborative activity in which there emerges a collaborative construction of new problem-solving knowledge wherein individuals negotiate and share meanings relevant to the problem-solving task at hand (Stahl, 2007, referring to Roschelle & Teasly, 1995). In turns 11 through 23, several computational concepts are mentioned by the pupils: In turn 11, Noah talks about loops and if-then conditionals. Lucas, in turn 12, also discusses how to program the variable and strives to understand how the conditionals should be defined. In turns 11 through 14, different suggestions for how to fix the code are proposed which contain similarities to a meaning-making process (Stahl, 2007). The pupils are collaborating in creating solutions for how James can fix the code and make it work again, which can be interpreted as an example of *collaborative debugging*. Collaborative debugging can be identified as a process wherein different pupils interact and collaborate in trying to fix a

program code that is not working. In turn 15, Lucas says, "the code is working now," and he continues to read aloud the result of the code ("0.003") and carefully says that the code still does not work. From this, one can infer that part of the collaborative debugging process is to read each line of the code aloud, to prompt discussion of the code, invite new suggestions, and build on the existing suggested code ideas. The pupils adapt, refine, run, and test the code to assess whether it works (as in turns 15 and 17) and continue the process in iterations. In turn 24, Noah proposes an idea for how to fix the code but is interrupted by Lucas. In turn 25, Lucas goes through the code once more and suggests another idea for how to fix the code, but in turn 26, Noah strongly disagrees and builds on Lucas' utterance by stating that he has found the solution for how to fix the code. In response, James eagerly exclaims that he has found the solution and elaborates on it, building on previous utterances by Noah. These utterances have similarities with a group cognition process (Stahl, 2007) in that they are collaborating on creating a common solution to the problem by presenting different ideas for how to fix the code; they build on each other's ideas and ultimately reach a common ground and solution. Group cognition, as defined by Stahl (2007), is a gradually evolving result of group discourse. Finally, in turns 30 through 32, they exclaim "yey," which is a confirmation that the code is working.

Based on analysing the empirical data in this extract, collaborative debugging is defined as a strategy that involves a gradual development of an emerging shared understanding of how to fix the code, in which the participants collaborate in creating by building on each other's statements. Collaborative debugging is an emerging social phenomenon that evolves from using block-based programming as a method for learning a subject (maths) in a collaborative context. Extract 1 describes how the pupils collaborate on programming the micro:bit to calculate probability. In extract 2, it becomes clear that James has not managed to program the code (which they agreed upon in extract 1), and the other pupils help him find a strategy to make the code work.

Discussion: Implications of integrating block-based programming into the maths subject

Integrating block-based programming into the maths subject enabled active and collaborative learning

In extracts 1 and 2, I have shown instances of a gradually evolving meaning-making process (Stahl, 2007) emerging as a result of turn-taking and social interactions between the pupils when discussing how to use block-based programming to solve the maths task. Stahl (2007) states that in these processes, the meaning depends upon indexical references to the shared situation, elliptical references to previous utterances, and projective preferences for future utterances. This article expands the view of CT to also account for collaborative learning. In extracts 1

and 2, several examples of social interactions which entailed knowledge-sharing, meaning-making, and group cognition were revealed. In processes of meaning-making, collaboration is conceptualised as a shared meaning construction, in which meaning-making processes are an interactional achievement (Stahl, 2007).

Integrating block-based programming into the maths subject proved to be an engaging means to learn maths, and it sparked active discussions around the maths concepts that were the subject of the learning process. For example, in extract 1, the pupils eagerly discussed the nuances regarding the implications of a high or low probability and how this could be transferred to the block-based programming code they were creating (Extract 1, turn 7). Thus, one implication for CT when integrating block-based programming in the maths subject is that it triggers discussion, social interaction, and active learning connected to the topic being learned. Combining aspects from both CT and collaborative learning in the conceptual framework provided a richer data analysis and enabled us to identify collaborative debugging as a computational practice. Collaborative debugging is identified in extract 2 as when pupils collaborate through a trial-and-error strategy for fixing each other's codes.

Collaborative debugging emerges from using block-based programming as a method for learning a subject in a collaborative context. It follows the same line of thought as Kafai (2016), who seeks to reframe computational thinking as computational participation. This implies viewing CT as social and creative practices emerging when sharing and collaborating with others. As a result, a contribution of this article is defining collaborative debugging as a social and collaborative process that arises from CT and computational practices when learning block-based programming as an integrated part of a subject. This aligns with Ludvigsen and Steier (2019), who asserted that joint attention, or meaning-making, is necessary when solving complex tasks, and that joint attention emerges when participants are involved in solving problems together via coupling between representational artifacts, practices, and the social systems in which humans participate.

Integrating programming into a maths subject enabled the development of CT

From a CT perspective, it is evident that computational concepts such as "loops" and "variables" are significant elements of the dialogue. The computational concepts the pupils engage with are common concepts in several programming languages (Brennan & Resnick, 2012). When pupils are solving maths tasks using block-based programming, the social interactions and collaborations contain central computational concepts. Brennan and Resnick (2012) are interested in how CT can be interpreted as materialising learning and development that takes place with Scratch (a block-based programming language). This article follows along the same lines when exploring how micro:bit is used as an integrated part when solving maths tasks. This article presents a context and set of opportunities for

enabling conversations about CT. This indicates that using block-based programming as a method for learning a subject-specific topic has the potential to facilitate active collaborations and social interactions empowering pupils to develop aspects of CT. One implication of this integrated approach to block-based programming is that it triggers CT processes. This is consistent with Shute et al. (2017), who posited that CT skills appear to be improved via computational tools, such as Scratch. A contribution from this article is expanding the perspective of Brennan and Resnick (2012) by taking the collaborative learning aspects into consideration when exploring CT. In extracts 1 and 2, there is an intertwined use of both CT and collaborative learning aspects.

Block-based programming as a method for learning maths

This article is an empirical study serving as an example of how block-based programming can be used as a method for learning a specific subject, in this case maths. In extracts 1 and 2, the students are programming the micro:bit to calculate probability. In these social interactions, the discussion is dynamically switching between discussing subject-specific skills, such as maths, and the more generic skills (such as CT and collaborative learning). This supports previous work where there is a mutual dependency between the subject specific and the more generic skills when using Minecraft as a digital learning resource in social science (Andersen et al., 2021). When block-based programming is used as a method for solving maths tasks in extracts 1 and 2, the pupils use computational concepts as part of their collaborative learning processes. This has similarities to the study by Lee et al. (2020) which contended that pupils who learn to develop computational solutions and to employ computational tools and methods will advance their understanding of subject-area content and CT skills. However, the present article differs from the study by Weintrop et al. (2016) because it is an empirical study exploring how CT is integrated into a STEM subject, whereas Weintrop et al. delineate a taxonomy for understanding CT in practice for maths and science (Weintrop et al., 2016). The present article also differs from Sengupta et al. (2013) because it is an empirical study as opposed to a review. This article provides an empirical study of block-based programming and CT in K–12. Barr and Stephenson (2011) clearly state that pupils must begin to work with algorithmic problem-solving and computational methods in K–12 education to manage their work lives and to live in a world heavily influenced by technology.

Conclusion and directions for further research

The research question addressed in this article is as follows: “In what ways can block-based programming be integrated into a maths subject, and what are the implications for computational thinking?” This has been addressed and discussed in previous sections. In summary, the main findings of this article are as follows:

1) Integrating block-based programming in a maths subject enabled active and collaborative learning.
2) Integrating programming in a maths subject enabled the development of CT.
3) Using block-based programming facilitated learning of maths.

This article has discussed the implications of introducing CT into K–12 education. The analysis indicates that integrating block-based programming into a STEM subject proved to be useful because it afforded the development of CT for pupils. The findings in this article have parallels to those of Weintrop et al. (2016) as they focused on block-based programming and how it can enable the development of CT skills. However, this article extends the work of Weintrop et al. (2016) because it is an empirical case study exploring the processes emerging between the pupils when using block-based programming in a maths class. In addition, the present article adds to current research on CT and block-based programming by extending the work of Sengupta et al. (2013), who presented a theoretical investigation of key issues that need to be considered for integrating CT into K–12 science classes.

One contribution from this article is elucidating how the different aspects of CT (computational concepts, computational practices, and computational perspectives) emerge and are addressed in discussions when collaborating on how to solve maths tasks using programming solutions in a block-based programming language. This follows the same strand of thought as Lye and Koh (2014), who emphasised that CT is more than programming, as it also encompasses problem-solving processes in which programming helps foster CT. However, this article extends this approach by presenting an empirical case study of the implications of integrating CT in a maths subject when using block-based programming as a method for learning maths. When discussing how to program the code, the pupils used computational concepts as part of their dialogue, which may trigger further development of CT. Reflections connected to computational perspectives were not prominently in the foreground of the data, but collaborative debugging was identified as a computational practice that evolved during the discussions. As a result, this article contains parallels to the notion of computational participation, because it regards collaborative learning as part of the CT process. Therefore, this article brings new knowledge to the field of block-based programming and CT in K–12 education. Kafai and Burke (2013) state that there has been a transition from a predominantly individualistic view of technology to one that incorporates a greater focus on sociological and cultural dimensions when learning programming and reconceptualising computational thinking as computational participation. However, this article extends Kafai and Burke's (2013) research by extending the current research in CT with an empirical case study exploring how CT is integrated into a subject in a collaborative setting and what the implications are. DiSessa (2018) emphasises that computers might fundamentally change learning,

with an emphasis on maths (and more generally STEM), and underscores that computation is viewed as a new form of literacy. Facilitating pupils to participate in collaborative learning processes by using technology is one of the essential 21st century skills that pupils are expected to have in the future (Voogt & Roblin, 2012). One novelty of this article is identifying how pupils develop CT skills while using block-based programming as a method for learning a subject in school, in a collaborative learning context. The findings in this article have implications for how teachers can design future learning, including how to integrate block-based programming in schools.

More research that explores the perspective of how block-based programming can facilitate and trigger collaborative discussions which help develop CT is needed. An area for further work could be to explore other subjects other than maths and to determine the implications for CT. It would be interesting to explore how block-based programming can be integrated into the social sciences and explore the implications for computational thinking.

Acknowledgement

This research is part of the research project “Programming in school” (ProSkap) funded by Oslo Regional Research Fund in Norway. Thank you to Ellen Egeland Flø for helping in organisation of the educational activities presented here.

About the author

Renate Andersen is an Associate Professor in technology enhanced learning at Oslo Metropolitan University. Her research interests are ICT and learning, computer supported collaborative learning, programming in school, and computational thinking in education.
Institutional affiliation: Section for digital competence, Faculty of teacher education and international studies, Oslo Metropolitan University, Pilestredet 52, 0167 Oslo, Norway.
Email: renate.andersen@oslomet.no

References

Andersen, R. (2019). *Mutual development in online collaborative processes. Three case studies of artifact co-creation at different levels of participation*. Doctoral dissertation, Faculty of Educational Sciences, University of Oslo. https://www.duo.uio.no/bitstream/handle/10852/70696/PhD-Andersen-2018.pdf?sequence=1

Andersen, R., & Mørch, A. I. (2009). Mutual development: A case study in customer-initiated software product development. In V. Pipek, M. B. Rosson, B. de Ruyter, & V. Wulf (Eds.), *End-User Development. IS-EUD 2009. Lecture Notes in Computer Science* (Vol. 5435, pp. 31–49). Springer, Berlin, Heidelberg.

Andersen, R., Mørch, A. I., & Litherland, K. T. (2021). Learning Domain Knowledge Using Block-Based Programming: Design-Based Collaborative Learning. In D. Fogli, D. Tetteroo, B. R. Barricelli, P. Markopoulos, & G. A. Papadopoulos (Eds.), *End-User Development. 8th International Symposium, IS-EUD 2021* (pp. 119–135). Springer. https://doi.org/10.1007/978-3-030-79840-6_8

Barab, S., & Squire, K. (2004). Design-Based Research: Putting a Stake in the Ground. *Journal of the learning sciences*, *13*(1), 1–14. https://doi.org/10.1207/s15327809jls1301_1

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). Developing computational thinking in compulsory education: Implications for policy and practice. *Joint Research Centre Report*, EUR 28295 EN. Luxembourg: Publications Office of the European Union. https://data.europa.eu/doi/10.2791/792158

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77–101. https://doi.org/10.1191/1478088706qp063oa

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* (Vol. 1, p. 1–25). Vancouver, Canada. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

diSessa, A .A. (2018). Computational Literacy and "The Big Picture" Concerning Computers in Mathematics Education. *Mathematical Thinking and Learning, 20*(1), 3–31. https://doi.org/10.1080/10986065.2018.1403544

Dwyer, H., Hill, C., Hansen, A., Iveland, A., Franklin, D., & Harlow, D. (2015). Fourth grade students reading block-based programs: predictions, visual cues, and affordances. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 111–119). https://doi.org/10.1145/2787622.2787729

Echeverría, L., Cobos, R., & Morales, M. (2019). Improving the students computational thinking skills with collaborative learning techniques. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, *14*(4), 196–206. https://doi.org/10.1109/RITA.2019.2952299

Flórez, F. B., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research*, *87*(4), 834–860. https://doi.org/10.3102/0034654317710096

Forsström, S. E., & Kaufmann, O. T. (2018). A literature review exploring the use of programming in mathematics education. *International Journal of Learning, Teaching and Educational Research, 17*(12), 18–32. http://hdl.handle.net/11250/2599710

Gretter, S., & Yadav, A. (2016). Computational thinking and media & information literacy: An integrated approach to teaching twenty-first century skills. *TechTrends, 60*(5), 510–516. https://doi.org/10.1007/s11528-016-0098-4

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Heintz, F., & Mannila, L. (2018). Computational thinking for all: an experience report on scaling up teaching computational thinking to all students in a major city in Sweden. *ACM Inroads*, *9*(2), 65–71. https://doi.org/10.1145/3210553

Heintz, F., Mannila, L., & Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K–12 education. In *2016 IEEE Frontiers in Education Conference (FIE)* (pp. 1–9). https://doi.org/10.1109/FIE.2016.7757410

Hine, C. (2000). *Virtual Ethnography*. Sage Publications, London.

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). *Computational thinking patterns.* Paper presented at the annual meeting of the American educational research association. New Orleans, LA. https://files.eric.ed.gov/fulltext/ED520742.pdf

Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the learning sciences*, *4*(1), 39–103. https://doi.org/10.1207/s15327809jls0401_2

Kafai, Y. B. (2016). From computational thinking to computational participation in K–12 education. *Communications of the ACM*, *59*(8), 26–27. http://dx.doi.org/10.1145/2955114

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, *95*(1), 61–65. https://doi.org/10.1177/003172171309500111

Kong, S. C., Abelson, H., & Lai, M. (2019). Introduction to computational thinking education. In S.-C. Kong & H. Abelson (Eds.), *Computational thinking education* (pp. 1–10). Springer, Singapore.

Lee, I., Grover, S., Martin, F., Pillai, S., & Malyn-Smith, J. (2020). Computational thinking from a disciplinary perspective: Integrating computational thinking in K–12 science, technology, engineering, and mathematics education. *Journal of Science Education and Technology*, *29*(1), 1–8. https://doi.org/10.1007/s10956-019-09803-w

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, *30*, 883–908. https://doi.org/10.1007/s11191-021-00202-5

Ludvigsen, S., & Steier, R. (2019). Reflections and looking ahead for CSCL: Digital infrastructures, digital tools, and collaborative learning. *International Journal of Computer-Supported Collaborative Learning*, *14*(4), 415–423. https://doi.org/10.1007/s11412-019-09312-3

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. https://doi.org/10.1016/j.chb.2014.09.012

McKenney, S., & Reeves, T. C. (2018). *Conducting educational design research*. Routledge.

Mohaghegh, M., & McCauley, M. (2016). Computational Thinking: The Skill Set of the 21st Century. *International Journal of Computer Science and Information Technologies*, *7*(3), 1524–1530. https://hdl.handle.net/10652/3422

Moreno-León, J., Robles, G., & Román-González, M. (2016). Code to learn: Where does it belong in the K-12 curriculum? *Journal of Information Technology Education: Research*, *15*, 283–303. https://doi.org/10.28945/3521

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*, *11*(1), 1–17. https://doi.org/10.1080/20004508.2019.1627844

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

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351–380. https://doi.org/10.1007/s10639-012-9240-x

Serrano-Cámara, L. M., Paredes-Velasco, M., Alcover, C. M., & Velazquez-Iturbide, J. Á. (2014). An evaluation of students' motivation in computer-supported collaborative learning of programming concepts. *Computers in human behavior*, *31*, 499–508. https://doi.org/10.1016/j.chb.2013.04.030

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Silverman, D. (2005). *Doing Qualitative Research*. London: Sage.

Stahl, G. (2007). Meaning making in CSCL: Conditions and preconditions for cognitive processes by groups. In C. A. Chinn, G. Erkens, & S. Puntambekar (Eds.), The Computer Supported Collaborative Learning (CSCL) Conference 2007 (Volume 8, Part 2, pp. 651–660). New Brunswick, NJ, USA: International Society of the Learning Sciences. https://repository.isls.org//handle/1/3424

Stahl, G., Koschmann, T., & Suthers, D. (2006). Computer-supported collaborative learning: An historical perspective. In R. K. Sawyer (Ed.), *Cambridge handbook of the learning sciences* (pp. 409–426). Cambridge, UK: Cambridge University Press.

Voogt, J., & Roblin, N. P. (2012). A comparative analysis of international frameworks for 21st century competences: Implications for national curriculum policies. *Journal of Curriculum Studies*, *44*(3), 299–321. https://doi.org/10.1080/00220272.2012.668938

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, *62*(8), 22–25. http://dx.doi.org/10.1145/3341221

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. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–36. https://dl.acm.org/doi/fullHtml/10.1145/1118178.1118215?casa_token=7LWW0vnmNjwAAAAA:QnoLWqIQR_eYKc8HxdCK1EsOs3dkqv6mqbGKv8RQMYm2aWhs1e1dPVZdRJFCL_et0YjEBhTNLeUqXHg

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education. In P. Rich & C. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 205–220). Springer, Cham. https://doi.org/10.1007/978-3-319-52691-1_13

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education*, *14*(1), Art. 5. https://doi.org/10.1145/2576872

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607. https://doi.org/10.1016/j.compedu.2019.103607

Zoom (2021). *Zoom*. Retrieved 4 October 2021 from https://zoom.us/