Education and Information Technologies (2023) 28:10183–10208
https://doi.org/10.1007/s10639-023-11577-z

A conceptual model of what programming affords secondary school courses in mathematics and technology

Niklas Humble[1]

Received: 28 February 2022 / Accepted: 5 January 2023 / Published online: 24 January 2023

Abstract
Due to increased need of professionals on the future labour market with competence in programming, many countries have integrated programming in kindergarten to grade 12 (K-12) education. In 2017, programming was integrated in Swedish primary and secondary school curriculum and the courses of Mathematics and Technology. Research has highlighted challenges in integrating programming and other new technologies, and the need for better teacher support. The aim of the study was to examine what programming affords secondary school courses in Mathematics and Technology according to teachers that use programming in these two courses. The study was conducted with a qualitative approach and collected data through interviews with 19 teachers that use programming in secondary school courses of Mathematics and/or Technology. Thematic analysis with inductive-deductive approach was used to analyse the collected data. Theory of Affordances was used to identify themes of interests in the collected material and group these into categories. Ten programming affordances are identified in this study: 1) flexibility, 2) creativity, 3) efficiency, 4) visualisation, 5) fun, 6) curiosity, 7) play, 8) holistic views, 9) fearlessness, and 10) interdisciplinary collaborations. Through discussion and comparison with previous research, these programming affordances are found to relate to three aspects of teaching and learning in secondary school Mathematics and Technology: A) support course content and learning, B) facilitate engagement and motivation, and C) foster developmental skills. The study concludes with a suggestion for a conceptual model on what programming can afford school courses in Mathematics and Technology, based on the findings of the study. Findings can be used by teachers, policymakers and other stakeholders in the integration and design of programming in secondary education.

Keywords Programming · K-12 education · Teacher perspective · Affordances · Constraints

✉ Niklas Humble
niklas.humble@miun.se

[1] Department of Computer and System Science, Mid Sweden University, Akademigatan 1 - Building Q, 831 40 Östersund, Sweden

1 Introduction

Reports show a need for professionals with programming competence on future labour market due to increased automation of jobs (Smit et al., 2020). As a response, many countries have integrated programming in kindergarten to grade 12 (K-12) education (Nouri et al., 2020; Tran, 2018). Two possible approaches for integrating programming in K-12 education are: in existing curriculum or as its own subject (Nouri et al., 2020). Sweden and Finland are two examples of countries that have integrated programming in existing curriculum (Pörn et al., 2021; Heintz et al., 2017); while UK has integrated programming through the subject of Computing (Royal Society, 2017). Both approaches have received similar criticism, stating that there is insufficient support for the teachers and their professional development (Pörn et al., 2021; Royal Society, 2017). As of 2017, programming is part of the Swedish primary and secondary school curriculum and the two courses of Mathematics and Technology (Heintz et al., 2017).

Previous research on integrating new technologies in educational setting show that there is a need for examples of good use for the integration to be successful (Viberg et al., 2020; Lindberg et al., 2017). Since the integration of programming technologies are still being conducted in many countries' K-12 education, where Sweden is one of them, the need for research that study the use, possibilities and challenges of programming is still crucial. Previous research show that teachers need guidance for the use and integration of programming in K-12 education, this includes support for teaching programming and using programming as a tool for teaching and learning in other courses (Humble, 2022; Humble et al., 2020; Pörn et al., 2021; Szabo et al., 2019; Humble et al., 2019; Webb et al., 2017; Royal Society, 2017). In Sweden, these courses are mainly the two courses of Mathematics and Technology. A study by Murai and Muramatsu (2020) show promising results for teacher professional development in programming where teachers can receive support from fellow K-12 teachers. The objective for this study was to provide guidance for teachers and other stakeholders in the ongoing integration of programming in K-12 education, through examples of programming use in Swedish secondary school courses of Mathematics and Technology.

Previous research has studied the use and integration of programming in K-12 setting. However, the focus is often on programming in relation to Computational Thinking (CT) and how these can be used to develop 21st century skills among students (Tikva & Tambouris, 2021). But there are other possible theories and frameworks that could be applied to better understand the possibilities and constraints of programming in educational contexts. One theory that seems especially well suited for investigating possibilities and constraints of programming to support teaching and learning in other subjects, such as Mathematics and Technology, is Affordances. Despite this, Affordances has not received the same attention as CT in previous research on programming in K-12 education, which is a gap that this study is addressing. The theory of Affordances highlight what actions are made (or perceived as) possible through the utilisation of an artefact (Bower & Sturman, 2015) and what actions are constraint (Norman, 2013, p.125).

Besides the more common concepts of 'perceived affordances' and 'constraints' in the theory, Norman (1999) argues that the most important part is the conceptual model. The conceptual model offers a useful and simplified explanation of how something works, which all other parts should be consistent with (Norman, 2013, p.25; 1999). This study concludes with a conceptual model of what programming can afford secondary school courses in Mathematics and Technology. To the author's knowledge, a conceptual model based in the theory of Affordances of what programming can afford K-12 education has not been presented in previous research.

This study focuses on the perceptions by Swedish secondary school teachers to develop an understanding of how programming can be used for teaching and learning in the courses of Mathematics and Technology. Data have been collected through interviews with 19 teachers that are experienced users of programming in secondary school courses of Mathematics and/or Technology. Data are therefore limited and findings should not be understood as representative for teachers' perceptions in general. However, with the qualitative approach used in this study, the objective was not to develop a generalisable understanding of teachers' perceptions, but rather to provide guidance for teachers and other stakeholders. This is provided through the investigation of how teachers, that are experienced users of programming, use and perceive programming for teaching and learning in secondary school courses of Mathematics and Technology.

The aim of the study was to examine what programming affords secondary school courses in Mathematics and Technology according to teachers that use programming in these two courses. The study was guided by the following research questions:

RQ1) What are the teachers perceived affordances of programming for secondary school courses in Mathematics and Technology?
RQ2) What are teachers perceived constraints of programming for secondary school courses in Mathematics and Technology?

2 Theoretical framework

The theory of *Affordances* was coined by Gibson and has its roots in ecological psychology with an empiricist approach to action and perception (Chong & Proctor, 2020). Since Gibson, affordances have been applied in several fields of research and deviated from the original definition (Chong & Proctor, 2020). One of these fields are information and communication technology (ICT) in education (Hammond, 2010). The application of, and related research to, affordances for ICT in education is presented in Sect. 3 *Related work*. In this section, descriptions of the foundational concepts of *affordances*, *perceived affordances*, *constraints*, and *conceptual models* are provided.

Affordances are what the environment provides the animal, including substances, objects, surfaces, places, medium, and other animals (Gibson, 1977). According to Bower and Sturman (2015), Gibson's definition entail that an affordance of the environment exists when there is an animal there to utilise it through actions. With that, affordances are neither good nor bad but simply entails what the environment provides for the animal (Gibson & Carmichael, 1966, p.285). For example, an affordance of a mailbox is to mail letters, and this is true weather a person recognises this or not (Bower & Sturman, 2015). The properties of affordances can have multiple meanings, which stem from the different functions that an object can serve for the animal; for example, a book can serve as both a doorstop, an information repository, and hold down papers when the wind blows (Heft, 2001, p.131).

With Norman's (1999) development of perceived affordances, a designer perspective is introduced to the theory. Perceived affordances differ from affordances according to Norman (2013, p.19) because affordances are all the interactions that are possible (perceivable or not) between environment and people; while perceived affordances are about signalling what course of actions are possible (Norman, 2013, p.19). Perceived affordances are often signifiers and as such they must be perceivable, else they lose their function (Norman, 2013, p.19). The reason people understand how to operate novel objects or devices is because the required information is part of the design or appearance as clues (Norman, 1999). According to Norman (1999), this understanding consists of three dimensions: *conceptual models*, *constraints*, and *perceived affordances*.

Constraints limits the actions perceived as possible with a known or novel objects and can therefore provide clues for the proper action to take (Norman, 2013, p.125). Constraints consists of four classes: logical, semantic, cultural, and physical (Norman, 2013, p.125). A logical constraint is for example the logical assumption that if there are a left and a right light and a left and a right light switch, the left light switch should control the left light (Norman, 2013, p.130). This is not determined by any cultural or physical principles, rather, there is a logical relationship between the objects and how these should affect each other (Norman, 2013, p.130). A semantic constraint is for example that the meaningful place to sit on a motorcycle is facing forward, behind the windshield (Norman, 2013, p.129). This is based on knowledge about the world and the situation, the windshield protects the face of the rider. But as the world and technologies develops, semantic (and cultural) constraints may change (Norman, 2013, p.129).

A cultural constraint is for example that red is the colour of a break light or that you should not look directly at another passenger in an elevator (Norman, 2013, p.129). These rules or allowable actions are based in our culture and differ between cultures and are likely to change over time (Norman, 2013, p.128–129). A physical constraint is for example that a small hole cannot fit a large peg (Norman, 2013, p.125). Physical constraints may seem the most obvious constraint, but they can still be difficult to interpret. Consider a cylindrical battery where both ends are nearly identical, a more effective physical constraint would be to design the battery so that its shape can only fit in one direction (Norman, 2013, p.125–126).

Conceptual models provide a simplified and useful explanation for how something works (Norman, 2013, p.25). For example, icons, folders, and files displayed on a computer screen creates the conceptual model of folders and documents inside the computer (Norman, 2013, p.25). These are representations to make the computer easier to use, but there are no "real" documents and folders in the computer (Norman, 2013, p.25).

According to Norman (1999), the underlying conceptual model is the most important part of design, and it is crucial that everything else is consistent with it.

3 Related work

To address high drop-out rates and failure in introductory programming courses, previous research has pointed out that not enough emphasis is put on problem-solving strategies (Malik & Coldwell-Neilson, 2017). Focus has been on surface knowledge of programming in the form of syntax, semantics, and concepts, when it should instead be on deep learning (Malik & Coldwell-Neilson, 2017). Previous research that investigates novice programmers' problem-solving skills through debugging has further concluded that debugging requires deeper understanding than simply how to write code (Beege et al., 2021; Liu et al., 2017). Studies that investigate game elements and gamification in programming courses have found that the approach can have a positive effect on students' motivation (Fotaris et al., 2016; Pilkington, 2018). Previous research on modelling games in K-12 science education has highlighted the importance of representing phenomenon in many ways as part of game design (Krinks et al., 2019). This is important since moving between different representations can engage and deepen students' conceptual understanding (Krinks et al., 2019).

Previous research has investigated differences between block programming and textual programming in K-12 education, regarding learning outcomes and program comprehension (Krishnamurthi & Fisler, 2019). A well-known example of block programming is Scratch, which was developed with the idea of low floor. Low floor in Scratch indicates that it should be easy to get started and that syntactic mistakes should be limited by using compatible blocks that represent programming code (Fagerlund et al., 2021; Maloney et al., 2010; Scaradozzi et al., 2019; Toma, 2021). Although graphical elements of block programming have affordances, programming is still a complex cognitive practice (Fagerlund et al., 2021). Previous research has used Scratch for project-based integration of programming in multiple courses of K-12 education with positive results (Sáez-López et al., 2016). Previous research has also examined the impact on learning textual programming in K-12 education by applying visualisation techniques (Mladenović et al., 2021). The study concluded that visualisation could boost programming understanding, but that students should actively engage with visualisation and not be passive participants (Mladenović et al., 2021).

Previous research has identified some affordances for block programming over textual programming. One example is that students show greater interests and are more likely to understand if-/else-expressions with block programming, than with textual programming (Krishnamurthi & Fisler, 2019; Weintrop & Wilensky, 2017). However, studies have also found constraints with block programming, such as K-12 students' misconceptions about loops, Booleans and variables (Krishnamurthi & Fisler, 2019; Grover & Basu, 2017). Previous research also show that students perceive the learning experience as more professional and effective for improving programming skills, when engaging with textual programming (Weintrop & Wilensky, 2017). Knowing that a programming tool is authentic and used in professional setting can be inspiring for students (Garneli et al., 2015; Maggiore et al., 2011).

In a study on affordances of BBC's micro:bit, it is concluded that the tangible element is key to support understanding, stimulate interest and act as motivator in classrooms (Sentance et al., 2017). According to the study, micro:bit encourages creativity and helps students connect the relevance of programming to computer science and the real world by creating digital products (Sentance et al., 2017). Tangible programming further has the potential of making programming more attractive for K-12 students since affordances of the tangible objects, controlled by programming, are less ambiguous than objects in virtual environments (Papavlasopoulou et al., 2017). However, students that are older and more experienced with computers tend to find graphical environments attractive as well (Papavlasopoulou et al., 2017).

In a study on affordances of block programming tools with virtual and tangible elements aimed for younger children, the authors conclude that all tools contain potentially problematic assumptions on the relationships between play and children's engagement and motivation (Clarke-Midura et al., 2019). For example, some children can be motivated by the idea of a coding playground, while others may find it difficult to code on their own (Clarke-Midura et al., 2019). Previous research has argued that interacting and tinkering with programming code offers unique affordances for K-12 students to engage in inquiry (Wagh et al., 2017). Interacting with code gives students opportunity to access rules, variables, and structures of scientific phenomenon, and the possibilities of breaking down phenomenon, look inside and make structures transparent (Wagh et al., 2017; Resnick et al., 2000).

Previous research that has investigated how Swedish kindergarten to grade 9 teachers interpret the integration of programming in the course of Mathematics found 4 different relationships between programming and Mathematics: 1) programming but not connected to mathematics, 2) programming in the context of mathematics, 3) doing efficient calculations with programming as a tool, and 4) exploring mathematics with programming as a tool (Kilhamn et al., 2021). In a study that investigated how Swedish teachers in grade 1 to 9 prepared for the implementation of programming in the Mathematics and Technology courses it is concluded that the teachers did not feel prepared for the implementation of programming (Vinnervik, 2022). It is further highlighted that there is risk for inequality among schools regarding access to support, professional development, IT infrastructure, and active leadership in the digital transformation (Vinnervik, 2022). Similar results were highlighted in a study that investigated Swedish secondary school teachers' perceptions on integrating programming in Mathematics and Technology courses and of different programming tools (Humble et al., 2020). The study concluded that programming is perceived by many of the teachers as fun but that there is a lack of time and directives for the integration, which the teachers perceived as challenging (Humble et al., 2020).

4 Method

According to Maxwell (2008), a common interest in qualitative studies is on participants' sensemaking or understanding of a phenomenon, and how their behaviours are affected by this understanding. This study was conducted with a qualitative approach to examine secondary school teachers' understanding of programming use and how this influence teaching and learning activities. Further, qualitative research should illuminate participants' understandings of their subjective actions, social contexts, and meanings (Fossey et al., 2002). This study illuminates teachers' subjective understandings of affordances and constraints in using programming for secondary school courses in Mathematics and Technology.

4.1 Data collection

Data were collected through semi-structured interviews with 19 teachers that teach secondary school courses in Mathematics and/or Technology in grades 7–9 (student ages 13–15) (Table 1). A drawback with semi-structured interviews is that they usually do not encompass a sample large enough to draw general conclusions (Adams, 2015). However, semi-structured interviews allow for a deep understanding of individual thoughts through open-ended and probing questions (Adams, 2015). Given the aim of the study, the goal is not to draw general conclusions of teachers' perceptions, but rather to examine how certain teachers, with specified experiences and interests, perceive the studied topic. With this background and a purposive sampling (Etikan et al., 2016; Campbell et al., 2020), the method was deemed appropriate for data collection.

The first step of data collection was to invite potential participants. This was done through invitation posts on social media and in an online forum of a teacher professional developments course on fundamental programming in the mid-Sweden region. Additionally, about 700 principals, administrators, and teachers in 64 municipalities in Sweden were contacted through email. The invitation contained a short presentation of the researcher and intended study. Invitations were submitted early in spring 2021. This resulted in 25 teachers registered their interest to participate.

The second step was to select participants. A short survey was sent to the potential participants, asking them to submit additional information about their workplace, experience in teaching and programming, and preference for conducting the interview. 20 teachers were selected with the criteria of teaching secondary school courses in Mathematics and/or Technology in grade 7–9 and be experienced in using programming for the courses in Mathematics and/or Technology. 1 teacher did not answer the invitation and was withdrawn from the study.

Third step was to interview the 19 teachers, which were conducted with video conference tools (Zoom or Teams), based on teacher preference. Each interview lasted between 24 and 50 min (with an average of about 36 min) and was recorded with consent from the teachers. Interviews were divided in two parts: 1) a structured introduction where teachers were asked to repeat information about themselves and their teacher background, 2) a semi-structured part where teachers were asked open-ended questions about programming in their courses (Appendix). Interviews were based on the same questions, but these were not asked in the same order or asked in the exact same way. Questions were used to guide and stimulate conversation rather than to regulate it. Interviews were conducted during the spring semester of 2021.

Participants were located all over Sweden, in schools and urban areas of different sizes. Age of the participants ranges from 27 to 63, years teaching ranges from 3 to

Table 1 Participating teachers

	South SWE	Mid SWE	North SWE	Age 20–30	Age 30–40	Age 40–50	Age 50–60	Age 60+	Teaching (years)	Programming (years)	Programming in Mathematics	Programming in Technology
T1	X						X		29	8–9	X	X
T2	X			X					5	3	X	X
T3		X				X			15	3	X	
T4	X				X				12	4		X
T5			X				X		12	1		X
T6		X			X				4	1		X
T7			X				X		27	2		X
T8		X						X	40	30		X
T9			X				X		6	3–4	X	X
T10		X				X			12	5	X	
T11		X			X				4	4	X	X
T12			X	X					3	3	X	
T13			X				X		9–10	3	X	X
T14			X				X		19	1	X	X
T15			X				X		26	5	X	X
T16			X			X			22	2	X	
T17			X					X	33	4		X
T18			X			X			11	3	X	X
T19			X		X				7	3	X	X

40, and years integrating programming in education range from 1 to 30. All participating teachers teach secondary school students (grades 7–9). 4 of the teachers only use programming in the secondary school course of Mathematics, 6 of the teachers only use programming in the secondary school course of Technology, and 9 of the teachers use programming in both courses (Table 1).

4.2 Data analysis

Thematic analysis was used to identify themes of shared or collective experiences and meanings in the collected data (Braun & Clarke, 2012). A mixture of inductive and deductive coding was used in three steps for analysis. First step, a data-driven inductive coding approach (Boyatzis, 1998) was used to identify themes in each interview that highlight important aspects of the conversation. Important aspects were considered topics that the conversation returned to, was emphasized by the teacher, or related to the aim and research questions of the study. First step was conducted directly after each interview and identified themes were collected in a text document.

Second and third step, a theory-driven deductive coding approach (Boyatzis, 1998) was used to identify themes of interests and group these in bigger themes, categories. Deductive coding in the second and third step was guided by the theoretical framework of Affordances, especially the concepts of 'perceived affordances' and 'constraints' (Norman, 1999, 2013). In the second step, themes identified in the first step were reviewed based on relevance for the theoretical framework and included or discarded. Included themes were moved to a spreadsheet and grouped in bigger themes (categories) formulated as 'main perceived affordances'. In the third step, each interview was analysed again in search for additional themes relating to the theoretical framework. Newfound themes could either support or rephrase existing categories or form new categories.

Categories, and included affordances and constraints, were also compared and discussed in relation to previous research. This could be considered a fourth step of analysis. In this step, three aspects of teaching and learning in secondary school courses of Mathematics and Technology were identified, to which the categories related to. Together these aspects, categories, and related affordances and constraints form a potential conceptual model for what programming can afford secondary school courses in Mathematics and Technology. Findings related to the first, second and third step of analysis are presented in the results and analysis-section. Findings related to the fourth step are presented and discussed in the discussion-section. The conceptual model is presented in the conclusion-section.

4.3 Trustworthiness

To establish trust, the study has been conducted in accordance with trustworthiness and associated criteria for credibility, transferability, dependability, and confirmability (Schwandt et al., 2007). According to Bryman (2016, p.383–384), trustworthiness was developed as a more applicable alternative for judging and evaluating qualitative research, compared to the more common validity and reliability in quantitative research. To ensure credibility, the study includes interviews with 19 teachers that teach secondary school courses in Mathematics and Technology, from all over Sweden, and are experienced users of programming. To facilitate transferability, details have been provided about context for the study, participants, data collection, and the interview guide. To ensure dependability and confirmability, detailed description of data analysis is provided and extracts from interviews are presented in the results and analysis-section to exemplify and support identified themes and categories.

5 Results and analysis

In this section, themes that were identified in the analysis and relates to the study's aim and research questions are presented. The aim of the study was to examine what programming affords secondary school courses in Mathematics and Technology according to teachers that use programming in these two courses. The research questions to address were: 1) What are the teachers perceived affordances of programming for secondary school courses in Mathematics and Technology? 2) What are teachers perceived constraints of programming for secondary school courses in Mathematics and Technology? Through the analysis, 10 categories, or main perceived affordances, have been identified, which contain themes of both affordances and constraints.

The 10 categories, which are also used as sub-headings below, are: 1) flexibility, 2) creativity, 3) efficiency, 4) visualisation, 5) fun, 6) curiosity, 7) play, 8) holistic views, 9) fearlessness, and 10) interdisciplinary collaborations. Extracts from the interviews are presented in the sub-headings to exemplify and support the identified themes. Extracts have been translated from Swedish to English and are to some extent rephrased for readability. However, the underlying meanings of the extracts have not been changed.

5.1 Programming affords flexibility

A teacher that uses programming in the Mathematics course explains that programming makes it possible to be more flexible in teaching and learning activities. With programming, mathematical concepts can be used in other contexts and from different perspectives to support and reinforce important practices. Such as being precise and doing step by step calculation. Another teacher explains the difference in teaching mathematics since programming was integrated and highlight the flexibility that it brings. Having yet another tool (programming) to work with, has allowed for slowing down the course and give students more time to process information and thoroughly understand it. However, this also means that fewer components can be part of the course, which can be viewed as a constraint for using programming. (Quote 1)

> ”I feel that the biggest difference for me, as a Mathematics teacher, […] is that you slow down and give students time to process information and understand. And there we’ve gotten another tool [programming] to work with. So, we probably do fewer [mathematics] components but we do them more thoroughly instead.”
> **Quote 1**. Teacher about differences in teaching mathematics since integrating programming

An identified constraint in the collected material is that block programming does not support flexibility to the same extent as textual programming. Teachers explain that this is because block programming, as used in Scratch or code.org, gives the user predefined blocks of code which limits what can be done. While textual programming, such as Python, give users flexibility to modify code in whatever way they like. This is perceived as important, especially in the Mathematics course. A teacher that uses programming in both the Mathematics course and the Technology course explains that textual programming is preferred because it is easier and faster to build solutions yourself, but that it could be useful for students to also work with block programming (Quote 2).

> ”I think that block programming can be a good thing. But I think that it’s faster and easier to work with text [textual programming] since it’s easier to, what should I say, customise and build it yourself. […] For the students, I think it’s good to work with both.”
> **Quote 2**. Teacher about textual programming being easier and faster than block programming

5.2 Programming affords creativity

An identified perceived affordance in interviews is that programming can afford creativity. One teacher explains that there is no single correct way of writing code and that it is the same in mathematics. Programming could therefore be used to foster creativity and support students to view mathematical problems in multiple ways. (Quote 3)

> ”In programming, there is not one way to code a program, and it’s the same thing in mathematics. You shouldn’t limit yourself and say: [explanation of change factor method], that is always the best. […] [If] you still solve the problem in a sensible way with mathematical methods and arrives at the correct conclusion. And I would say the same thing about programming, let them [the students] be creative […] and solve problems in different ways.”
> **Quote 3**. Teacher about creativity in programming and mathematics

Foster creativity in mathematics with support by programming is not only a question of solving mathematical problems in different ways, but also a question of not limiting yourself to one programming tool, according to a teacher. Even though textual programming is perceived as more flexible, it can be viewed as overwhelming and thereby constrain creativity. Therefore, it could be more appropriate to start with block programming. (Quote 4)

> "You shouldn't limit yourself to one way of solving programming assignments and not to one specific programming language, yet another connection to mathematics […] You shouldn't say that now it's only Python. Because I've heard from many other teachers […] that it feels overwhelming to start with a textual programming language and I can understand that. In that case, I think you should focus on the basics in programming instead."
> **Quote 4**. Teacher about textual programming as overwhelming and not limiting yourself to one programming language

5.3 Programming affords efficiency

Teachers explain in interviews that programming makes it possible, especially in mathematics, to solve problems that otherwise would be difficult or nearly impossible. A teacher explains that if students get to see the power programming brings to mathematics in efficiency for problem-solving, they will want to use it. A teacher that uses programming in both the Mathematics course and the Technology course explain that programming can be used to reinforce and support important practices that might be perceived as dull or routine. In the Mathematics course, the teacher explain that students got to develop a program to conduct left-side and right-side checks of equations. By doing this, more students now check their equations, which makes both the students' and the teacher's work more efficient. (Quote 5)

> ”We made a program that can check the solutions' right and left side. […] And in this way, I got more than half of the students to actually start to routinely check their equations. While otherwise it's only a few that does it, even though I keep nagging about it. This should be done with pen and paper […] I believe that because I got more to check their equations with a program, I have manged to repeat the practice until it stuck."
> **Quote 5**. Teacher about using programming to reinforce mathematical practices

An identified perceived constraint in the collected material is that textual programming does not afford efficiency for all students. For students with writing difficulties, the Mathematics course and the Technology course can be a place in school where performance is not as affected by these difficulties. By introducing textual programming in these courses, spelling and accuracy becomes more important. A teacher that uses programming in the Mathematics course explains that many students struggle with textual programming because of this (Quote 6).

> ”I have a lot of problems regarding students with writing difficulties. Because spelling is such a fundamental part of this [textual programming], and there is no spelling check. The smallest mistake gives an error to the whole program."
> **Quote 6**. Teacher about the challenges of spelling in textual programming

5.4 Programming affords visualisation

Teachers explain in the interviews that programming can support the understanding of complex and abstract knowledge in the Mathematics course and the Technology course through visualisation. A teacher explains that programming suits the Technology course very well, since it can be used to visualise how things work in real life. For example, automation and large constructions. Teachers further explain that programming can be used to visualise geometry in mathematics with block programming in Scratch or textual programming in Python (by importing Turtle library). One teacher explains that programming can be used to make probability theory in mathematics less abstract by programming visualisation of dice-rolling (Quote 7).

> ”In probability theory, very efficient, because it’s difficult to concretise. Weirdly enough, programming becomes a tool for concretising problem solving. Let us say that I would like to roll dice a thousand times, I wouldn’t do that during a lesson because it would take too much time, but with programming I can create a program and press a button.”
> **Quote 7**. Teacher about using programming to visualise probability theory

An identified constraint is that programming, or certain programming tools, are perceived as difficult by students and therefore limits visualisation. A teacher that uses programming in the Mathematics course explains that many students perceive textual programming in Python as more difficult than block programming. Therefore, they sometimes start with building the structure of the program in Scratch, before moving over to Python. However, Scratch has some perceived constraints, compared to Python, regarding visualisation of geometry. For example, it is not as intuitive to reset a program and return to origin, which can cause problems for students (Quote 8).

> ”When you’re going to draw something [on the screen], it’s easier to reset, to get back to origin. That’s usually what they [students] have the biggest difficulties with, they write something, they get it wrong, and they forget that they must reset the program. Because then they do their corrections, but the program starts from the wrong place, and so forth. That’s a bit easier to do in Python than in Scratch.”
> **Quote 8**. Teacher about constraints for geometry in Scratch compared to Python

5.5 Programming affords fun

It is expressed in interviews that programming is perceived as fun and exciting by both students and teachers. A teacher explains that the opportunity to explore new ways of teaching mathematics is what makes it fun. Even though integrating and learning programming requires extra work, the teacher express that this is the case for all new tools and approaches to teaching. A teacher that uses programming in both the Mathematics course and the Technology course explains that programming in the Technology course is always going to be more fun for students. In the Technology course, programming makes something happen and it becomes more tangible. The teacher further explains that both girls and boys find programming fun in the Technology course. (Quote 9)

> ”Programming is always more fun in Technology, because you program for something to happen, something tangible. [...] and there is definitely no difference in gender. Girls find this at least as fun as the boys do. Programming in Mathematics is not as, it’s not the same feeling of Wow.”
> **Quote 9**. Teacher about programming in the Technology course as more fun for both girls and boys

As mentioned above, the tangible nature of the Technology course affords more fun with programming than in the Mathematics course. This can be considered a constraint for programming in the Mathematics course, which typically does not have the same tangible elements. Another teacher also states that students are not as enthusiastic about programming in the Mathematics course as they are in the Technology course. But this could also be because students find textual programming in Python boring (Quote 10).

> ”In Mathematics on the other hand, there they [students] sigh more. [...] Because, as they said: Yes, but it’s so boring to sit here and write. And it’s Python that they are writing in.”
> **Quote 10**. Teacher about textual programming being perceived as boring in the Mathematics course

5.6 Programming affords curiosity

Students’ curiosity for programming is expressed in interviews to also spark interests for the courses of Mathematics and Technology. A teacher expresses that programming has elevated mathematical knowledge of both teacher and students, and that curiosity for programming can be used to reach students that previously were difficult to reach. Teachers mention that certain programming techniques and tools are better at sparking curiosity for the Mathematics course and the Technology course. Tangible programming with micro:bit and visualisation with Turtle library in Python are mentioned as examples. A teacher that uses programming in both the Mathematics course and the Technology course explain that the Turtle library in Python can be used to spark curiosity for programming among fellow teachers that have a negative attitude towards programming. With the Turtle library, they can use their knowledge in mathematics to program. (Quote 11)

> ”We have started to get some colleagues aboard now. We noticed that Turtle [library in Python] and to draw figures, colleagues were on board for that rather quickly. Because it’s quite easy for them to learn since they get quite far with their knowledge in mathematics. So, it’s easier to get colleagues on board, and not only students, and also colleagues that have a more negative attitude and don’t want to learn.”
> **Quote 11**. Teacher about Turtle library in Python to get teachers and students on board with programming

An identified constraint in the collected material is that there are negative preconceptions about programming, and who programming is for, that could hinder curiosity. A teacher that uses programming in the Technology course explain that important objectives are to rid programming of perceptions of mysticism and prejudice that it is only for computer geeks. To reach this objective, all students need to experience that they can program something. (Quote 12)

> ”My objective is to rid this about computer geeks, that it is only computer geeks that program. I want to show that you can program without being a computer geek and that this is not some kind of hocus-pocus. And that’s why […] I have progression so that all will be able to do something, that there are different levels [of programming].”
> **Quote 12**. Teacher about preconceptions of programming and programmers

5.7 Programming affords play

An identified perceived affordance in interviews is that programming affords play and that this can be used to make both programming and mathematics more playful. One teacher explains that programming is used to get students in a playful mindset before introducing mathematics (Quote 13).

> ”The first lessons are of course playful. We learn print and input [commands in Python] and then we can play with stuff such as, what are you going to eat during the weekend? And then they write. The computer can answer, I hope that you get pancakes, or whatever. Only that kind of playful stuff. Then we can start to introduce mathematics.”
> **Quote 13**. Teacher about programming as a playful introduction to mathematics

An identified constraint in the collected material is that some programming tools are perceived as just for play or used as an escape from course content. One teacher explains that there are differences between how students play with textual programming in Python and block programming at code.org. According to the teacher, students that are fascinated by Python likes the structure and becomes happy in another way than those who prefer block programming. While those who prefer block programming, view it more as a playground and escape from mathematics. (Quote 14)

> ”I see that it divides the students in two camps. Those who likes Python are those who are really fascinated about using the structure […] and to really get it to work, to write code and program. I see that there are many that becomes happy in a different way. It’s not a playground. Because there are those that, can’t we do some block programming instead? And they view it more as play.”
> **Quote 14**. Teacher about the differences of play in textual and block programming

5.8 Programming affords holistic views

In interviews, teachers explain that programming can be used to situate the Mathematics course and the Technology course in a bigger picture. That is, not only as school courses but also knowledge relevant in the real world. One teacher that uses programming in both the Mathematics course and the Technology course explains that programming with micro:bit is used to draw connections between technologies and students' everyday life. For example, mobile phones and how they work. It is further mentioned during interviews that students sometimes perceive the Mathematics course as composited of different parts that have nothing or little to do with each other. One teacher explain that the practice of reuse in programming can affect students' perception of Mathematics as a more coherent course (Quote 15).

> ”It can be a difficulty in Mathematics when you work chapter by chapter […] that you don't see that the different chapters are related. Because sometimes it feels like the students have this thought that, now we are done with geometry […]. So, you see more of these connections and not only between different areas of mathematics, but also between mathematics and programming.”
> **Quote 15**. Teacher about making the Mathematics course more coherent with programming

An identified constraint in the collected material is that some types of programming tools could be less relevant for the holistic views that programming can afford. A teacher that uses programming in the Technology course explains that because programming is a relevant skill for future labour market and digitalised society, it could be a disservice by teachers to focus too much on block programming. In future labour market and higher grades, textual programming is what students need because it affords complex problem-solving. (Quote 16)

> ”But it can feel like a disservice towards them [students] to focus too much on block programming, because when they reach grade 10-12 and study something technical, or later on the labour market, they might look at you funny. No, block programming, we don't do that here. […] In the future it's going to be textual programming that they learn. Because with that they can do more complicated, or I mean, complex things.”
> **Quote 16**. Teacher about textual programming being more important for students to learn than block programming

5.9 Programming affords fearlessness

An identified perceived affordance in interviews is that programming could afford less fear of errors among students. One teacher expresses that students are often afraid of making errors in mathematics and get a sense of panic when something is wrong. The teacher further explains that error handling is part of the programming practice and that it would be preferable if a similar mindset could be applied in the Mathematics course. (Quote 17)

> ”when you get that Error […] you get this, you know, sense of panic that something is wrong. You get that sense quite often in mathematics unfortunately, and you want to get rid of that to be able to solve a problem. […] So that you are not afraid of getting an Error […] that is something that I would

like to see influence [mathematics], because you often see programmers view it in that way. […] now it really is a culture that you should not make Errors in mathematics, but that happens all the time and that's how you learn"
Quote 17. Teacher about fear of making errors in mathematics

An identified constraint in the collected material is that programming does not automatically foster fearlessness. One teacher explains that students' practice of Error handling or troubleshooting in programming does not automatically influence their practice in mathematics. However, the practice can be drawn upon to support students that likes programming but have difficulties in the Mathematics course. (Quote 18)

"I believe that I can use that, those [students] that are at risk of failing Mathematics but likes Python and troubleshooting […] They [students] do not troubleshoot in the same way in mathematics, but I believe that I can use it to get a similar feeling."
Quote 18. Teacher about using programming practice of troubleshooting to support students in the Mathematics course

5.10 Programming affords interdisciplinary collaborations

An identified perceived affordance in interviews is that programming can afford interdisciplinary collaborations. A teacher that uses programming in both the Mathematics course and the Technology course explain that Mathematics is generally a course that is more difficult than others to achieve interdisciplinary collaborations. However, with programming it is possible to collaborate with the courses in Technology, English, Swedish and Social science. To have something like programming that can link the different courses together can create a greater incentive for students to learn and develop. (Quote 19)

"I also believe that it is a much greater incentive for the students if they know that what they do in Technology is also important for Mathematics. […] It's quite difficult to connect Mathematics interdisciplinary sometimes. […] generally it is a course that is more difficult than others to get the lesson plans to work [with other courses]. But English work very well with programming, and also Swedish if you do it the right way. […] It's all about what approach you have, and I hope that we will be able to do more of this in the future."
Quote 19. Teacher about interdisciplinary collaborations through programming

An identified constraint in the collected material is that some programming tools are perceived as childish by the students, although they might work well for interdisciplinary collaborations. A teacher that uses programming in the Technology course explains that an opportunity with block programming in Scratch is that it is easy to get other courses on board for collaborations. However, it can also be perceived as too easy and childish by the students. (Quote 20)

"Well, let's say opportunities, Scratch for example, it is to get other courses on board. Because you can use Scratch to do blocks, to do mathematical calcula-

tions, or to ask questions about Swedish lakes, or whatever it might be. If you can get other courses involved, then it is a boost you know. […] The students feel that this is, those who gets it that is, they think this is too easy you know. You can often perceive that this is a bit too childish."
Quote 20. Teacher about Scratch for interdisciplinary collaborations

6 Discussion

The key take-aways from previous research are briefly summarised here. To address failure and high dropout rates in programming education, more focus should be on problem-solving strategies and not only on syntax and semantics (Malik & Coldwell-Neilson, 2017). Problem-solving through debugging is a complex practice that requires more knowledge than how to write code (Beege et al., 2021; Liu et al., 2017). Game elements and gamification can have positive effects on student motivation in programming courses (Fotaris et al., 2016; Pilkington, 2018). Students conceptual understanding of a phenomenon can be enhanced by multiple representations (Krinks et al., 2019). Previous research on the integration of programming in Swedish primary and secondary school courses of Mathematics and Technology have highlighted different potential relationships between programming and course content (Kilhamn et al., 2021). It has further been highlighted that teachers did not feel sufficiently prepared for the integration (Vinnervik, 2022) and that programming is often perceived as fun by teachers but that there is a lack of time and directives for the integration that can be challenging (Humble, 2022; Humble et al., 2020).

In the comparison of block programming tools to textual programming tools, previous research show affordances and constraints regarding both. Block programming can support students with visualisation and understanding (Fagerlund et al., 2021; Maloney et al., 2010; Scaradozzi et al., 2019; Toma, 2021). While textual programming tools provide a more professional learning experience, which can be inspiring for students (Weintrop & Wilensky, 2017; Garneli et al., 2015; Maggiore et al., 2011). Tangible programming tools, more hands-on approach and focus on creating physical objects, can be attractive for students since it becomes less ambiguous than objects in a virtual environment (Sentance et al., 2017; Papavlasopoulou et al., 2017). Tinkering with code can be an opportunity for students to engage in inquiry, learn about rules, variables, and structures, and to break down phenomenon in manageable parts (Wagh et al., 2017; Resnick et al., 2000). However, coding on your own does not motivate all students, some require more guidance (Clarke-Midura et al., 2019).

6.1 Comparing results to previous research

Previous research highlights the importance of addressing deep learning in teaching and learning activities (Malik & Coldwell-Neilson, 2017). Malik and Coldwell-Neilson (2017) emphasise problem-solving strategies and not only syntax and semantics when learning programming. Similarly, teachers in this study highlight ways in which programming can support course content and learning in secondary school courses of Mathematics and Technology. This relationship between programming and course content was also highlighted in the study by Kilhamn et al. (2021), where programming was used in the Mathematics course for exploring and conducting calculations efficiently. In the study presented here, programming as support for course content in Mathematics and Technology were especially addressed in the perceived affordances of flexibility, creativity, efficiency, and visualisation. Programming affords flexibility in that course related concepts can be used and examined in different ways. The importance of representing a phenomenon in multiple ways to deepen students' conceptual understanding is stressed by Krinks et al. (2019). Teachers in this study used programming to provide visualisation and make course content less abstract, for example geometry and probability theory. Previous research also shows that visualisation techniques in programming can boost understanding, if used mindfully by the teacher to engage students (Mladenović et al., 2021).

Teachers in this study highlighted creativity in programming as a way of supporting course content and learning, for example that there is no single right approach to solve a problem. Teachers highlighted perceived flexibility of textual programming as an opportunity. But it was also mentioned that textual programming can be perceived as overwhelming and could therefore limit creativity. This can be related to findings in previous research, which have highlighted that teachers did not feel sufficiently prepared for the integration of programming in course content and perceived a lack of time and directives for the integration (Humble, 2022; Humble et al., 2020; Vinnervik, 2022). Block programming and Scratch is mentioned in both this study and previous research as a good approach for drawing on complex cognitive practice in programming, while still limit syntactic challenges (Fagerlund et al., 2021; Maloney et al., 2010). Challenges of syntactics in textual programming is mentioned in relation to programming affordance of efficiency in this study. It is also mentioned that once students understand the power programming can bring to a subject, they will want to use it. Which can be related to previous research, where students who work with textual programming view their experience as more professional and effective (Weintrop & Wilensky, 2017).

Teachers in the study highlight that programming can be used for engagement and motivation in the classrooms. This is addressed related to perceived affordances of fun, curiosity, and play. That programming is perceived as fun by teachers has also been addressed in related research (Humble, 2022; Humble et al., 2020). Previous research also shows that game element and gamification in programming have positive effects on students' motivation (Fotaris et al., 2016; Pilkington, 2018). Similarly, it is pointed out in this study that introducing programming should be playful and fun. It is also expressed that block programming can be too playful, and therefore used as an escape from course content. Previous research show that there can be problematic assumptions on the relationship between play and children's motivation and engagement when engaging with block programming tools (Clarke-Midura et al., 2019).

Previous research has highlighted the opportunity of tangible programming for facilitating understanding, interest, and motivation in classrooms, and making programming more attractive for K-12 students (Sentance et al., 2017; Papavlasopoulou et al., 2017). It is mentioned in this study, that programming tends to be more fun for students in the Technology course than in the Mathematics course for the same reason. Course content in Technology is more tangible and students, both girls and boys, find the act of making something happen fun. It is further mentioned that curiosity for programming can be used to spark interest for courses and reach students that previously where difficult to reach. Which can be related to the exploration of mathematics with programming as a tool in the study by Kilhamn et al. (2021). Previous research has indicated that block programming tends to afford more interest and basic understanding for programming than textual programming (Krishnamurthi & Fisler, 2019; Weintrop & Wilensky, 2017). Teachers in this study also highlighted tangible programming and textual programming libraries for visualisation as good approaches for sparking curiosity.

An interesting finding in this study was that teachers used programming to afford developmental skills, which stretched beyond course content and student motivation. This was mainly addressed related to perceived affordances of holistic views, fearlessness, and interdisciplinary collaborations. Previous research has pointed out the opportunity of using programming for engaging in inquiry, breaking down scientific phenomenon, and to make underlying structures and rules transparent (Wagh et al., 2017; Resnick et al., 2000). This can be related to the holistic views identified in this study, teachers express that programming practice of reuse can afford the Mathematics course to be perceived as more coherent by the students. Programming is used by teachers to connect courses to the outside world, which is also highlighted as an opportunity in previous research (Sentance et al., 2017). Both teachers in this study and previous research suggest that textual programming could have an opportunity over block programming in this regard, since textual programming is often perceived as authentic and professional (Weintrop & Wilensky, 2017; Garneli et al., 2015; Maggiore et al., 2011).

Debugging is a common practice in programming, which require deep understanding that goes beyond knowing how to write code (Beege et al., 2021; Liu et al., 2017). Teachers in this study describe that programming debugging, error handling and troubleshooting can afford less fear of making errors. That is, not to view errors as a failure, but as something to correct and learn from. It is also pointed out that programming practices associated with fearlessness does not necessarily influence all students and transfer to other knowledge domains. Some students need teacher's support in making the connection. Teachers further expressed that programming can be used for interdisciplinary collaboration, which was perceived as an opportunity especially for the Mathematics course. Teachers perceived that the Mathematics course is often difficult to include in collaborations with other courses. Previous research has shown positive results for project-based learning across multiple courses in K-12 education using block programming in Scratch (Sáez-López et al., 2016). Teachers in this study recognised the opportunity of getting other courses involved with programming by using Scratch. But some also highlighted the risk of block programming becoming too easy and perceived as childish by students.

6.2 Expected and unexpected findings

By discussing and comparing identified perceived affordances and related constraints with previous research, three aspects of teaching and learning emerge: A) support course content and learning, B) facilitate student engagement and motivation, and C) foster developmental skills. Compared to previous research, A and B are well established and should not be considered as unexpected findings. C, however, has not been addressed in previous research to the same extent as A and B to the author's knowledge. C is further considered an interesting and unexpected finding since it, unlike A and B, address affordances that stretch beyond the professional domains of teachers in the study. Programming affordances related to foster development skills (aspect C) has the potential of influencing students in ways that affect their learning in other courses where programming is not addressed. These findings can be used by teachers and other stakeholders as inspiration and examples for programming use in ongoing integration of programming in K-12 education.

7 Conclusion

This study has identified ten (main) perceived affordances, and related constraints, with programming for secondary school courses in Mathematics and Technology. By discussing and comparing these to previous research, three aspects of teaching and learning were found: A) support course content and learning, B) facilitate engagement and motivation, and C) foster developmental skills. These aspects and perceived affordances form a potential conceptual model of what programming can afford secondary school Mathematics and Technology (Fig. 1).

Fig. 1 Summary of findings

A. Support course content and learning
    1) Flexibility
    2) Creativity
    3) Efficiency
    4) Visualisation
B. Facilitate student engagement and motivation
    5) Fun
    6) Curiosity
    7) Play
C. Foster developmental skills
    8) Holistic views
    9) Fearlessness
    10) Interdisciplinary collaborations

This conceptual model address which aspects of secondary school Mathematics and Technology programming can support, facilitate, and foster, and through which programming affordances this can be achieved. A conceptual model, based on teacher experiences, for what programming can afford K-12 Mathematics and Technology has, to the author's knowledge, not been presented in previous research. Findings presented in this study can be viewed as inspiration for programming use in secondary school courses of Mathematics and Technology, and be used by teachers, policymakers and other stakeholders in the integration and design of programming activities for K-12 education.

8 Limitations and future research

Findings presented in this study should not be used for generalisation of teachers' perceptions of programming. The study is based on limited data and participants were selected based on their interest and experience in using programming for secondary school courses in Mathematics and Technology. Therefore, participants do not represent the 'typical' teacher. Findings should instead be viewed as suggestions for what programming can be used for in K-12 education. A next step of research that would be interesting, is a wider collection of data with multiple data sources for examining potential additional programming affordances. This could be conducted with additional interviews, surveys, and classroom observations.

Appendix: Interview guide

Introduction

1) What is your name?
2) Where do you work?
3) (If you are comfortable disclosing) how old are you?
4) What is your educational background?
5) How long have you worked as a teacher?
6) In what courses and grades do you use programming?
7) How long have you been using programming in your teaching and learning activities?

Main interview

- How do you use programming in mathematics/technology?
- What tools/programming languages do you use?
- Do you perceive any opportunities or obstacles with these tools/programming languages?

- Would you consider programming to be an integrated or isolated part of mathematics/technology?
- Does programming afford anything to your teaching or the students learning?
- Are there any specific parts of mathematics/technology content where you use programming?
- Are there any specific parts of mathematics/technology content where you consider programming appropriate?
- Are there any specific parts of mathematics/technology content where you consider programming not appropriate?
- Are there any differences and/or similarities in how you use programming in mathematics and technology?
- Are there any differences and similarities in how you use programming with different grades/students?
- Have you learned anything by using programming in mathematics/technology?
- Is there anything with programming that you consider especially good or bad for mathematics/technology?
- Do you have any programming recommendations for other teachers?

Authors' contributions Not applicable.

Funding Open access funding provided by Mid Sweden University.

Data availability Not public to protect study participants.

Code availability Not applicable.

Declarations

Declarations of interest None.

Conflicts of interest The author declare that there is no conflict of interest.

References

Adams, W. C. (2015). Conducting semi-structured interviews. In: Newcomer, K. E., Hatry, H. P., Wholey, J. S. (Ed.). *Handbook of practical program evaluation*, *4*, 492–505.

Beege, M., Schneider, S., Nebel, S., Zimm, J., Windisch, S., & Rey, G. D. (2021). Learning programming from erroneous worked-examples. Which type of error is beneficial for learning? *Learning and Instruction, 75*, 101497. https://doi.org/10.1016/j.learninstruc.2021.101497

Bower, M., & Sturman, D. (2015). What are the educational affordances of wearable technologies? *Computers & Education, 88*, 343–353. https://doi.org/10.1016/j.compedu.2015.07.013

Boyatzis, R. E. (1998). *Transforming qualitative information: Thematic analysis and code development*. Sage Publications.

Braun, V., & Clarke, V. (2012). Thematic analysis. In H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA handbook of research methods in psychology, Vol. 2. Research designs: Quantitative, qualitative, neuropsychological, and biological* (pp. 57–71). American Psychological Association.

Bryman, A. (2016). *Social research methods*. Oxford University Press.

Campbell, S., Greenwood, M., Prior, S., Shearer, T., Walkem, K., Young, S., ... & Walker, K. (2020). Purposive sampling: complex or simple? Research case examples. *Journal of Research in Nursing*, *25*(8), 652-661. https://doi.org/10.1177/2F1744987120927206

Chong, I., & Proctor, R. W. (2020). On the evolution of a radical concept: Affordances according to Gibson and their subsequent use and development. *Perspectives on Psychological Science, 15*(1), 117–132. https://doi.org/10.1177/2F1745691619868207

Clarke-Midura, J., Lee, V. R., Shumway, J. F., & Hamilton, M. M. (2019). The building blocks of coding: A comparison of early childhood coding toys. *Information and Learning Sciences, 120*(7/8), 505–518. https://doi.org/10.1108/ILS-06-2019-0059

Etikan, I., Musa, S. A., & Alkassim, R. S. (2016). Comparison of convenience sampling and purposive sampling. *American journal of theoretical and applied statistics, 5*(1), 1–4. https://doi.org/10.11648/j.ajtas.20160501.11

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with scratch in primary schools: A systematic review. *Computer Applications in Engineering Education, 29*(1), 12–28. https://doi.org/10.1002/cae.22255

Fossey, E., Harvey, C., McDermott, F., & Davidson, L. (2002). Understanding and evaluating qualitative research. *Australian & New Zealand Journal of Psychiatry, 36*(6), 717–732. https://doi.org/10.1046/2Fj.1440-1614.2002.01100.x

Fotaris, P., Mastoras, T., Leinfellner, R., & Rosunally, Y. (2016). Climbing up the leaderboard: An empirical study of applying gamification techniques to a computer programming class. *Electronic Journal of e-Learning, 14*(2), 94–110.

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

Gibson, J. J. (1977). The theory of affordances. *Hilldale, USA, 1*(2), 67–82.

Gibson, J. J., & Carmichael, L. (1966). *The senses considered as perceptual systems* (Vol. 2, No. 1, pp. 44–73). Houghton Mifflin.

Grover, S., & Basu, S. (2017). 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).

Hammond, M. (2010). What is an affordance and can it help us understand the use of ICT in education? *Education and Information Technologies, 15*(3), 205–217. https://doi.org/10.1007/s10639-009-9106-z

Heft, H. (2001). *Ecological psychology in context: James Gibson, Roger Barker, and the legacy of William James's radical empiricism*. Psychology Press.

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

Humble, N. (2022). Teacher observations of programming affordances for K-12 mathematics and technology. *Education and Information Technologies, 27*(4), 4887–4904. https://doi.org/10.1007/s10639-021-10811-w

Humble, N., Mozelius, P., & Sällvin, L. (2020). Remaking and reinforcing mathematics and technology with programming – Teacher perceptions of challenges, opportunities and tools in K-12 settings. *The International Journal of Information and Learning Technology, 37*(5), 309–321. https://doi.org/10.1108/IJILT-02-2020-0021

Humble, N., Mozelius, P., & Sällvin, L. (2019). Teacher challenges and choice of programming tools for teaching k-12 technology and mathematics. In *International Conference on Education and New Developments (END 2019), Porto, Portugal, 22–24 June, 2019* (Vol. 1, pp. 431–435). inScience Press.

Kilhamn, C., Rolandsson, L., & Bråting, K. (2021). Programmering i svensk skolmatematik: Programming in Swedish school mathematics. *LUMAT: International Journal on Math, Science and Technology Education, 9*(1), 283–312. https://doi.org/10.31129/LUMAT.9.2.1457

Krinks, K. D., Sengupta, P., & Clark, D. B. (2019). Modeling games in the K-12 science classroom. *International Journal of Gaming and Computer-Mediated Simulations (IJGCMS), 11*(1), 31–50. https://doi.org/10.4018/IJGCMS.2019010103

Krishnamurthi, S., & Fisler, K. (2019). Programming paradigms and beyond. In: S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research*, (pp. 377–413). Cambridge University Press.

Lindberg, O. J., Olofsson, A. D., & Fransson, G. (2017). Same but different? An examination of Swedish upper secondary school teachers' and students' views and use of ICT in education. *The International Journal of Information and Learning Technology*. https://doi.org/10.1108/IJILT-09-2016-0043

Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education, 27*(1), 1–29. https://doi.org/10.1080/08993408.2017.1308651

Maggiore, G., Torsello, A., Sartoretto, F., & Cortesi, A. (2011). Engaging high school students in computer science via challenging applications. In *Proceedings of the 2011 conference on Information technology education* (pp. 43–48).

Malik, S. I., & Coldwell-Neilson, J. (2017). A model for teaching an introductory programming course using ADRI. *Education and Information Technologies, 22*(3), 1089–1120. https://doi.org/10.1007/s10639-016-9474-0

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Maxwell, J. A. (2008). Designing a qualitative study. *The SAGE Handbook of Applied Social Research Methods, 2*, 214–253.

Mladenović, M., Žanko, Ž, & Aglić Čuvić, M. (2021). The impact of using program visualization techniques on learning basic programming concepts at the K–12 level. *Computer Applications in Engineering Education, 29*(1), 145–159. https://doi.org/10.1002/cae.22315

Murai, Y., & Muramatsu, H. (2020). Application of creative learning principles within blended teacher professional development on integration of computer programming education into elementary and middle school classrooms. *Information and Learning Sciences, 121*(7/8), 665–675. https://doi.org/10.1108/ILS-04-2020-0122

Norman, D. A. (1999). Affordance, conventions, and design. *Interactions, 6*(3), 38–43. https://doi.org/10.1145/301153.301168

Norman, D. (2013). *The design of everyday things: Revised and* (expanded). The MIT Press.

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

Papavlasopoulou, S., Giannakos, M. N., & Jaccheri, L. (2017). Reviewing the affordances of tangible programming languages: Implications for design and practice. In *2017 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1811–1816). IEEE.

Pilkington, C. (2018). A playful approach to fostering motivation in a distance education computer programming course: Behaviour change and student perceptions. *The International Review of Research in Open and Distributed Learning*, *19*(3). https://doi.org/10.19173/irrodl.v19i3.3664

Pörn, R., Hemmi, K., & Kallio-Kujala, P. (2021). Inspiring or confusing–a study of Finnish 1–6 teachers' relation to teaching programming. *LUMAT: International Journal on Math, Science and Technology Education, 9*(1), 366–396. https://doi.org/10.31129/LUMAT.9.1.1355

Resnick, M., Berg, R., & Eisenberg, M. (2000). Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. *The Journal of the Learning Sciences, 9*(1), 7–30. https://doi.org/10.1207/s15327809jls0901_3

Royal Society. (2017). *After the reboot: Computing education in UK schools*. Policy Report.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education, 97*, 129–141. https://doi.org/10.1016/j.compedu.2016.03.003

Scaradozzi, D., Screpanti, L., Cesaretti, L., Storti, M., & Mazzieri, E. (2019). Implementation and assessment methodologies of teachers' training courses for STEM activities. *Technology, Knowledge and Learning, 24*(2), 247–268. https://doi.org/10.1007/s10758-018-9356-1

Schwandt, T. A., Lincoln, Y. S., & Guba, E. G. (2007). Judging interpretations: But is it rigorous? Trustworthiness and authenticity in naturalistic evaluation. *New Directions for Evaluation, 2007*(114), 11–25. https://doi.org/10.1002/ev.223

Sentance, S., Waite, J., Hodges, S., MacLeod, E., & Yeomans, L. (2017). Creating Cool Stuff" Pupils' Experience of the BBC micro: bit. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 531–536).

Smit, S., Tacke, T., Lund, S., Manyika, J., & Thiel, L. (2020). *The future of work in Europe*. McKinsey Global Institute.

Szabo, C., Sheard, J., Luxton-Reilly, A., Becker, B. A., & Ott, L. (2019). Fifteen years of introductory programming in schools: a global overview of K-12 initiatives. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (pp. 1–9). https://doi.org/10.1145/3364510.3364513

Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Computers & Education, 162*, 104083. https://doi.org/10.1016/j.compedu.2020.104083

Toma, R. B. (2021). Measuring acceptance of block-based coding environments. *Technology, Knowledge and Learning*, 1–11. https://doi.org/10.1007/s10758-021-09562-x

Tran, Y. (2018). Computer programming effects in elementary: Perceptions and career aspirations in STEM. *Technology, Knowledge and Learning, 23*(2), 273–299. https://doi.org/10.1007/s10758-018-9358-z

Viberg, O., Grönlund, Å., & Andersson, A. (2020). Integrating digital technology in mathematics education: a Swedish case study. *Interactive Learning Environments*, 1–12. https://doi.org/10.1080/10494820.2020.1770801

Vinnervik, P. (2022). Implementing programming in school mathematics and technology: Teachers' intrinsic and extrinsic challenges. *International Journal of Technology and Design Education, 32*, 213–242. https://doi.org/10.1007/s10798-020-09602-0

Wagh, A., Cook-Whitt, K., & Wilensky, U. (2017). Bridging inquiry-based science and constructionism: Exploring the alignment between students tinkering with code of computational models and goals of inquiry. *Journal of Research in Science Teaching, 54*(5), 615–641. https://doi.org/10.1002/tea.21379

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Education and Information Technologies, 22*(2), 445–468. https://doi.org/10.1007/s10639-016-9493-x

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE), 18*(1), 1–25. https://doi.org/10.1145/3089799