Computers & Education 141 (2019) 103607

Contents lists available at ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

A systematic review of learning computational thinking through Scratch in K-9

LeChen Zhang*, Jalal Nouri

Department of Computer and Systems Sciences, Stockholm University, Borgarfjordsgatan 12, SE-16407, Kista, Sweden

ARTICLE INFO

Keywords:
Computational thinking
Programming
Scratch
K-9
Systematic review

ABSTRACT

As computational thinking (CT) is being embraced by educational systems worldwide, researchers and teachers have posed important questions such as "what to teach" and "what can be learned." These questions are universally crucial to the learning of all subjects. Nevertheless, there is no up-to-date, systematic overview of CT education for K-9 students that attempt to provide answers to these crucial questions. Thus, this systematic review presents a synthesis of 55 empirical studies, providing evidence of the development of computational thinking through programming in Scratch, one of the most popular visual block programming languages in schools. The purpose of this review is to systematically examine the CT skills that can be obtained through Scratch in K-9 based on empirical evidence. This systematic review has adopted Brennan and Resnick's (2012) framework as the basis for defining and identifying the expected CT skills in K-9. The major findings entail what computational thinking skills students in K-9 can learn through Scratch in relation to the framework mentioned above, taking the progression of learning into account. Additional CT skills that are not captured by the framework were identified including input/output, reading, interpreting and communicating code, using multimodal media, predictive thinking, and human-computer interaction. These additional CT skills are not currently presented in Brennan and Resnick's (2012) framework and can be considered as possible supplements to their framework. Furthermore, the paper discusses the difficulties regarding assessment and the progression of the identified skills, as well as problems with study designs. Finally, the paper sets out suggestions for future studies based on the current research gaps.

1. Introduction

Computational thinking (CT) is a 21st century skill which future generations must develop. This belief has been internationally acknowledged and an increasing number of educational systems have integrated CT into their compulsory education in recent years. Heintz, Mannila and Färnqvist's (2016) and Mannila et al.'s (2014) reviews presented updated curricula, country by country. Unlike other well-established subjects (such as mathematics and history), which have a unified and unambiguous name throughout the world, different countries have adopted different terms for this new subject, e.g. "ICT capability" in Australia and "digital competence" in Sweden. The four prevalent topics in these documents are coding, programming, computing and CT (Fig. 1). Usually, these are perceived as having blurred boundaries and they are sometimes mistaken as synonyms. Coding and programming are often interchangeable in colloquial use (Duncan, Bell, & Tanimoto, 2014). The main advantage of using the term "coding" is that it attracts interest because there is a hint of a secret code and achievement in cracking the code (Duncan et al., 2014). Coding can be referred to

* Corresponding author.
E-mail addresses: chen@dsv.su.se (L. Zhang), jalal@dsv.su.se (J. Nouri).

https://doi.org/10.1016/j.compedu.2019.103607
Received 6 September 2018; Received in revised form 14 June 2019; Accepted 16 June 2019
Available online 22 June 2019

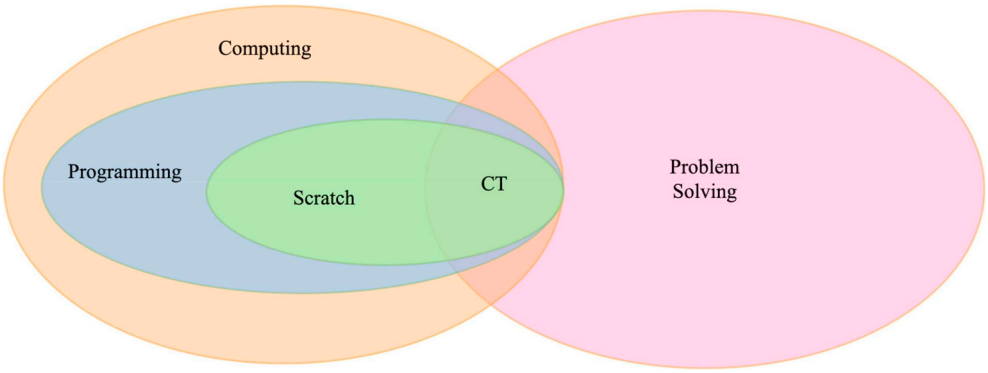

Fig. 1. The relationship among the topics.

as the writing of computer programming code. Programming itself, however, is more than just coding (Lye & Koh, 2014). Originally, as Blackwell (2002) explained in his work "what is programming", coding was considered to be a subtask of programming, which involves the complex task of understanding a problem, in addition to designing, coding and maintenance. Computing is an even larger concept. "The core of computing is computer science, in which pupils are taught … how to put this knowledge to use through programming" (Department of Education UK, 2015). It seems that everything is connected through programming. Scratch is just one drop in the bucket of programming tools. These programming tools are seen as a means of developing CT (e.g. Berland & Wilensky, 2015; Brennan & Resnick, 2012; Wilson & Moffat, 2010), as well as digital competency and 21st century skills (Nouri et al., 2019), and are defined in more detail later in the text. Nevertheless, CT can be taught without these programming tools: so-called "programming unplugged" (Bell, Alexander, Freeman, & Grimley, 2009).

Fuelled by all the attention on CT, many empirical studies have looked at CT education in K-9 (Kindergarten to Grade 9), answering important questions such as "what to teach" (Grover & Pea, 2013) and "what can be learned" (Wing, 2006). These questions are universally crucial to the learning of all subjects. Nevertheless, there is no up-to-date, systematic overview of empirical studies that address "what to teach" and "what can be learned" in CT education for K-9 students. The definition of CT varies between the curricula of different countries, as it does in the literature (see Table 1). Although many curricula declare "what to teach," and they may be perfectly supported by meticulous theoretical reasoning, the content of CT has not been examined in practice systematically. Thus, the "what to teach" in the curricula may not match "what can actually be learned." Therefore, it is reasonable to investigate what content/components of CT have been assessed in younger children's learning, i.e., "what can be learned" as a starting point to answer "what to teach." The latest systematic review was conducted by Lye and Koh (2014). Certainly, their work provided important insights into learning CT through programming for K-12. However, the area that is in dire need of mapping is the compulsory setting (K-9). Heintz, Mannila, and Färnqvist (2016) and Mannila et al. (2014) state that, historically, programming has

Table 1
Summary of CT frameworks.

	Barr and Stephenson (2011)	Brennan and Resnick (2012)	Selby (2012)	Grover and Pea (2013)	Seiter and Foreman (2013)	Kalelioglu, Gülbahar, and Kukul (2016)	Angeli et al. (2016)	Repenning, Basawapatna and Escherle (2016)
Abstraction	✓	✓	✓	✓	✓	✓	✓	✓
Algorithm	✓	✓	✓	✓	✓	✓	✓	
Data	✓	✓		✓	✓	✓		
Decomposition	✓	✓	✓	✓	✓	✓	✓	
Parallelisation	✓	✓		✓	✓	✓		
Testing & Debugging	✓	✓		✓		✓		✓
Control structure	✓	✓		✓		✓		
Automation						✓		✓
Generalisation			✓			✓	✓	
Simulation	✓		✓			✓		
Event		✓						
Being incremental & iterative		✓						
Expressing & connecting & questioning		✓						
Reusing & remixing		✓						
Efficiency & performance constraints				✓				
Systematic processing				✓				
Conceptualising						✓		

been available to upper secondary students (Grade 10–12), either as a compulsory or a selective course, depending on the study programme. Hence, all of the aforementioned updated curricula focus on programming in K-9, as CT was simply absent. In Lye and Koh's (2014) review, only a quarter of the studies were carried out in a K-12 setting, with the majority being undertaken in higher education. What is more, the landscape and focus of research may have changed tremendously since then. To sum up, this review intends to address the research gap and determine what CT skills can be obtained by K-9 students.

The review focused exclusively on Scratch for the following reasons. First of all, Scratch is used more frequently than any other similar software (Price & Barnes, 2015; Wong, Cheung, Ching & Huen's, 2015). As of January 2018, it ranks the highest in visual block programming languages on the TIOBE index (2018), which measures programming languages' popularity based on search engine results. More importantly, Scratch has a theoretical foothold in Brennan and Resnick's (2012) CT framework. As a matter of fact, this framework guided the design of the next generation of Scratch: Scratch 2.0 (Brennan & Resnick, 2013). Moreover, Scratch possesses certain advantages when it comes to young learners. It provides "lower the floor" programming so that children can get started earlier (Maloney, Resnick, Rusk, Silverman & Eastmond, 2010). Scratch has a great number of young users, starting from four years of age (https://scratch.mit.edu/statistics). Other similar languages, such as Alice, "is best suited for those students who are already coming across problems in other STEM … and have sufficient background to be able to tackle a more rigorous approach toward problem solving". Greenfoot is better for students older than 14 years (Utting, Cooper, Kölling, Maloney & Resnick, 2010, pp. 2–3).

"CT skills" is a terminology that often appears in CT research (e.g., Atmatzidou & Demetriadis, 2016; Voogt, Fisser, Good, Mishra & Yadav, 2015; Basawapatna, Repenning, Koh, & Nickerson, 2013). However, the definitions of CT and CT skills vary. So far, there are three categories of definitions present in the literature (Román-González, Pérez-González & Jiménez-Fernández, 2017). Firstly, the generic definitions such as Wing's (2011) and Aho's (2012) focus on the thought process for problem-solving that leads to a solution of computational steps or algorithm. Therefore, the CT skills underlying this definition can be somewhat general as "a universally applicable skill set" just like reading, writing, and arithmetic. Second, the operational definitions define CT as a problem-solving process provided by the Computer Science Teachers Association (CSTA, 2011). Its corresponding CT skills can be characterized as formulating, organizing, analyzing, automating, presenting, implementing, and transferring. The third category is the educational definitions, which is presented in Table 1 and discussed in details in the next chapter due to their relevance to this study. Based on the common core of these three types of definitions, i.e., *problem-solving*, this study defines CT as a thought process, through skills that are fundamental in programming (CT skills), to solve problems regardless of discipline. More specifically, we have adopted Brennan and Resnick's (2012) framework as the basis for delineating CT skills. Problem-solving is incorporated in their framework and represented by three categories of CT skills that are fundamental for programming, namely: CT concepts, practices, and perspectives. The framework of Brennan and Resnick's (2012) has been proposed to be a suitable framework for conceptualising CT skills that can be developed through block-based programming languages such as Scratch (Brennan & Resnick, 2012; Chen et al., 2017; Sáez-López, Román-González & Vázquez-Cano, 2016). Furthermore, as mentioned above, Scratch has a theoretical foothold in Brennan and Resnick's framework.

As a matter of fact, there are several other motivations for choosing Brennan and Resnick's work as basis for identifying the CT skills. This framework is considered as providing "a wide coverage of CT" (Kong, 2016, p. 379). Many curricula emphasize the learning of basic CT through mastering what Brennan and Resnick (2012) term 'computational thinking concepts' (Falloon, 2016). Their framework has also been widely discussed and laid out as a basis for previous empirical and theoretical studies (Lye & Koh, 2014; Zhong, Wang, Cheng & Lee, 2016). More importantly, "Brennan and Resnick's framework concentrates primarily on the sort of knowledge used, and how it is used, when students create code rather than general thinking skills" (Falloon, 2016, p. 578), which serves the purpose of the study. Nevertheless, several questions arise regarding its application. First, even though it was Brennan and Resnick's endeavour that gave birth to Scratch, does that mean their framework captures all the CT skills that can be taught through Scratch? What is the validity for teachers to engage this framework in their teaching? Does Scratch deliver any CT skills that were not foreseen by the framework? Questions such as these are the reason why the framework only serves as a starting point for identifying CT skills.

The purpose of this review is to obtain a better understanding of "what to teach" and "what can be learned" through Scratch by systematically examining the CT skills that can be obtained through Scratch in K-9, asking the research question: what CT skills can be obtained through Scratch for K-9 learners, given the empirical evidence?

The remainder of the paper is structured as follows. The background section offers a brief introduction to the history of CT, a summarized view of the definition of CT and the obstacles to CT education. The methodology section describes the procedure for conducting this review. The results section presents a quantitative and qualitative analysis of the selected studies. Finally, the discussion lays out suggestions for future studies, based on current research gaps.

2. Background

The concept of CT was originally introduced in Seymour Papert's book "Mindstorms: Children, Computers, and Powerful Ideas" (1980) with his creation of the programming language LOGO. Papert referred to CT primarily as the relationship between programming and thinking skills. He believed that students' constructions through programming with LOGO could facilitate their procedural thinking across multiple disciplines. Unlike Papert, many definitions of CT in the 21st century emphasize the concepts that are commonly engaged in programming or computer science. Jeanette Wing, the researcher who brought CT back to public attention in 2006, renewed its definition with tremendous influence (3467 citations upon the drafting of this review). Wing (2006) defines CT as "solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (p. 33) and argues that CT, "just like reading, writing, and arithmetic, should be added to every child's analytical ability".

Cuny, Snyder, and Wing (2010) then updated the definition, stating "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (p. 1). In both cases, the vision is that "everyone can benefit from thinking like a computer scientist" (Cuny, Snyder, & Wing, 2010 p. 1).

Extensive research over recent years has focused on issues relating to teaching and learning skills and the concepts and practices relevant to CT (Grover & Pea, 2013). In practice, many countries have taken action and included CT in their curricula, for example, the new computing curriculum in England and an updated curriculum with a strong focus on CT in Australia (Bower et al., 2017). Despite the rising recognition of CT, "there is little agreement on what CT encompasses" (Brennan & Resnick, 2012, p. 2). The CT literature has not reached a consensus on explaining what CT is (Kalelioğlu & Gülbahar, 2014). The lack of a universal definition (Weintrop et al., 2016) has caused problems in CT education. The definition of CT is not a fixed constant yet. For example, in 2010 Wing updated her earlier definition of CT from 2006. Teachers and researchers have engaged programming tools in training abilities other than those we deem to be CT skills. As Selby and Woollard (2013) point out, the design of a consistent curriculum and appropriate assessment is out of the question without knowing what CT entails. Based on the core concepts of CT provided by computer scientists such as Wing, several definitions have emerged for what CT involves in schooling at K-12 (Voogt et al., 2015). Rose, Habgood and Jay's (2017) summary of the definitions of CT, complemented by Selby's (2012) model, is given in Table 1. Due to the scope of this study (i.e. CT across disciplines in K-12 education), the table does not present all existing definitions of CT. For instance, Weintrop et al.'s (2016) work embedded CT mainly in a mathematical and scientific context. The College Board's (2013, pp. 7–8) scheme of computer science education for universities is also not presented.

Based on Table 1, different sources define CT by offering a list of concepts and competencies, which support and differ from each other. As Rose et al. (2017) pointed out, the most common skills concern abstraction, algorithms, data, problem decomposition, parallelism, debugging and testing and control structure. Thus, this list offers this study a guideline regarding the content of CT skills in education.

In parallel to work on creating frameworks, the creation of CT learning tools also promotes computational competencies in a K-12 context (Voogt et al., 2015). Scratch, which was developed in MIT's Media Lab in 2007, is one of the most popular programming languages. Scratch provides a foothold to Brennan and Resnick's framework (see details below), which is likely to be suitable for considering computational thinking for programming contexts in K-12 education (Lye & Koh, 2014).

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

- CT concepts: the concepts designers engage with as they program, i.e. sequences, loops, parallelism, events, conditionals, operators and data;
- CT practices: 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;
- CT perspectives: the perspectives designers form about the world around them and about themselves, i.e. expressing, connecting and questioning.

3. Methodology

This review followed Kitchenham and Charters' (2007) guideline for performing a systematic literature review and was carried out through five phases: research/question/definition, search strategy design, study selection, data extraction and data synthesis.

3.1. Research question

Based on the aforementioned rationale, it is necessary for researchers and teachers to renew their understanding of the empirical evidence on developing CT skills through Scratch. Hence, this review explores the following research question:

What CT skills can be obtained through Scratch for K-9 learners, given the empirical evidence?.

3.2. Search strategy

The search was executed in databases that are well-known and well established in the field of computing: ACM (Association for Computing Machinery), ERIC (Education Resources Information Center), IEEE (Institute of Electrical and Electronics Engineers), ScienceDirect, SpringerLink and Web of Science.

The search terms were constructed by Boolean logic as follows: ((teach OR teaching OR learn OR learning OR education) AND (computational thinking)) AND (code OR coding OR program OR programming) AND (Scratch) AND (school OR K-9).

In a pilot search, it appeared that the search engines of different databases use different syntax for search strings. Therefore, the search terms were adjusted to accommodate different databases.

3.3. Study selection

The selection process consisted of two stages.

Table 2
Selection criteria.

Inclusion criteria	Exclusion criteria
Kindergarten and compulsory education (1st to 9th grade).	Other stages of education, such as higher education or students older than 16 years. Students with special needs, such as Autism or mathematically talented students. Teacher training-related work.
Empirical studies.	Theoretical work, such as frameworks, reviews, etc.
Assessments of CT skills developed using Scratch.	Studies solely focusing on other visual programming languages.
Studies providing empirical evidence for CT skills separately.	Studies not providing information or content on assessment. Studies assessing general skills or transferred skills rather than focusing on the development of specific CT skills.

3.3.1. Selection stage 1

Stage 1 was a preliminary screening, focusing on three exclusion criteria.

- As the first version of Scratch was released in 2007, the search covered the period from 2007 to January 2018. Studies outside this range were excluded.
- Studies not written in English were excluded.
- Studies without an abstract or in forms other than a paper (such as a poster, presentation, idea paper, etc.) were excluded.

Otherwise, the study was downloaded for further selection.

3.3.2. Selection stage 2

This stage applied the following selection criteria to identify studies relevant to the research questions (See Table 2).

This systematic review aimed to investigate students' development of CT skills through the use of Scratch. Only studies providing empirical evidence were accepted. However, the definition of “middle school” and “high school” vary from land to land, sometimes even among different regions within one country. A high school in one region may correspond to middle school in another. To avoid ambiguity, this review looked beyond the stages of schooling and admitted research conducted in kindergarten and up to the 9th grade. Alternatively, the maximum age of the participants in the studies was set at 16 years. Although Scratch can be used by any age group, it is primarily intended for users younger than 16. Studies lacking information regarding age were excluded. Moreover, studies were excluded if they did not specify the development of each CT skill, but rather provided an overall performance evaluation. Also, assessing general skills, such as transferred effects in problem-solving etc., was not the primary focus of this study.

The studies that were included had to have engaged Scratch (Scratch Jr.) as the means of fostering CT skills. As stated in the title, this review exclusively concerns Scratch. Any papers solely investigating other, similar, visual programming languages, such as Alice, were excluded. Furthermore, the review required the studies included to have assessed the effect of using Scratch on the development of CT skills. Studies that did not provide an assessment of the acquisition of CT skills, but rather other aspects such as self-efficacy, self-confidence, acceptance of Scratch, etc., were excluded. In short, an eligible study for data extraction contained empirical evidence of developing CT skills through Scratch.

The search terms in the databases generated 1956 articles. The screening in Stage 1 excluded 1439 articles and 517 articles remained. Then 9 articles were removed due to duplication. 508 articles entered Stage 2. The application of inclusion and exclusion criteria eliminated 452 articles, leaving 55 eligible studies (see Fig. 2).

3.3.3. Critical appraisal of evidence

Several criteria were employed to ensure the quality of the studies included. First of all, the search was performed in databases known for studying technology in education. Second, the journals that published the studies had to require peer review. Third, the authors of this article reviewed the methodology of the studies. Based on consensus, those studies missing the important description of the methodology were excluded.

3.4. Data analysis

This review adopted Elo and Kyngäs' (2008) inductive content analysis to identify the CT skills obtained by using Scratch in the selected studies. The steps are: selecting the unit of analysis, making sense of the data and the whole, open coding, coding sheets, grouping, categorisation, abstraction, and conceptual mapping. Two coders performed a pilot analysis on five papers together in order to reach agreement on the semantics of “CT skills”. Despite the inductive nature of this analysis, the coders used the common CT concepts listed in the background section as a reference. Open coding allowed the possibility of collecting, analysing and categorising other assessed skills.

3.5. Data extraction

An Excel form was designed to aid data extraction (Table 3). Each study was analysed to derive these data, most of which are

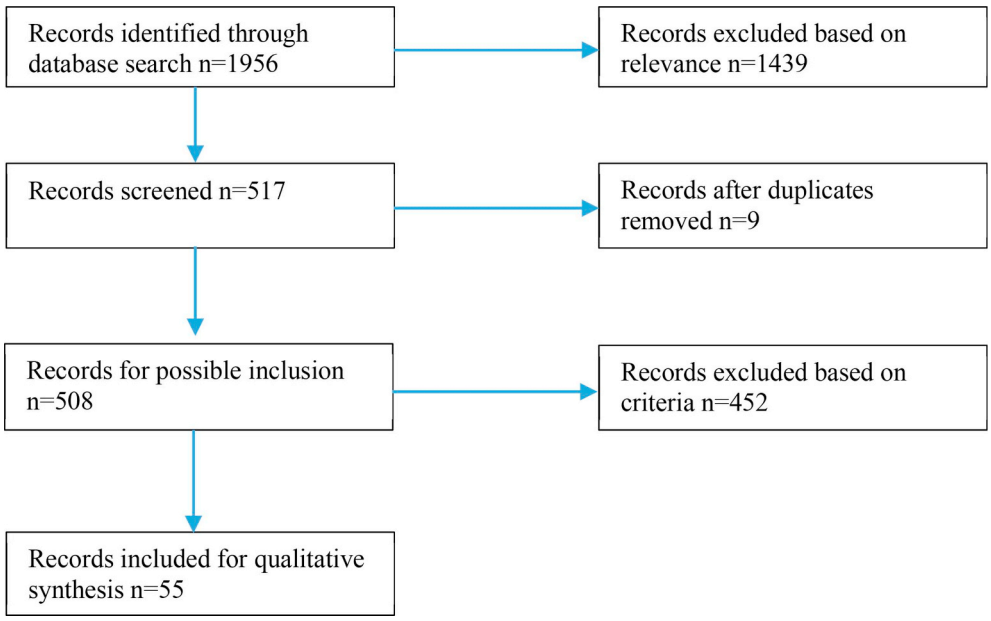

Fig. 2. Review process.

briefly presented in the results section. Location and age are given in Appendix A. The analysis primarily focused on the assessed CT skills.

3.6. Limitations of the study

The validity of the review may be threatened by two factors:

- study selection and data extraction. To mitigate this threat, the authors collaborated closely to ensure high interrater reliability. The two coders selected studies independently and compared the results. The studies of discordance were rated together by the two coders. Discussions were carried out until decisions of inclusion/exclusion were reached. Regarding data extraction, the two coders first constructed the same coding scheme together. The coding results were compared, and discussions were carried out to ensure consensus. Only codes of agreement were allowed entry to the final coding table. Based on the coding table, the coders sat together to categorise the CT skills. All disagreement was documented and discussed until consensus was reached. In the end, the inter-coder reliability in the study resulted in a 99% rating agreement. Furthermore, this study made a comparison with previous systematic reviews for possible study inclusion. Another factor that may have affected the study selection is that the systematic review only conducted searches in six large databases; it might not have presented all works in the area. What is more, only literature in English was eligible.
- publication bias. Studies with positive results are more likely to be published than negative ones (Kitchenham & Charters, 2007). Although there are few articles with negative/neutral outcomes in this study, there is a risk that the overview does not offer a complete picture of CT development through Scratch.

4. Results

4.1. Distribution of studies

4.1.1. Distribution by database

Table 4 shows the number of papers found in each database. The search string generated 1956 items after selection stage 1. In total, 508 studies were considered relevant based on their abstracts. The application of the exclusion criteria finally resulted in 55 selected studies. Non-empirical studies and studies with participants older than 16 were the two main reasons for exclusion.

4.1.2. Distribution by publication year

The number of publications appears to have increased throughout the years since the release of Scratch (Fig. 3). A steady increase can be seen from 2010 until 2017, with a decline in 2016. There were no eligible studies found from 2007 to 2009. This is in accordance with the statistics of "monthly new user" (Scratch statistics, 2018). During the first three years, the new users ranged from approximately 1000 to 15,000 per month, whereas there were almost 1,000,000 new users in May 2017 alone.

Table 3
Coding sheet.

Author	Title of publication	Year of publication	Location of the empirical study	Subjects	Age/grade of participants	Number of participants	Instruments and artefacts used to assess CT	Analysis methods	CT skills assessed	Scratch artefacts

4.1.3. Distribution by country

The United States (US) seems to be the most productive country, comprising 43% of the studies. Spain produced 13% and the United Kingdom (UK) 7% (Fig. 4). As MIT was the birth place of Scratch, 46% of Scratch users are located in the US (https://scratch.mit.edu/statistics/), which might explain the dominant status of American studies.

4.1.4. Distribution by sample size

As shown in Fig. 5, the sample size varies from fewer than 20 participants up to more than 100. Three studies did not specify the sample size. There were studies, such as Moreno-León, Robles and Román-González (2015), which used Dr. Scratch, an automated assessment tool, to examine a large number of Scratch artefacts (approximately 2500 Scratch projects). Unfortunately, these studies did not specify the user age and, hence, were excluded.

4.1.5. Distribution by subject areas

The majority of the studies (63%) implemented Scratch in computer science (CS) or programming courses (Fig. 6). The second most common way of integrating Scratch was through language teaching and learning. Overall, 9% of the studies experimented on the combination of Scratch and mathematics.

4.1.6. Distribution by scratch artefacts

Of the studies included, 41% produced Scratch games and 18% created animation or storytelling. Another 24% allowed participants to choose the category of product on their own. Besides games, animation and storytelling, products such as picture collage, art/music projects, web adverts, etc. were found in the literature (Fig. 7).

4.1.7. Distribution by instrument or artefacts for CT skills assessment

Qualitative, quantitative and mixed methodologies were present in the research for assessing CT skills (Fig. 8). It seems equally popular to assess CT skills through qualitative (e.g. observation, field notes, video content analysis) and quantitative (e.g. pre/post-test) methods. Moreover, the automated assessment tool, Dr. Scratch (in four included studies), was engaged in counting the number of CT skills in the coding.

Scratch code, observation and pre/post-tests were the top three analytic instruments (Fig. 9), with 61% of the studies examining the Scratch code in students' creations. The Scratch code was analysed both quantitatively and qualitatively. The qualitative assessment of code evaluates the variation and depth of CT, such as distinguishing simple loops from conditional loops or nested loops. Certain theoretical models (e.g. Bloom's cognitive taxonomy, Brennan and Resnick's (2012) framework, lesson objectives/curriculum goals, etc.), were used as a reference in the qualitative assessments. In quantitative assessment, a score-based pre/post-test measure often provided a statistical analysis. Regarding statistical analysis, both descriptive and inferential statistics were employed (Fig. 10). Different types of inferential statistical tools were present in the studies, including *t*-test, ANOVA, etc. (Fig. 10).

4.2. Development of CT skills through scratch

Table 5 presents the CT abilities that were identified. In total, 19 categories were identified in the studies and all of the CT skills from Brennan and Resnick's (2012) framework were found. Moreover, six additional categories of CT skills were assessed in the studies.

The results are divided into two parts. First, the identified skills that are covered by Brennan and Resnick's framework (2012) are presented. Then, the CT skills not addressed by the framework are grouped to form new categories based on inductive content analysis (Elo & Kyngäs, 2008).

Although not demonstrated in Table 5, 11 articles assessed problem-solving ability and were excluded. The reason for this is that problem solving itself is not considered to be part of the CT skills but as being a general skill that is supported by CT skills. For instance, the new Computing At School (CAS, p.8) curriculum in England argues that “Computational thinking shouldn't be seen as just a new name for ‘problem-solving skills” but it can help problem solving.

4.2.1. CT skills within Brennan and Resnick's framework

The selected studies examined all the skills in Brennan and Resnick's (2012) framework. As demonstrated in Table 6, some skills were more frequently developed than others. In general, CT concepts, such as loops (28 studies), sequences (26 studies) and conditionals (24 studies), were most frequently identified. In CT practices, abstracting and modularising (9 studies) and testing and debugging (8 studies) appear more often, while perspectives are the least frequently identified. For example, questioning was only explicitly assessed twice in the literature. As Brennan and Resnick (2012) pointed out, concepts are more assessment-friendly than practices and perspectives. This may explain the high frequency of the assessment of concepts in the studies.

Table 4
Distribution of studies in each database.

Database	Number of articles identified	Number of included articles
ACM	139	24
ERIC	17	4
IEEE	727	7
ScienceDirect	171	2
SpringerLink	860	13
Web of Science	42	5
Total	1956	55

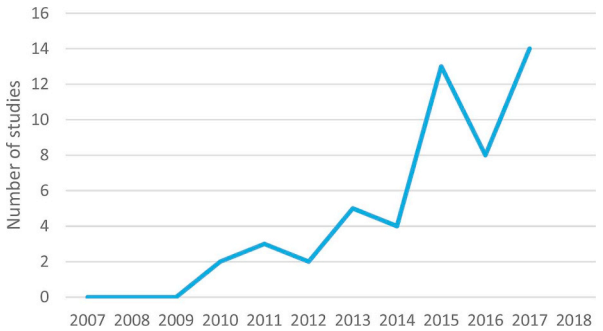

Fig. 3. Number of publications from 2007 to 2017. Note: To avoid possible misinterpretation, the three studies found in 2018 are not included in this chart as the search was conducted in January 2018.

It is necessary to note that the following text is a collection of how CT skills were assessed and the assessment results in the experiments conducted. These should not be compared directly as the experiments were not conducted under similar circumstances, differing in terms of teaching methods, time, materials and students' previous knowledge of computing, as well as their maturity and sophistication, etc.

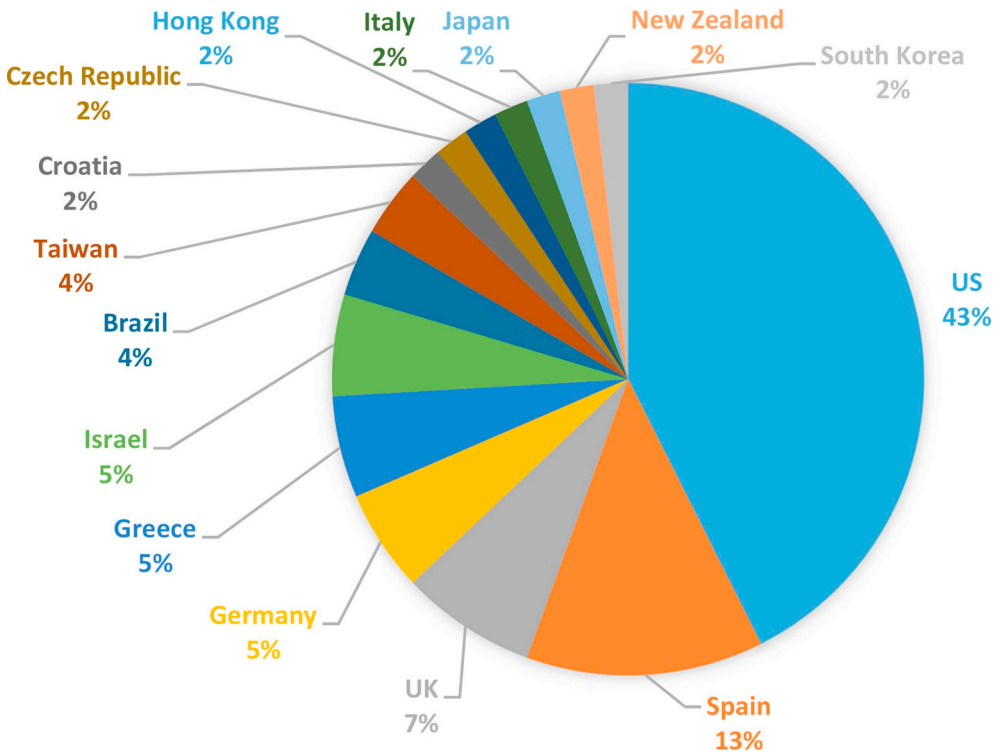

Fig. 4. Number of publications by country.

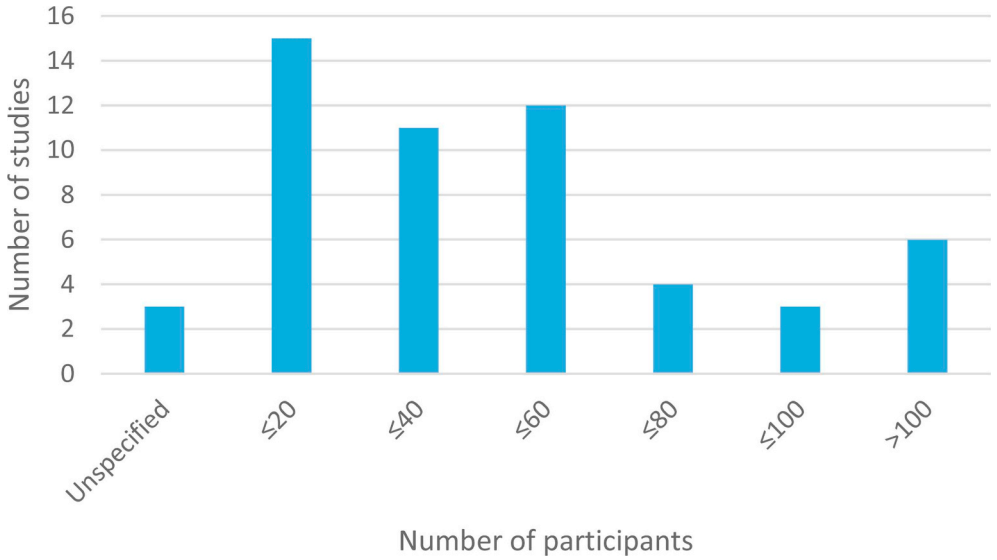

Fig. 5. Sample size of studies included.

Among the concepts, loops (iterations), conditionals and variables are considered to be core programming concepts (Funke & Geldreich, 2017; Meerbaum-Salant, Armoni & Ben-Ari, 2013) (Fig. 11). Sequences, loops and conditionals, for example, characterise most solutions expressed in any imperative language (Grover, Pea, & Cooper, 2015). What is more, these three concepts are common, regardless of the type of programming language. They are the basic structures and components in any programming language (Lahtinen, Ala-Mutka & Järvinen, 2005). Besides, in order to solve a problem, it is necessary to repeat certain statements or choose between possible scenarios and change the value of variables in the iterations or conditionals. As Table 3 demonstrated, these three concepts are commonly assessed in the same studies. One of the reasons for this is that the articles analysing the artefacts through Dr. Scratch, treated sequence and loop as one concept "flow control" and, similarly, conditionals and Boolean was treated as "logical thinking".

4.2.1.1. Computational concepts

4.2.1.1.1. Loops. The studies included examined different kinds of loops: simple loops, conditional loops and nested loops. Students showed a varied comprehension of these loops. In Baytak and Land's (2011) experiment, students' use of loops was limited to "forever" command blocks in programming games. Franklin et al. (2013) noticed that only three pairs out of ten successfully implemented complex animation with loop control in their culminating project. Similar results were also observed in Grover and Basu's (2017) and Mladenovic, Boljat & Zanko's (2017) work. They reported that most students struggled with nested loops. They got confused by loops with one or more variables. Vaníček (2015) pointed out that the use of endless loops often resulted in mistakes in

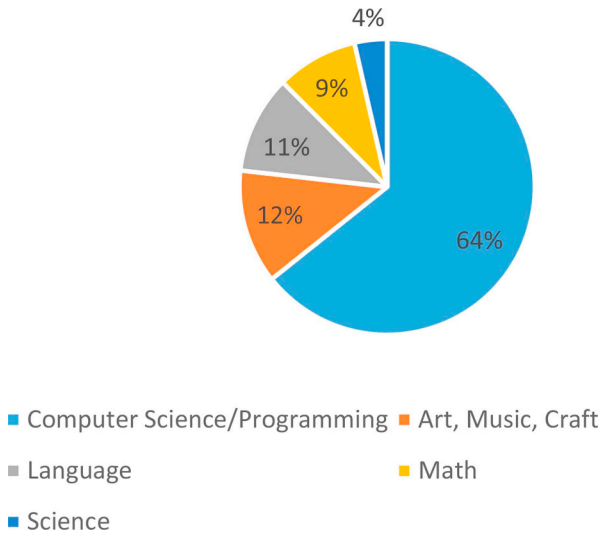

Fig. 6. Subjects implementing Scratch.

Fig. 7. Categories of Scratch artefacts.

the Scratch environment. However, there is also evidence of positive results. Giordano and Maiorana (2014, p. 560) found that despite "augmented cognitive load to master the use of a variable", the inner cycle of a nested loop was identified by more students after training in Scratch. Loops in programming allowed proper multimedia products, according to Sáez-López et al. (2016). Moreover, Meerbaum-Salant et al.'s (2013) article indicated that students grasped the concept of loops, as 57.5% of the students correctly defined bounded loops and 75% correctly defined conditional loops in the post-test. In general, although difficulties with complicated loops were encountered, students could at least internalise the basic concept of loops, which was commonly verified during the practice of creating in Scratch.

4.2.1.1.2. Sequences/algorithms. Several studies assessed sequences (algorithms) in a qualitative fashion. Wohl, Porter and Clinch (2015) interviewed students in a small group, asking them to explain the concept of algorithms. The pupils understood this concept easily as it could be related to their everyday lives. Sáez-López and Sevillano-García (2017) tested sequences in the process of creating music. The results showed statistically significant improvements in understanding algorithms after the programming session. Lee, Kim and Lee (2017, p. 201) evaluated students' handwritten algorithms on a three tier scale (low, medium and high) based on the following categories: "Real-life problems can be solved by algorithm design and programming; Real-life problems can be identified, and solutions can be expressed with the algorithm; Real-life problems can be identified, and solutions can be expressed in a natural language". Students achieved 3.2 points out of 6 when it came to implementing an algorithm. Sequences were also examined in Giordano and Maiorana's (2014) study. Students were required to read the specification of a problem, analyse it, design an algorithm, implement it and test the solution.The results were encouraging, showing a decreasing number of unanswered questions on the exam. Two assessment kits were developed and used by Strawhacker, Lee and Bers (2017): "Match the program" and "Reverse engineering". The first task required children to observe a project on a screen without seeing the code and then select from four options of code to best match the program. The second asked students to observe a video of a project. Then they were given all possible blocks in Scratch Jr. and had to recreate all of the sequences used to create the project. The experiment concluded that learners grasped sequences. Burke and Kafai (2012) suggested that students' storytelling used linear coding sequences and believed that storytelling offered a good entryway into learning Scratch's coding sequences. Generally speaking, learners comprehended sequences relatively easily since they could use analogies in daily life to conceptualise the concept as a list or series of instructions.

4.2.1.1.3. Conditionals. Conditionals were assessed through pre/post-test and Scratch artefacts. Tsukamoto, Oomori, Nagumo, Takemura, Monden and Matsumoto (2017) reported that it was difficult for the children to understand the meaning of the instructions for the problems, especially concerning the problem of conditional operations, in the pre-test. In contrast, Giordano and Maiorana's (2014) experiment led to an increased number of correct answers to conditional questions (from 5 to 11) in an examination after training in Scratch. A similar result was seen in Begosso and da Silva's (2013) assessment as the majority (63.5% of students) successfully answered the question regarding conditional and loop structures. The artefacts demonstrated different levels of

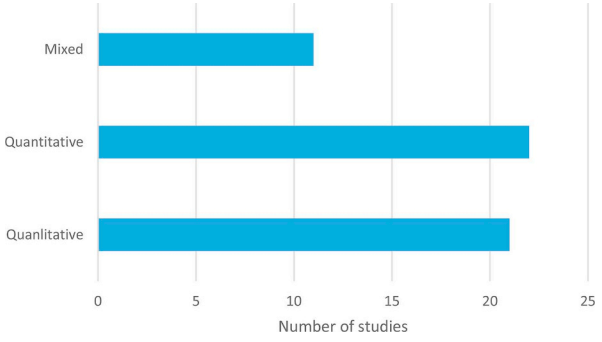

Fig. 8. Type of research methodology.

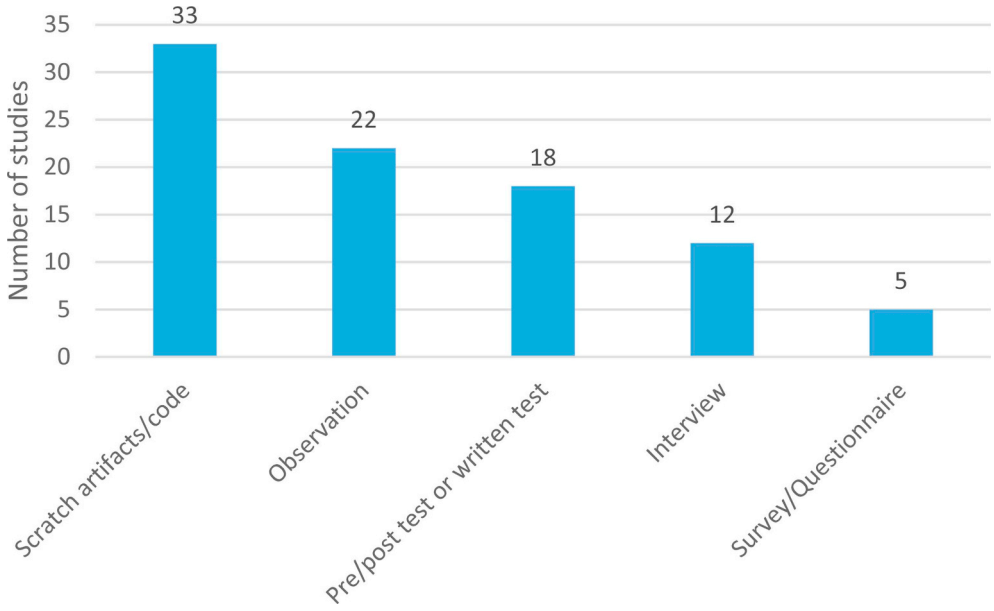

Fig. 9. Instruments for CT skills assessment.

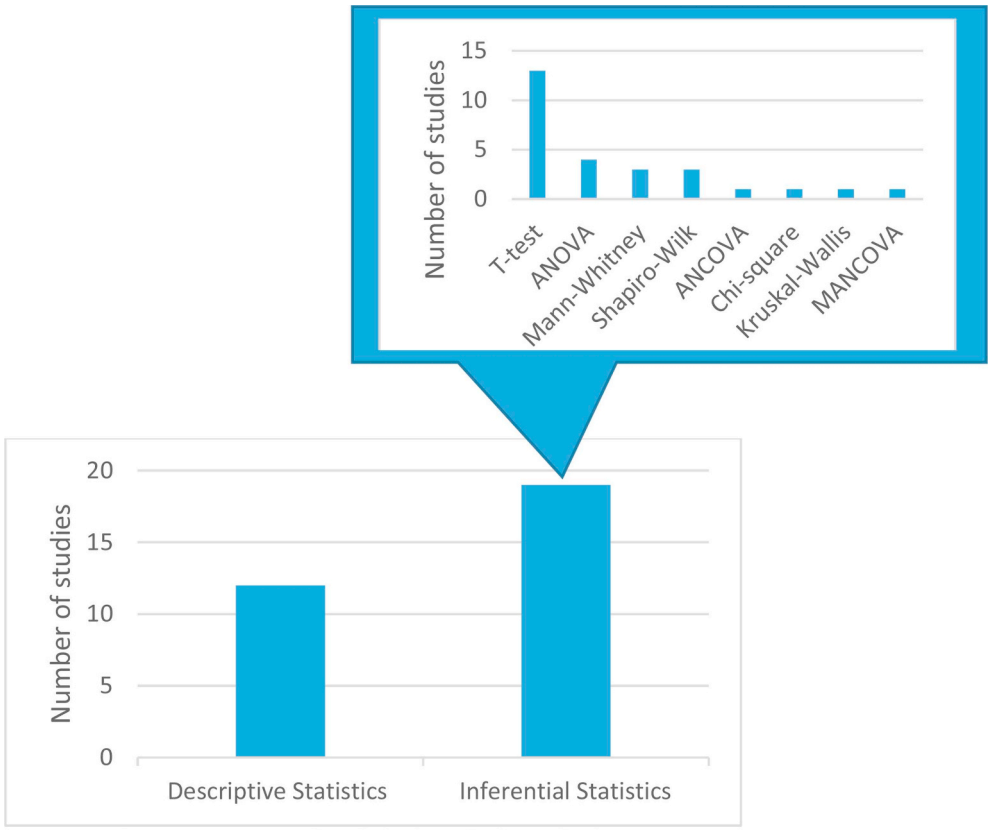

Fig. 10. Types of statistical analysis methods.

use of conditionals by students. For example, many games (more than 68%) used basic commands such as conditional logic mostly to manipulate actors' appearance, backgrounds and game scores (von Wangenheim, Alves, Rodrigues & Hauck, 2017). Kafai and Vasudevan (2015a) also mentioned how conditionals are used to increment and maintain the health of a character, for example, through a game. However, Baytak and Land (2011) noted that "if" command blocks were mostly employed in their study, while "if/

Table 5
CT skills identified.

CT Skills	Frequency	CT Skills	Frequency	CT Skills	Frequency	CT Skills	Frequency
Loop	28	Reading, interpreting and communicating code	15	Input/output	8	Remixing/reusing	4
Sequence/algorithms	26	Boolean logic/operators	12	Testing/debugging	8	Predicative thinking	3
Conditionals	24	Concurrency/parallelism	12	User interaction	7	Connecting	2
Variables/initialization	20	Events	12	Multimodal design	6	Being incremental and iterative	2
Coordination/synchronisation	16	Abstraction modularisation/problem decomposition	9	Expressing	5	Questioning	2

Note: See Appendix B for article details.

Table 6
CT skills in Brennan and Resnick's (2012) framework.

Category	Description (number of articles)
Computational concepts: the concepts designers engage with as they program	Sequences (26) Loops (28) Events (12) Parallelism (12) Conditionals (24) Operators (12) Data (20)
Computational practices: the practices designers develop as they engage with the concepts	Abstracting and modularising (9) Being incremental and iterative (2) Testing and debugging (8) Reusing and Remixing (4)
Computational perspective: the perspectives designers form about the world around them and about themselves	Expressing (5) Questioning (2) Connecting (3)

else" was rarely used in programming games. To sum up, the improved post-test results and the use of conditionals for flow control in programming games imply that the affordances of Scratch can facilitate the learning of conditionals.

4.2.1.1.4. Variables. Three kinds of variables were assessed: local variables, sprite-level variables and global variables (Fatourou, Zygouris, Loukopoulos, & Stamoulis, 2017. In the artefacts, variables were used to control the game score (Vaníček, 2015), adjust the position, distance and direction of sprites (Falloon, 2016; Meerbaum-Salant et al., 2013), draw geometric figures (Plaza, Sancristobal, Carro, Castro, Blázquez & Munoz et al., 2017) or control using messages and synchronising sprites (Fatourou et al., 2017). Although few studies observed positive learning results, such as Fields, Vasudevan, and Kafai (2015, p. 626), who noted "students consistently initialized all of their variables in every relevant aspect including location, layering, and appearance", students generally experienced difficulties in utilising variables. Variables were "under-represented" (Hermans & Aivaloglou, 2017, p. 55) or "not widely used" (Burke & Kafai, 2012, p. 437). In Burke and Kafai's (2012) experiment, the concept of variable attained a mean of merely 0.3 for "frequency per project", appearing in only 10% of the projects. The average use of variables in programs was also reported as low in Baytak and Land's (2011) study. Grover and Basu (2017) examined the ability to create variables, assign values and update variables, the ability to describe how a variable changes values in a loop and the ability to determine what variables are required in a program to achieve the goals of the computational solution. They found that "Students do not have a deep understanding of how loops work, what variables are, and what they do in a programming context. "Meaningful use of variables appears to be rare in middle school students' programs and curricula" (Fatourou et al., 2017, p. 271). Pre/post-tests also confirmed the challenge of comprehending variables. For example, most students did not define the concept of a variable but a few answers reflected its mathematical meaning: "a parameter", "a variable is x and x can be anything" (Meerbaum-Salant et al., 2013, p. 73). Regarding variables, the evidence in the literature points towards students' poor understanding of the concept and reveals the low ability to integrate variables in artefacts.

4.2.1.1.5. Boolean logic. The assessment of Boolean logic (operators) appeared 11 times in the literature. Davis, Kafai, Vasudevan, and Lee (2013) found that 6 out of 7 of the projects used operators. Giordano and Maiorana (2014) assessed composite Boolean logic and their study showed an encouraging rate of improvement. Grover and Basu (2017) examined students' understanding of Boolean logic and required the prediction of outputs of expressions with Boolean operators. Only about half of the students answered the questions about the OR operator, as students mixed how OR and "exclusive OR" operators work. Burke and Kafai (2012) stated that

Fig. 11. Word cloud of CT skills in Brennan and Resnick's (2012) framework as assessed in empirical studies.

20% of projects involved operators, although there was low use of operators per project. Students could not set Boolean expressions without the teacher's help (Baytak & Land, 2011). The projects in Moreno-León et al.'s (2015) study did not employ any operators, while one out of five projects used Boolean logic in Hoover, Barnes, Fatehi, Moreno-León, Puttick & Tucker-Raymond et al.'s (2016) study. It is worth noting that Dr. Scratch may have a drawback when it comes to analysing Boolean logic as including a single logic operation in a game would result in a score of 3 out of 3. The online version of Dr. Scratch was found to have problems recognising lists of logic operations (Raymond et al., 2016). Students should be encouraged to utilise Boolean logic as it is of merit in enhancing the complexity of conditionals and learning of loops.

4.2.1.1.6. Parallelism. Different types of parallelism (concurrency) were tested. According to Meerbaum-Salant et al. (2013, p. 74), "Type I concurrency occurs when several sprites are executing scripts simultaneously, such as sprites representing two dancers. Type II concurrency occurs when a single sprite executes more than one script simultaneously, for example, a Pac-Man sprite moving through a maze while opening and closing its mouth". The article (Meerbaum-Salant et al., 2013, p. 73) also mentioned that most students (92.5%) could not give a definition for concurrency but used unrelated mathematical concepts, such as "a rectangle with four parallel edges" and "two parallel lines that can never meet each other", when they were asked to define parallelism. In Lee et al.'s (2017) post-test, students received acceptable results (10 out of 24 points on average). An improvement was also seen in Sáez-López and Sevillano-García's (2017) statistical analysis, with 1.77 out of 5 points (poor) in the control group compared to 3.79 (good) in the experimental group. Furthermore, parallel execution of two independent scripts was supported by 16% of all projects and Funke and Geldreich's (2017) study even found 63% supporting the parallel launching of more than two scripts. Burke and Kafai (2012) noted that 100% of the projects used parallelism, although with low frequency per project. Some studies employed Dr. Scratch to assess parallelism on a 0–3 scale. Participants in Hoover et al.'s (2016) experiments were awarded scores of either 2 or 3 in Dr. Scratch. However, the projects in von Wangenheim et al.'s (2017) study ended up with scores of less than 2 and projects in Weng and Wong's (2017) even less, with 1. Weng & Wong (2017) also found that parallelism was used by only a few students. By and large, the experiments suggested that the learners performed better on Type I concurrency as it is more intuitive in using a sequential structure. The challenge lies in Type II concurrency, for example, requiring the execution of two or more scripts when a message is received or a backdrop changes.

4.2.1.1.7. Events. The learning of simple events (triggers) appeared positive. Funke and Geldreich's (2017) data showed that 93% of the projects used at least one event, while 80% included more than two events. Funke and Geldreich (2017) established that "when green flag clicked" and "when key pressed" were some of the most popular blocks. Kafai and Vasudevan (2015a; 2015b) and von Wangenheim et al. (2017) pointed out that in Scratch, designers most often start a program by clicking the green flag and that event commands were widely used. In Sáez-López and Sevillano-García (2017), a satisfying result was obtained by the experimental group in terms of their understanding of events, i.e. 3.83 out of 5 points. However, Burke and Kafai (2012) observed that although 100% of the projects in their study used events, there was low frequency per project. Moreover, 70% of the projects used no (or almost no) event blocks in Baytak and Land's (2011) research. Interestingly, most of the studies only explicitly mentioned "flag clicked" and "key pressed" blocks. This might explain the positive learning results, as these are such fundamental triggers in the event category that many projects simply would not be able to begin without them. However, the real challenge is to master more complicated events, such as "when backdrop switches to" and "when I receive a message", as students did not demonstrate sufficient use in the experiments.

4.2.1.2. Computational practices

4.2.1.2.1. Abstracting and modularising. The studies that were included point to positive results, in terms of mastering abstraction and modularisation (problem decomposition), which is the second most frequently assessed practice. Studies involving Dr. Scratch evaluated abstraction and modularisation together. In Moreno-León et al.'s (2015) work, students achieved 2 on a scale of 0–3, but the scores were relatively lower in Hoover et al.'s (2016) study, reaching 1 and 2. However, modularisation (problem decomposition) is promoted as an independent CT skill in both theoretical and empirical studies. First of all, with the exception of Brennan and Resnick (2012), all the frameworks separate problem decomposition from abstraction, as demonstrated in Table 1 in Section 2. Other researchers, such as Bers (2017, p. 72), define "modularising as the breaking down of tasks and procedures into simpler, manageable units that can be combined to create a more complex process". Several studies evaluated them separately. Statter and Armoni (2017, p. 10) analysed abstraction on its own through interviews in order to examine "the ability to choose the correct levels of abstraction at which to work, during the problem-solving process and move freely between levels of abstraction". In their study, a post-test was also conducted, which showed good results for abstraction ability. Statter and Armoni (2017) concluded that better abstraction skills induce better overall CT skills. Teachers have responded that learning how to program teaches students abstraction (von Wangenheim et al., 2017). Webb and Rosson's (2013) study, on the other hand, witnessed difficulties in grasping abstraction. Their study found that lists, an abstract "data structure", resulted in mechanical procedures among students. Their responses seem to tie in with a concrete understanding of the physical procedures for manipulating lists rather than an abstract concept for data storage. On the other hand, modularisation was not discussed in terms of patterns and consequences of decomposition in Scratch projects. Hence, this paper proposes to separate "abstracting and modularising" into two independent practices, because modularising (which involves subdividing jobs and engaging in decomposition), is a powerful idea on its own (Bers, 2017).

4.2.1.2.2. Debugging and testing. Debugging and testing is the most commonly observed computational practice. Debugging can take place both before and after testing (Falloon, 2016). Falloon noted that students assessed the effect of different code combinations before testing and also debugged code after testing. In this study, it was interesting that the students spent considerably more time analysing code before testing (11%) than after (6%). Often students began debugging systematically (line-by-line), which is more likely to lead to a successful result. In game design, play testing has been found to be essential for improving quality, as the making of

the game is closely intertwined with the playing of the game in the process (Kafai & Vasudevan, 2015b). Kafai and Vasudevan (2015b) found that students practised testing and debugging in a recursive fashion. As their projects evolved in terms of complexity, students revisited the previously coded part. Students also demonstrated their understanding of the debugging process in Strawhacker et al.'s (2017) and Burke and Kafai's (2012) experiments. Positive results were also confirmed by Webb and Rosson (2013, p. 100) as they observed "considerable success in the use of testing and debugging for learning how a program works, learn by testing and debugging example applications that could run but had logical or plot-related errors". Grover et al. (2015) used a post-test with multiple-choice and open-ended questions to assess learners' CT ability through questions that required code tracing and/or debugging. As a matter of note, the selected studies that engaged robotics or hardware with Scratch suggested that debugging and testing is not confined in the Scratch environment but can happen in the hardware or the connection between software and hardware. Generally speaking, the acquisition of testing and debugging ability has not proven to be successful within a relatively short time. Understanding of debugging is developed at a slow but steadily increasing pace (Wohl et al., 2015). Debugging code is a complex process requiring perseverance and a systematic approach (Falloon, 2016).

4.2.1.2.3. Remixing and reusing. Two types of remixing and reusing were found in the studies. Vasudevan, Kafai and Yang (2015) defined simple remixes as copies with surface modifications, whereas creative remixes depart in theme and structure from the original. This is consistent with Kafai and Vasudevan's (2015b) findings, as they categorise this practice as remixing aesthetics (e.g. using background, sprites, images, sounds) or remixing programs' functionality (e.g. making the game level more difficult). Both studies observed that participants took the original code and remixed it by changing the aesthetics and updating the text and functionality. According to these authors, the large majority of students went beyond surface changes in remixing code and designs. Davis et al. (2013) and Seiter (2015) also noted that some students either reused and modified code that they had written themselves previously or finished their final projects by customising code given by teachers. Reusing and remixing were popular practices and were considered a valid approach for supporting beginner programmers (Kafai & Vasudevan, 2015b). However, as useful as they are, only four studies evaluated these abilities. Hence, we recommend looking into this skill in greater depth in future research.

4.2.1.2.4. Being incremental and iterative. Being incremental and iterative was the least-often assessed practice. In Seiter's (2015) assessment, students were required to code in Scratch to make a given animation. Seiter (2015, p. 544) wrote that "they tended to write and test their code incrementally, i.e. one scene of the animation at a time. This allowed them to encounter errors while their scripts were smaller and easier to debug". Kafai and Vasudevan (2015a) observed in their craft experiment that all student teams participated in an iterative design process, moving from play testing to incorporating peer feedback. Then they updated their code or made improvements as they bug tested and played. This practice is often intertwined with testing and debugging. Altogether, it is argued that learners can be instilled with the perseverance to create iteratively and refine their work continuously (Bers, 2017, p. 76).

4.2.1.3. Computational perspectives. Six studies investigated expressing perspectives, while three did so for connecting and two for questioning. One of the studies used the three perspectives as their research question. Jun, Han, and Kim (2017, p. 47) asked three questions: "Could students create a new game using Scratch?" for the perspective of expressing; "Did students collaborate and interact when they made projects using Scratch?" for the perspective of connecting; "Did students have various questions and thoughts to solve real-world problems using Scratch?" for the perspective of questioning. The comparison between pre and post-tests showed that, through design-based learning, the experimental group made a greater number of significant positive changes in perspectives than the control group. Sáez-López et al.'s (2016) report showed connecting ability was enhanced to a greater extent than expressing. Moreover, Falloon (2016) also observed that questioning was naturally integrated into the collaborative approach when students developed solutions. When they analysed a problem, they frequently gave advice to and received from others to solve problems. He concluded that connecting was a common strategy employed at all stages of these students' work.

In summary, through the contrast between experimental and control groups in various experiments and the differences before and after using Scratch, one can see that Scratch helps students explore and master different levels of understanding of the various CT skills, depending on their cognitive difficulties.

4.2.2. Other CT skills identified in the selected studies

Besides the CT skills described above, that correspond to the Brennan and Resnick (2012) framework, several other CT skills were identified in the empirical studies (Table 7). This study borrows the Brennan and Resnick terms "concept, practice and perspective". This review identified an ability to work with the concept of input/output. Furthermore, three new computational practices were found in the literature, namely: reading, interpreting and communicating code (using multimodal media and predictive thinking). We also argue that human–computer interaction should be considered as being a computational perspective.

4.2.2.1. Input and output

"Coding is an attempt to articulate a precise input to facilitate a particular output" (Burke & Kafai, 2012, p. 438). Two studies explicitly examined the concept of input/output. Statter and Armoni (2017) used a "black box" question with the input-output mechanism to measure students' abstraction ability. As pointed out by Statter and Armoni (Statter and Armoni, 2017, p. 8):

In CS, a black-box argument means taking an algorithmic solution for a certain problem as an input-output mechanism (without any knowledge of its internal details) and using it to solve another problem. For example, as part of designing a solution to the searching problem, one can use a solution to the sorting problem without actually knowing how it was implemented and which algorithm is being used. To use a black-box argument, one must be able to think about what that part of the algorithm does, rather than how it does it.

Webb and Rosson's (2013) experiment involved hardware that asked students to integrate input from a sensor with corresponding output in the form of a motor movement. The students' responses to this confirmed that most of them could distinguish between input and output. Students also noticed the consequence of not handling input correctly in their programming. When user input was not tested appropriately, it led to nonsense responses on the screen. Another study also examined input/output among several other CT skills in a test of 10 true-or-false test items, 15 multiple-choice test items and 5 fill-in-the-blank test items (Wang, Hwang, Liang & Wang, 2017). Studies using Dr. Scratch categorise input under “user interactivity”. One point is awarded for “green flag clicked”, two points for “key pressed, sprite clicked, ask and wait, mouse blocks” and three points for “video/audio input".

We suggest that a distinction should be made between events and input/output. Events focus on causality: one thing triggers another to happen. Input and output exist independently as indispensable elements of a computational process. Input and output are associated with events. Inputs, such as clicking, moving the mouse or pressing buttons on the keyboard, are the trigger of an event. Without input, the event is not really possible. Hence, it is safe to say that input is the prerequisite of events. Therefore, learners should be acquainted with the concept of input/output. As Brennan and Resnick's (2012) text was based on Scratch, input was narrowly described as “click and press”. Besides, they may not have expressly mentioned output. Output can easily be seen as a part of expressing practice. The output of a Scratch program can be of functional or aesthetic value for the learner, enabling the learner to express his or her idea though programming. The variation in input/output goes beyond a mouse and a keyboard. When Scratch works closely with hardware such as Bluebot, Micro:bit and Arduino, input/output range from sound and light to textual or visual aspects. Being able to choose a suitable form of “output” allows the student to observe the program flow easily (Ruf, Mühling & Hubwieser, 2014), which can facilitate testing and debugging. Input and output, therefore, are useful for testing and debugging, as well as predicting, which is discussed later.

The concept of input/output should also be construed from the user-centred perspective. Funke and Geldreich's (2017, p. 59) assessment framework stated “The usage of mouse or loudness is used for dynamic interaction”. The input/output concept can be applied to increase learners' awareness of the possibilities, limitations and complexity of computational systems. “Producers are, in many cases, unable to identify user problems, even when the information is available on their desks … An important question for design is, therefore, how to anticipate the user's needs” (Hasu & Engeström, 2000, p. 62). This might be even more the case for beginners because they have so little understanding of computing that they may have unrealistic expectations about technologies and their complexity. When they test, debug or evaluate their programs, they should alternate their role between producer and user, while using input/output as a control mechanism to examine if their programs can withstand realistic demands. For instance, from a new user's perspective, this includes whether the text in a program can be understood easily and clearly, whether peers with disabilities can play the game and what the output should be when the wrong mouse button is clicked, a wrong key entered or number formats cannot be processed in a game or interactive poster (as well as how the outcome should be interpreted and how the output might guide the adjustment). This means that input/output from the user-centred perspective focuses on understanding the needs of the user as a way of informing design. A producer should give a considerable amount of thought to the users' perspective in programming artefacts.

4.2.2.2. Multimodal design. In total, eight articles examined coordination and synchronisation. Coordinating and synchronising require the synthesis of sound, images, actions and sprites; they entail a multimodal media design process. One strength of Scratch is that it has been designed to develop multimedia products, such as animation, games, and interactive arts (Lee, 2011). In Scratch, before the use of multimodal media takes place, the learners are supposed to be able to employ different types of media separately, for example being able to edit or draw new characters and settings and use simple sound or motion to change a sprite's costume in a script (Flannery et al., 2013). Then they can advance to the next level, synchronising complex multimodal media, such as the customisation of sprites and the background or including singing and dancing while costumes change their appearance (Franklin et al., 2016; Funke & Geldreich, 2017). This enables the learners to communicate their ideas through orchestrating different types of media, which is closely associated with expressing. Dr. Scratch offers a scheme to assess this practice. Hoover et al. (2016) reported positive results, namely that 4 out of 5 of the participants achieved the highest level of synchronising as they could manipulate “wait until”, “when backdrop changes to”, “broadcast” and “wait” blocks, while the rest had command of “broadcast”, “when I receive message”, “stop all”, “stop program”, “stop programs sprite”. Moreno-León et al. (2015) found that students presented less impressive performance as no work reached the highest level but they achieved medium and low levels. Moreover, Weng and Wong's (2017) study even showed that none of their seven learners demonstrated synchronisation ability. Studies not employing Dr. Scratch also noted different levels of the use of synchronising multimodal media. Fatourou et al. (2017) concluded that novice programmers can synchronise sprites using time and messages, whereas experienced programmers can even synchronise using conditional variables. All projects in Burke and Kafai's (2012) study used coordination and synchronisation with the highest frequency per project,

Table 7
Other CT skills not explicitly named within Brennan and Resnick's (2012) framework.

Category	Description (number of articles)
Computational concept	Input and output (8)
Computational practice	Reading, interpreting and communicating code (15)
	Multimodal design (6)
	Predictive thinking (3)
Computational perspective	User interaction (7)

compared to others assessing CT skills, such as loops, conditionals, etc. Fields et al. (2015) pointed out that, with respect to synchronisation, individual students applied a variety of strategies that ranged from the simple to the complex to produce cohesive music videos. Seiter (2015) noted that while students excelled at synchronising a simple conversation, they struggled to synchronise costume changes within a conversation. That is to say, synchronisation of multiple concerns across multiple scripts remained a challenge. As demonstrated above, one affordance of Scratch is that it provides the possibility to integrate multimodal media in programming. Learners ought to be able to solve problems that require the manipulation of multimodal media.

4.2.2.3. User interaction. Funke and Geldreich (2017) promoted human–computer interaction (HCI) in their assessment framework as "A highly interactive program provides the user with opportunities to interact with it … The project is intuitive if the user understands how the program runs with little or no information. Sometimes programs use unorthodox keys for controlling the program … This is only partly intuitive for users" (p. 1232). As both user-centred awareness and multimodal media production fall within the field of HCI, which can affect students' perceptions of computing (Yardi, Krolikowski, Marshall & Bruckman, 2008), we believe that HCI should be included in Brennan and Resnick's (2012) computational perspectives. CT suggests that one should approach problem solving like a computer scientist: "Computer scientists solve problems, but those solutions are ultimately for people, so they must be designed to work for people" (Curzon, 2013, p. 48). In other words, CT embraces how people interact with a computer. Curzon, McOwan, Plant, and Meagher (2014) went on to argue that the design and evaluation aspects of CT have to consider the HCI perspective. Calderon and Crick (2015, p. 2) also posited that "HCI can be embedded as a valuable part of developing computational thinking skills in young students, and that these will have a positive impact more broadly across their future learning, not just for computer science". Besides, in practice, when Scratch collaborates with robotics such as Bluebot or other hardware, such as Micro:bit and Arduino, the input/output can be from sound to movement, or light to textual and visual. They make computing more tangible and HCI plays an increasing role in tangible programming (Catlin & Woollard, 2014). It is against such a background that we propose the inclusion of HCI in computational perspectives.

4.2.2.4. Predictive thinking. It is necessary to take time to predict outcomes before building a program as this makes it possible to establish whether the original intuition was correct or whether this knowledge needs to be remodelled (Papert, 1980). As evidently presented in the data and aligned with debugging, Falloon (2016) identified the skill Papert described, which was termed predictive thinking: "Predictive thinking was analysis that occurred while students were building code, and can be described as thinking whereby they predicted or hypothesized a result from running code, before actually testing it" (p. 589). Benton, Hoyles, Kalas, and Noss (2017) observed that teachers use formative assessment for predictive thinking. One teacher in the experiment made changes to the sample scripts and asked pupils to predict the outcome and debug any errors. Another teacher often got pupils to predict the outcome of a script before she would run it. They argued that "it is important to have a goal in mind when building a computer program and to predict what the outcome might be before trying it out" (p.122). Flannery et al. (2013) argued that Scratch provides immediate feedback on the accuracy of students' predictions. As they program, they develop skills in estimating "How many?" or "How far?" This means they predict what will happen when they run each refined iteration of their program and think about whether the changes they have made will result in their intended output. Predictive thinking is an important skill. However, Falloon (2016) considers that predictive thinking needs to be balanced with encouragement to take risks. Concern arises due to the considerable amount of time spent on speculating possible issues, hence significantly slowing overall progress. Because making predictions in Scratch occurs as an implicit part of abstraction, it is without a doubt the hardest concept for young people to understand (Wohl et al., 2015). Therefore, we recommend that pupils learn to predict outcomes before program execution and compare these with the actual outcomes, then reflect on them. Predictive thinking should be listed as a practice and given appropriate attention and training to enhance overall problem-solving ability.

4.2.2.5. Reading, interpreting and communicating code. Reading, interpreting and communicating code are practices that require learners to identify, decode and interpret the code, as well as express ideas using computational vocabulary. Strawhacker et al. (2017) asked learners to identity the blocks in a task to test their ability to decode symbols. Those in kindergarten decoded symbols by recognising the actions they observed on the screen and matching them with specific programming icons on their answer sheet. In Bell's (2015) analysis, the researchers looked for evidence of a knowledge of vocabulary based on Scratch and systems thinking. They interviewed a student after a Scratch workshop and found that the student had learned to recognise the new CT vocabulary but the students' understanding was not sufficiently advanced to define the terms. Peel, Fulton and Pontelli (2015) also designed a test to check if students could recognise CT terms. The results indicated a significant improvement in understanding of CT vocabulary as students were attentive and responsive during the process of learning the terms. Grover et al. (2015) used pre/post-test measures to assess students' ability to read and decipher code or pseudo code and to explain computing terms using academic CT language. Another evaluation of CT terminology through interviews was carried out by Webb and Rosson (2013). Computing, like other disciplines, has its academic vocabulary. One purpose of using these terms is that ideas can be expressed and received in a manner that is as accurate and comprehensible as possible. During the process of programming, learners need to communicate with others, for example to exchange ideas, to debug, to connect and to question etc. Being able to communicate using CT terms can avoid ambiguity, hence improving efficiency. When trying to comprehend others' work, students need to be able to decode and interpret. As an algorithm is built on a synthesis of single blocks in Scratch and the ability to identify, decode and interpret code is intertwined with the algorithm. Therefore, we argue that this should be added as a computational practice in Brennan and Resnick's (2012) framework.

5. Discussion

The purpose of this review is to systematically examine the CT skills that can be obtained through Scratch in K-9 based on empirical evidence. The results demonstrate that all CT skills in Brennan and Resnick's (2012) framework can be delivered through the use of Scratch. Additional CT skills were found in the examined literature: input/output, reading, interpreting and communicating code, using multimodal media, predictive thinking and human–computer interaction. These additional CT skills are not currently presented in Brennan and Resnick's (2012) framework and can be considered as possible supplements to their framework.

However, different challenges are associated with the learning of these skills. In general, CT concepts are those most frequently assessed, while CT perspectives are those least assessed. It is likely that this is because the concepts grant easier access to assessment as they can be evaluated in different ways, such as written exams (pre/post-tests) and using (automated) artefacts analysis. In contrast, it is difficult to assess perspectives directly using any of the three approaches suggested by Brennan and Resnick (2012): project analysis, artefacts-based interviews, designing scenarios. This may place CT education in a dilemma. On the bright side, concepts may be well developed as teachers can take advantage of the convenience of assessment. However, there may be opting out of some CT practices and perspectives during assessment and thus these may present less development due to progress not being measured. Additionally, the results showed that different school subjects assessed different types of Scratch artefacts. Games, being the most popular artefact overall, are mainly associated with maths and computer science. Language subjects pay more attention to the development of plot and character than the efficiency of Scratch coding, hence prefer storytelling. Animation is art, music and crafts and mainly associated with the aesthetic design and quality. This may inspire potential investigations to see if certain Scratch artefacts give more opportunities for assessing certain CT dimensions. As games require correctness and clarity of coding, the assessment can primarily look into CT concepts and practices. On the other hand, storytelling and animation can highlight more on the CT perspectives. This difficulty of assessing CT perspectives, brought up by Lye and Koh as early as 2014, should be further addressed by the scholarly community. In view of the importance of the assessment of the learning process, researchers and teachers need to reflect on the methods of assessment so that all aspects of CT have an equal chance of being developed.

Now, if we zoom in on these CT skills, although Scratch helps students grasp the basics, many of them impose certain challenges for learning and teaching, such as nested loops, conditional loops, loops/conditionals with variables, loops/conditionals with Boolean logic, parallel multiple events, systematic testing and debugging etc. Certainly, the learners' age and their cognitive development are positively correlated with the level of understanding of CT skills (Lee et al., 2017; Strawhacker et al., 2017). This is reflected in Table 8 which shows the progression of identified CT skills based on the learners' age. The youngest group's (K-3rd grade) learning is limited to being able to read and construct simple sequences/algorithms regarding directions using Scratch Jr. The debugging practice concerns the finding of the wrong directional block and replacing them in the right sequence. What is worth noting here is that there is a more rounded assessment of CT perspectives than in the older age group. This might be caused by the needs of the early stage of oral and social skill development. So the children were encouraged to work in groups or pairs to complete a task. The teachers may value the “connecting, expressing and questioning” perspectives more than complicated algorithms. On the contrary, the oldest age group (7th to 9th grades) are more independent thinkers and they have the prerequisite for coding by themselves. The teachers also focus on the depth of the CT concepts. According to the table, all CT concepts and practices were introduced from the 4th grade, for instance, abstracting through the use of maths functions, data and variables on both local and global levels. Meanwhile, progression was observed. Loops were assessed already in 4th to 6th grades but not nested loops, which were mentioned by studies of 7th to 9th grades. Similarly, conditionals were taught to the 4th to 6th grade but not the combination of conditional and loop (conditional loop). There were also more articles assessing parallelism and user interaction for 7th to 9th grades than 4th to 6th.

5.1. Contributions and implications for research and practice

The contribution of this systematic review is threefold. First of all, the 55 included studies presented the CT skills that can be obtained by K-9 learners through Scratch. It answered the important questions “what to teach” (Grover & Pea, 2013) and what can be learned? As pointed out earlier (Table 1), there is no consensus of what CT constitutes, especially what can be learned in compulsory education. With the overview provided by this study, teachers and researchers can have a better understanding of the CT components that can be obtained through Scratch for K-9 learners, as well as the difficulties and challenges learners may encounter. Thus, the overview can guide teachers for lesson planning regarding the content of learning activities.

Furthermore, the progression of learning (Table 8) revealed in this systematic review can support curriculum planning and assessment of CT education on three levels, i.e., the individual teacher, school and regional/national curriculum. This finding can also be used to inform the CT skills that teachers on different educational level need to develop in order to be able to make the most out of visual programming languages such as Scratch. Regarding what can be learned at different educational levels and the challenges of learning CT skills, the findings of this study, in a sense, corroborates Ambrósio, Xavier, and Georges (2014) and Román-González et al. (2017) who demonstrated that development of computational thinking follows general cognitive development. Thus, when teachers and researchers ask themselves the questions of “what to teach” and “what can be learned” they need to, as concluded by Román-González et al. (2017), take into consideration the mental abilities developed by students at different levels. We believe that the progression table (Table 8) can support such considerations. However, we also believe that we need to start exploring if other programming languages and programming tools, such as text-based programming languages or robots, can better support the development of certain CT skills that are challenging for young learners to develop through Scratch. Furthermore, even though learning certain CT skills through Scratch might be difficult depending on the level of cognitive development, it has been shown that other more generic skills can be developed through Scratch, such as digital competency and 21st century skills (Nouri et al., 2019).

Table 8
Progression of CT skills based on learners' age.

Age of learner Grade	5–9 years old Kindergarten -3rd grade	9–12 years old 4th -6th grade	12–15 years old 7th -9th grade
CT concepts	Algorithm/Sequences Event Variable	Sequences Loops Events Parallelism Conditionals Operators Data Input/output	Sequences Loops Events Parallelism Conditionals Operators Data Input/output
CT practices	Being iterative and incremental Debugging Predictive thinking Reading, interpreting and communicating the code	Abstracting and modularising Being incremental and iterative Multimodal design Reusing and Remixing Reading, interpreting and communicating the code Testing and debugging	Abstracting and modularising Being incremental and iterative Multimodal design Predictive thinking Reusing and Remixing Reading, interpreting and communicating the code Testing and debugging
CT perspectives	Connecting Expressing Questioning	Connecting Expressing Questioning User interaction	Expressing User interaction

The third contribution of this study is related to the framework of Brennan and Resnick (2012). Firstly, this systematic review showed that the CT skills students obtain by using Scratch to high extent can be captured by their framework. Thus, the implication is that the framework can be used both by teachers when and researchers interested in the relation between Scratch and CT development when planning lessons or designing research studies. Secondly, based on the findings, we also argue that the framework is a suitable departure point for the construction of CT assessment instruments with a focus on Scratch, such as the validated CT test presented by Román-González et al. (2018). However, this systematic review also identified additional CT components that are not covered by Brennan and Resnick's CT framework, which we believe would make meaningful additions to the framework and assessment practices. At least, the identification of these additional CT skills encourages further investigation in both research and teaching practices.

5.2. Identified research gaps

The study selection process revealed that some studies lacked rigor in their research methodology. Firstly, some studies do not provide a complete background description of the study setup. Lack of information, such as the age and number of participants and length of the intervention, can all impede the replication of studies and comparison between different studies. An inability to replicate or reproduce results are major critiques of educational research (Makel & Plucker, 2014). The lack of information concerning the experimental settings simply will not help resolve this. The results of this study witnessed a rarity of replication of previous experimental results. To strengthen the validity of educational research in general, we recommend replicating and reproducing more studies with a rigorous experimental design. Moreover, teachers with limited experience and knowledge of CT might need detailed descriptions of the studies already conducted in order to guide their practice. Secondly, there are issues with the content of the assessments: some studies assessed several CT skills in one comprehensive evaluation to examine the overall performance of CT, such as a problem-solving test. Other studies, however, evaluated each CT skill with specific questions to see exactly how each skill developed and other studies provided no details of the assessments at all. Hence, it is difficult for other researchers to measure the suitability and reliability of these assessments. The questions that were asked and how they are presented can influence the student's results. Furthermore, the analysis of the assessment was often abstruse. Not least, statistical significance is a powerful tool for establishing an empirical position in education research (McLean & Ernest, 1998) and teachers at the K-9 level, especially those without extensive mathematical training, may experience difficulties in understanding the results of research without this. To facilitate teachers' understanding of CT education at the K-9 level and enable them to be inspired and guided by research, we recommend that some interpretation and explanation be provided in more everyday language. Moreover, many of the studies considered in this review did not include the ethical aspects of the interventions. These problems should be avoided in future studies.

5.3. Future studies

Lower cognitive ability can hinder CT learning. However, is this the only factor causing the challenges of learning the aforementioned CT skills? Future research could be dedicated to determining other factors that cause difficulties in understanding such learning. For example, are inappropriate didactic approaches accountable for encountered difficulties? Different learning strategies were engaged in these studies, for instance, collaborative learning. Are these learning strategies effective or unsuitable for delivering certain CT skills? Further efforts are needed to explore if the current didactic approaches are efficient in teaching CT skills. In the case where some of the didactic approaches are not working, we need to identify which and how they should be changed. Interventions

can be conducted to compare different ways of learning the same skill. Furthermore, we need research that explores if and how other programming languages and tools, such as text-based programming languages or robots, based on their particular affordances, better can support the development of certain CT skills that are difficult to develop trough Scratch. That is, we identify a need to compare how different programming languages and tools support the development of CT skills. Another gap relates to research in the application of CT in kindergarten. As shown in Appendix A, only four studies at the kindergarten level were included in this review.

Appendix

Appendix A. Selected studies

Database	Author(s)	Article title	Year of publication	Location of the empirical study	Age (years)/ grades of the participants
ACM (24)	Bell	Learning complex systems with story-building in scratch	2015	US	Middle school
	Burke and Kafai	The writers' workshop for youth programmers: Digital storytelling with Scratch in middle school classrooms	2012	US	12–14
	Davis, Kafai, Vasudevan, and Lee	The education arcade: Crafting, remixing, and playing with controllers for Scratch games	2013	US	11–12
	Flannery et al.	Designing ScratchJr: Support for early childhood learning through computer programming	2013	US	1st and 2nd grades
	Franklin et al.	Assessment of computer science learning in a scratch-based outreach program	2013	US	Middle school
	Franklin et al.	Initialization in scratch: Seeking knowledge transfer	2016	US	8–12
	Funke and Geldreich	Gender differences in Scratch programs of primary school children	2017	Germany	9–10
	Grover, Cooper, and Pea	Assessing computational learning in K-12	2014	US	7th and 8th grades
	Grover and Basu	Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic	2017	US	6th to 8th grade
	Hermans and Aivaloglou	To Scratch or not to Scratch? A controlled experiment comparing plugged first and unplugged first programming lessons	2017	Netherlands	8–12
	Hoover et al.	Assessing computational thinking in students' game designs	2016	US	Middle school
	Jun, Han, and Kim	Effect of design-based learning on improving computational thinking	2017	South Korea	10–12
	Kafai and Vasudevan	Hi-Lo tech games: Crafting, coding and collaboration of augmented board games by high school youth	2015a	US	13–15
	Kafai and Vasudevan	Constructionist gaming beyond the screen: Middle school students' crafting and computing of touchpads, board games, and controllers	2015b	US	11–14
	Lewis	How programming environment shapes perception, learning and goals: LOGO vs. Scratch	2010	US	10–12
	Meerbaum-Salant, Armoni, and Ben-Ari	Habits of programming in Scratch	2011	Italy	14–15
	Meerbaum-Salant, Armoni, and Ben-Ari	Learning computer science concepts with Scratch	2013	Italy	14–15
	Papadakis, Kalogiannakis, and Zaranis	Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education	2016	Greece	K
	Ruf, Mühling, and Hubwieser	Scratch vs. Karel: Impact on learning outcomes and motivation	2014	Germany	7th grade
	Seiter	Using SOLO to classify the programming responses of primary grade students	2015	US	4th grade
	Statter and Armoni	Learning abstraction in computer science: A gender perspective	2017	Italy	13–15
	Vasudevan, Kafai, and Yang	Make, wear, play: Remix designs of wearable controllers for Scratch games by middle school youth	2015	US	11–13
	Webb and Rosson	Using scaffolded examples to teach computational thinking concepts	2013	US	10–13
	Wohl, Porter, and Clinch	Teaching computer science to 5–7 year-olds: An initial study with Scratch, Cubelets and unplugged computing	2015	UK	5–7
IEEE (7)	Begosso and da Silva	Teaching computer programming: A practical review	2013	Brazil	11–13
	Funke, Geldreich, and Hubwieser	Analysis of Scratch projects of an introductory programming course for primary school students	2017	Germany	9–10
	Giordano and Maiorana	Use of cutting edge educational tools for an initial programming course	2014	Italy	14–16
	Moreno-León and Robles	Computer programming as an educational tool in the English classroom: A preliminary study	2015	Spain	9–11
	Peel, Fulton, and Pontelli	DISSECT: An experiment in infusing computational thinking in a sixth grade classroom	2015	US	6th grade
	Tsukamoto et al.	Evaluating algorithmic thinking ability of primary schoolchildren who learn computer programming	2017	Japan	3rd, 4th and 6th grades
	Weng and Wong	Integrating computational thinking into English dialogue learning through graphical programming tool	2017	Hong Kong	10–11

ERIC (4)	Burke	The markings of a new pencil: Introducing programming-as-writing in the middle school classroom	2012	US	12–14
	Fields, Vasudevan, and Kafai	The programmers' collective: Fostering participatory culture by making music videos in a high school Scratch coding workshop	2015	US	14–15
	Rose, Habgood, and Jay	An exploration of the role of visual programming tools in the development of young children's computational thinking	2017	UK	6–7
	von Wangenheim, Alves, Rodrigues, and Hauck	Teaching computing in a multidisciplinary way in social studies classes in school –A case study	2017	Brazil	5th and 7th grades
ScienceDirect (2)	Benton, Saunders, Kalas, Hoyles, and Noss	Designing for learning mathematics through programming: A case study of pupils engaging with place value	2018	UK	5th and 6th grades
	Sáez-López, Román-González, and Vázquez-Cano	Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools	2016	Spain	5th and 6th grades
SpinrgerLink (-13)	Athanasiou, Topali, and Mikropoulos	The use of robotics in introductory programming for elementary students	2016	Greece	9–12
	Baytak and Land	An investigation of the artefacts and process of constructing computers games about environmental science in a fifth grade classroom	2011	US	5th grade
	Benton, Hoyles, Kalas, and Noss	Bridging primary programming and mathematics: Some findings of design research in England	2017	UK	9–11
	Calao, Moreno-León, Correa, and Robles	Developing mathematical thinking with Scratch.	2015	Spain	6th grade
	Fatourou, Zygouris, Loukopoulos, and Stamoulis	Evaluation of early introduction to concurrent computing concepts in primary school	2017	Greece	11–12
	Ke and Im	A case study on collective cognition and operation in team-based computer game design by middle-school children	2014	US	Middle grades
	Lee, Kim, and Lee	Analysis of factors affecting achievement in maker programming education in the age of wireless communication	2017	UK	4th and 5th grades
	Mladenović, Boljat, and Žanko	Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level	2017	Croatia	11–12
	Plaza et al.	Scratch as educational tool to introduce robotics	2017	Spain	7–17
	Portelance, Strawhacker, and Bers	Constructing the ScratchJr programming language in the early childhood classroom	2016	US	K to 2nd grade
	Segredo, Miranda, León, and Santos	Developing computational thinking abilities instead of digital literacy in primary and secondary school students	2016	Spain	11–13
	Strawhacker, Lee, and Bers	Teaching tools, teachers' rules: Exploring the impact of teaching styles on young children's programming knowledge in ScratchJr.	2017	US	K to 2nd grade
	Vaníček	Programming in Scratch using inquiry-based approach.	2015	Czech Republic	15
Web of Science (5)	Falloon	An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad	2016	New Zealand	5–6
	Grover, Pea, and Cooper	Designing for deeper learning in a blended computer science course for middle school students	2015	US	11–14
	Moreno-León, Robles and Román-González	Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking	2015	Spain	10–14
	Sáez-López and Sevillano-García	Sensors, programming and devices in art education sessions. One case in the context of primary education	2017	Spain	6th grade
	Wang, Hwang, Liang, and Wang	Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: An online peer-assessment attempt	2016	Taiwan	9th grade

Appendix B. CT skills identified

CT Skills	Frequency	Article
Loops	28	Baytak and Land (2011), Bell (2015), Burke (2012), Burke and Kafai (2012), Davis et al. (2013), Flannery et al. (2013), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Grover and Basu (2017), Hoover et al. (2016), Lewis (2010), Meerbaum and Armoni (2011), Meerbaum-Salant et al. (2013), Mladenovic et al. (2017), Moreno-León and Robles (2015), Moreno-León et al. (2015), Peel et al. (2015), Portelance et al. (2016), Ruf et al. (2014), Sáez-López and Sevillano-García (2017), Sáez-López et al. (2016), Tsukamoto et al. (2017), Vaníček (2015), von Wangenheim et al. (2017), Wang et al. (2017), Webb and Rosson (2013), Weng and Wong (2017)

Sequences/algorithms	26	Athanasiou et al. (2016), Baytak and Land (2011), Benton et al. (2018), Burke (2012), Calao et al. (2015), Davis et al. (2013), Falloon (2016), Funke and Geldreich (2017), Grover et al. (2014, 2015), Hoover et al. (2016), Kafai and Vasudevan (2015b), Lee et al. (2017), Mladenović et al. (2017), Moreno-León and Robles (2015), Moreno-León et al. (2015), Peel et al. (2015), Ruf et al. (2014), Sáez-López and Sevillano-García (2017), Sáez-López et al. (2016), Segredo et al. (2016), Strawhacker et al. (2017), Tsukamoto et al. (2017), Weng and Wong (2017), Wohl et al. (2015)
Conditionals	24	Baytak and Land (2011), Burke (2012), Burke and Kafai (2012), Davis et al. (2013), Giordano and Maiorana (2014), Grover et al. (2014, 2015), Hoover et al. (2016), Kafai and Vasudevan (2015a), Lewis (2010), Meerbaum and Armoni (2011), Meerbaum-Salant et al. (2013), Moreno-León and Robles (2015), Moreno-León et al. (2015), Peel et al. (2015), Portelance et al. (2016), Ruf et al. (2014), Sáez-López and Sevillano-García (2017), Strawhacker et al. (2017), Tsukamoto et al. (2017), Vaníček (2015), von Wangenheim et al. (2017), Wang et al. (2017), Weng and Wong (2017)
Variables/initialization	20	Baytak and Land (2011), Burke (2012), Burke and Kafai (2012), Falloon (2016), Fatourou et al. (2017), Flannery et al. (2013), Franklin et al. (2013, 2016), Giordano and Maiorana (2014), Grover and Basu (2017), Hoover et al. (2016), Meerbaum and Armoni (2011), Moreno-León and Robles (2015), Moreno-León et al. (2015), Plaza et al. (2018), Statter and Armoni (2017), Vaníček (2015), von Wangenheim et al. (2017), Wang et al. (2017), Weng and Wong (2017)
Coordination/synchronisation	16	Burke (2012), Burke and Kafai (2012), Fatourou et al. (2017), Fields et al. (2015), Flannery et al. (2013), Franklin et al. (2013), Funke and Geldreich (2017), Hoover et al. (2016), Meerbaum and Armoni (2010, 2011), Moreno-León and Robles (2015), Moreno-León et al. (2015), Seiter (2015), Vaníček (2015), Weng and Wong (2017)
Reading, interpreting and communicating code	15	Begosso and da Silva (2013), Bell (2015), Burke and Kafai (2012), Flannery et al. (2013), Grover et al. (2014, 2015), Hermans and Aivaloglou (2017), Papadakis et al. (2016), Peel et al. (2015), Portelance et al. (2016), Ruf et al. (2014), Sáez-López and Sevillano-García (2017), Statter and Armoni (2017), Strawhacker et al. (2017), Webb and Rosson (2013)
Boolean logic/operators	12	Baytak and Land (2011), Burke (2012), Burke and Kafai (2012), Davis et al. (2013), Giordano and Maiorana (2014), Grover and Basu (2017), Hoover et al. (2016), Moreno-León and Robles (2015), Moreno-León et al. (2015), Sáez-López and Sevillano-García (2017), von Wangenheim et al. (2017), Weng and Wong (2017)
Concurrency/parallelism	12	Baytak and Land (2011), Burke and Kafai (2012), Flannery et al. (2013), Funke and Geldreich (2017), Hoover et al. (2016), Kafai and Vasudevan (2015b), Lee et al. (2017), Moreno-León and Robles (2015), Moreno-León et al. (2015), von Wangenheim et al. (2017), Weng and Wong (2017)
Events	12	Baytak and Land (2011), Burke (2012), Burke and Kafai (2012), Davis et al. (2013), Falloon (2016), Fields et al. (2015), Funke and Geldreich (2017), Hoover et al. (2016), Kafai and Vasudevan (2015a,b), Portelance et al. (2016), von Wangenheim et al. (2017)
Abstraction modularisation/problem decomposition	9	Hoover et al. (2016), Lee et al. (2017), Moreno-León and Robles (2015), Moreno-León et al. (2015), Peel et al. (2015), Segredo et al. (2016), Statter and Armoni (2017), Webb and Rosson (2013), Weng and Wong (2017)
Input/output	8	Funke and Geldreich (2017), Hoover et al. (2016), Moreno-León and Robles (2015), Moreno-León et al. (2015), Wang et al. (2017), Webb and Rosson (2013), Weng and Wong (2017)
Testing/debugging	8	Baytak and Land (2011), Burke (2012), Falloon (2016), Kafai and Vasudevan (2015b), Strawhacker et al. (2017), Webb and Rosson (2013), Wohl et al. (2015)
User interaction	7	Funke and Geldreich (2017), Hoover et al. (2016), Moreno-León and Robles (2015), Moreno-León et al. (2015), Plaza et al. (2018), von Wangenheim et al. (2017)
Multimodal design	6	Flannery et al. (2013), Funke and Geldreich (2017), Portelance et al. (2016), Vaníček (2015), Wang et al. (2017)
Expressing	5	Falloon (2016), Jun et al. (2017), Kafai and Vasudevan (2015b), Ke and Im (2014), Sáez-López and Sevillano-García (2017)
Remix/reuse	4	Davis et al. (2013), Kafai and Vasudevan (2015b), Vasudevan et al. (2015)
Connecting	3	Falloon (2016), Jun et al. (2017), Sáez-López and Sevillano-García (2017)
Predicative thinking	3	Benton et al. (2018), Statter and Armoni (2017), Wohl et al. (2015)
Being incremental and iterative	2	Falloon (2016), Kafai and Vasudevan (2015a)
Questioning	2	Falloon (2016), Jun et al. (2017)

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Ambrósio, A. P., Xavier, C., & Georges, F. (2014). Digital ink for cognitive assessment of computational thinking. *2014 IEEE frontiers in education conference (FIE) proceedings* (pp. 1–7). IEEE.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., et al. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society, 19*(3), 47–57.

Athanasiou, L., Topali, P., & Mikropoulos, T. A. (2016, November). The use of robotics in introductory programming for elementary students. *In International Conference EduRobotics 2016* (pp. 183–192). Athens, Greece: Springer Cham.

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems, 75*, 661–670.

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.

Basawapatna, A. R., Repenning, A., Koh, K. H., & Nickerson, H. (2013, August). The zones of proximal flow: Guiding students through a space of computational thinking skills and challenges. *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 67–74). California US. ACM.

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research & Development, 59*(6), 765–782.

Begosso, L. C., & da Silva, P. R. (2013, October). Teaching computer programming: A practical review. *Frontiers in education conference, 2013* (pp. 508–510). Oklahoma, US: IEEE.
Bell, A. M. (2015, June). Learning complex systems with story-building in scratch. *Proceedings of the 14th international conference on interaction design and children* (pp. 307–310). Massachusetts US. ACM.
Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *New Zealand Journal of Applied Computing and Information Technology, 13*(1), 20–29.
Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education, 3*(2), 115–138.
Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International journal of child-computer interaction, 16*, 68–76.
Berland, M., & Wilensky, U. (2015). Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. *Journal of Science Education and Technology, 24*(5), 628–647.
Bers, M. U. (2017). *Coding as a playground: Programming and computational thinking in the early childhood classroom.* (Routledge).
Blackwell, A. (2002, June). What is programming. *14th workshop of the psychology of programming interest group (204–218)*. London: Brunel University.
Bower, M., Wood, L. N., Lai, J. W., Howe, C., Lister, R., Mason, R., et al. (2017). Improving the computational thinking pedagogical capabilities of school teachers. *Australian Journal of Teacher Education, 42*(3), 4.
Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American educational research association* (pp. 1–25). (Vancouver, Canada).
Brennan, K., & Resnick, M. (2013, March). Stories from the scratch community: Connecting with ideas, interests, and people. *Proceeding of the 44th ACM technical symposium on computer science education* (pp. 463–464). Colorado, US: ACM.
Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education, 4*(2), 121–135.
Burke, Q., & Kafai, Y. B. (2012, February). The writers' workshop for youth programmers: Digital storytelling with scratch in middle school classrooms. *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 433–438). North Carolina, US: ACM.
Calao, L. A., Moreno-León, J., Correa, H. E., & Robles, G. (2015). Developing mathematical thinking with scratch. *Design for teaching and learning in a networked world* (pp. 17–27). Toledo, Spain: Springer Cham.
Calderon, A. C., & Crick, T. (2015, November). Interface design and its role in computational thinking. *Proceedings of the 10th workshop in primary and secondary computing education* (pp. 127–129). London, UK: ACM.
Catlin, D., & Woollard, J. (2014, July). Educational robots and computational thinking. *Proceedings of 4th international workshop teaching robotics, teaching with robotics & 5th international conference robotics in education* (pp. 144–151). (Padova, Italy).
Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education, 109*, 162–175.
Computer Science Teachers Association (CSTA) Standards Task Force (2011). *CSTA K-12 computer science standards* (pp. 9). . Available from: http://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf.
Computing At School (n.d.) Computational thinking. Available from:http://community.computingatschool.org.uk/resources/252.
Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.
Curzon, P. (2013, November). cs4fn and computational thinking unplugged. *Proceedings of the 8th workshop in primary and secondary computing education* (pp. 47–50). Aarhus, Denmark: ACM.
Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014, November). Introducing teachers to computational thinking using unplugged storytelling. *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 89–92). Berlin, Germany: ACM.
Davis, R., Kafai, Y., Vasudevan, V., & Lee, E. (2013, June). The education arcade: Crafting, remixing, and playing with controllers for scratch games. *Proceedings of the 12th international conference on interaction design and children* (pp. 439–442). New York, US: ACM.
Department of Education UK (2015). *Statutory guidance: National curriculum in England: Computing programmes of study.* https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study.
Duncan, C., Bell, T., & Tanimoto, S. (2014, November). Should your 8-year-old learn to code? *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 60–69). Berlin, Germany: ACM.
Elo, S., & Kyngäs, H. (2008). The qualitative content analysis process. *Journal of Advanced Nursing, 62*(1), 107–115.
Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning, 32*(6), 576–593.
Fatourou, E., Zygouris, N. C., Loukopoulos, T., & Stamoulis, G. I. (2017, September). Evaluation of early introduction to concurrent computing concepts in primary school. *20th international conference on interactive collaborative learning* (pp. 543–552). Budapest, Hungary: Springer, Cham.
Fields, D., Vasudevan, V., & Kafai, Y. B. (2015). The programmers' collective: Fostering participatory culture by making music videos in a high school scratch coding workshop. *Interactive Learning Environments, 23*(5), 613–633.
Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013, June). Designing Scratch Jr: Support for early childhood learning through computer programming. *Proceedings of the 12th international conference on interaction design and children* (pp. 1–10). New York, US: ACM.
Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., et al. (2013, March). Assessment of computer science learning in a scratch-based outreach program. *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 371–376). Colorado, US: ACM.
Franklin, D., Hill, C., Dwyer, H. A., Hansen, A. K., Iveland, A., & Harlow, D. B. (2016, March). Initialization in scratch: Seeking knowledge transfer. *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 217–222). Tennessee, US: ACM.
Funke, A., & Geldreich, K. (2017, November). Gender differences in scratch programs of primary school children. *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 57–64). Nijmegen, Netherlands: ACM.
Giordano, D., & Maiorana, F. (2014, April). Use of cutting edge educational tools for an initial programming course. *Global engineering education conference (EDUCON), 2014 IEEE* (pp. 556–563). IEEE.
Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 267–272). Washington, US: ACM.
Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.
Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57–62). Uppsala, Sweden: ACM.
Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25*(2), 199–237.
Hasu, M., & Engeström, Y. (2000). Measurement in action. An activity-theoretical perspective on produce user interaction. *International Journal of Human-Computer Studies, 53*(1), 61–89.
Heintz, F., Mannila, L., & Färnqvist, T. (2016, October). A review of models for introducing computational thinking, computer science and computing in K-12 education. *Frontiers in education conference (FIE) 2016 (1–9)*. Pennsylvania, US: IEEE.
Hermans, F., & Aivaloglou, E. (2017, November). To scratch or not to scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons. *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 49–56). Nijmegen, The Netherlands: ACM.
Hoover, A. K., Barnes, J., Fatehi, B., Moreno-León, J., Puttick, G., Tucker-Raymond, E., et al. (2016, October). Assessing computational thinking in students' game designs. *Proceedings of the 2016 annual symposium on computer-human interaction in play companion extended abstracts* (pp. 173–179). Texas, US: ACM.

Jun, S., Han, S., & Kim, S. (2017). Effect of design-based learning on improving computational thinking. *Behaviour & Information Technology, 36*(1), 43–53.
Kafai, Y., & Vasudevan, V. (2015a). Hi-Lo tech games: Crafting, coding and collaboration of augmented board games by high school youth. *Proceedings of the 14th international conference on interaction design and children* (pp. 130–139). Massachusetts, US: ACM.
Kafai, Y., & Vasudevan, V. (2015b). Constructionist gaming beyond the screen: Middle school students' crafting and computing of touchpads, board games, and controllers. *Proceedings of the workshop in primary and secondary computing education* (pp. 49–54). London, UK: ACM.
Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing, 4*(3), 583–596.
Kalelioğlu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13*(1), 33–50.
Ke, F., & Im, T. (2014). A case study on collective cognition and operation in team-based computer game design by middle-school children. *International Journal of Technology and Design Education, 24*(2), 187–201.
Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*Technical Report EBSE 2007-001. Keele University and Durham University Joint Report.
Kong, S. C. (2016). A framework of curriculum design for computational thinking development in K-12 education. *Journal of Computers in Education, 3*(4), 377–394.
Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM Sigcse Bulletin, 37*(3), 14–18.
Lee, Y. J. (2011). Scratch: Multimedia programming environment for young gifted learners. *The Gifted Child Today, 34*(2), 26–31.
Lee, S., Kim, J., & Lee, W. (2017). Analysis of factors affecting achievement in maker programming education in the age of wireless communication. *Wireless Personal Communications, 93*(1), 187–209.
Lewis, C. M. (2010, March). How programming environment shapes perception, learning and goals: logo vs. scratch. *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 346–350). ACM.
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.
Makel, M. C., & Plucker, J. A. (2014). Facts are more important than novelty: Replication in the education sciences. *Educational Researcher, 43*(6), 304–316.
Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education, 10*(4), 16.
Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., et al. (2014, June). Computational thinking in K-9 education. *Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference* (pp. 1–29). Uppsala, Sweden: ACM.
McLean, J. E., & Ernest, J. M. (1998). The role of statistical significance testing in educational research. *Research in the Schools, 5*(2), 15–22.
Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011, June). Habits of programming in scratch. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168–172). Darmstadt, Germany: ACM.
Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239–264.
Mladenović, M., Boljat, I., & Žanko, Ž. (2017). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies, 23*(4), 1483–1500.
Moreno-León, J., & Robles, G. (2015, March). Computer programming as an educational tool in the English classroom a preliminary study. *2015 IEEE Global Engineering Education Conference (EDUCON)* (pp. 961–966). Tallin, Estonia: IEEE.
Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia, 46*, 1–23.
Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2019). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry, 10*(3).
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books, Inc.
Peel, A., Fulton, J., & Pontelli, E. (2015, October). DISSECT: An experiment in infusing computational thinking in a sixth grade classroom. *Frontiers in education conference (FIE), 2015 IEEE (1-8)*. Texas, US: IEEE.
Plaza, P., Sancristobal, E., Carro, G., Castro, M., Blázquez, M., Muñoz, J., et al. (2017, September). Scratch as educational tool to introduce robotics. *International conference on interactive collaborative learning* (pp. 3–14). Budapest, Hungary: Springer, Cham.
Portelance, D. J., Strawhacker, A. L., & Bers, M. U. (2016). Constructing the ScratchJr programming language in the early childhood classroom. *International Journal of Technology and Design Education, 26*(4), 489–504.
Price, T. W., & Barnes, T. (2015, July). Comparing textual and block interfaces in a novice programming environment. *Proceedings of the 11th annual international conference on international computing education research* (pp. 91–99). Neveda, US: ACM.
Repenning, A., Basawapatna, A., & Escherle, N. (2016, September). Computational thinking tools. *2016 IEEE symposium on visual languages and human-centric computing (VL/HCC)* (pp. 218–222). Cambridge, UK: IEEE.
Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior, 72*, 678–691.
Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the computational thinking test. *International Journal of Child-Computer Interaction, 18*, 47–58.
Rose, S., Habgood, J., & Jay, T. (2017). An exploration of the role of visual programming tools in the development of young children's computational thinking. *Electronic Journal of e-Learning, 15*(4), 297–309.
Ruf, A., Mühling, A., & Hubwieser, P. (2014, November). Scratch vs. Karel: Impact on learning outcomes and motivation. *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 50–59). Berlin, Germany: ACM.
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.
Sáez-López, J. M., & Sevillano-García, M. L. (2017). Sensors, programming and devices in Art Education sessions. One case in the context of primary education. *Cultura y Educación, 29*(2), 350–384.
Scratch statistics. Retrieved: https://scratch.mit.edu/statistics/.
Segredo, E., Miranda, G., León, C., & Santos, A. (2016). Developing computational thinking abilities instead of digital literacy in primary and secondary school students. *Smart Education and e-Learning 2016* (pp. 235–245). Springer.
Seiter, L. (2015, March). Using SOLO to classify the programming responses of primary grade students. *Proceedings of the 46th ACM technical symposium on computer science education* (pp. 540–545). Missouri, US: ACM.
Seiter, L., & Foreman, B. (2013, September). Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of the 9th annual international ACM conference on International computing education research* (pp. 59–66). Auckland, New Zeeland: ACM.
Selby, C. (2012, November). Promoting computational thinking with programming. *Proceedings of the 7th workshop in primary and secondary computing education* (pp. 74–77). Hamburg, Germany: ACM.
Selby, C., & Woollard, J. (2013). Computational thinking: The developing definition. Available: http://eprints.soton.ac.uk/356481, Accessed date: 23 March 2018.
Statter, D., & Armoni, M. (2017, November). Learning abstraction in computer science: A gender perspective. *Proceedings of the 12th workshop on primary and secondary computing education (5-14)*. Nijmegen, Netherlands: ACM.
Strawhacker, A., Lee, M., & Bers, M. U. (2017). Teaching tools, teachers' rules: Exploring the impact of teaching styles on young children's programming knowledge in scratch Jr. *International Journal of Technology and Design Education, 28*(2), 347–376.
The College Board. (2013). *AP computer science principles draft curriculum framework.* New York, NY: College Board. Retrieved from http://www.csprinciples.org/home/about-the-project/docs/csp-cf-2013. pdf?attredirects=0&d=1.
TIOBE index. (2018). Retrieved from: https://www.tiobe.com/tiobe-index/.

Tsukamoto, H., Oomori, Y., Nagumo, H., Takemura, Y., Monden, A., & Matsumoto, K. I. (2017, October). Evaluating algorithmic thinking ability of primary schoolchildren who learn computer programming. *Frontiers in education conference (FIE)* (pp. 1–8). Indiana, US: IEEE.

Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, greenfoot, and scratch-a discussion. *ACM Transactions on Computing Education (TOCE), 10*(4), 2–3 17.

Vaníček, J. (2015, September). Programming in Scratch using inquiry-based approach. *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 82–93). Ljubljana, Slovenia: Springer, Cham.

Vasudevan, V., Kafai, Y., & Yang, L. (2015, June). Make, wear, play: Remix designs of wearable controllers for scratch games by middle school youth. *Proceedings of the 14th international conference on interaction design and children* (pp. 339–342). Massachusetts, US: ACM.

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

von Wangenheim, C. G., Alves, N. C., Rodrigues, P. E., & Hauck, J. C. (2017). Teaching computing in a multidisciplinary way in social studies classes in school–A case study. *International Journal of Computer Science Education in Schools, 1*(2).

Wang, X. M., Hwang, G. J., Liang, Z. Y., & Wang, H. Y. (2017). Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: An online peer-assessment attempt. *Journal of Educational Technology & Society, 20*(4), 58–68.

Webb, H., & Rosson, M. B. (2013, March). Using scaffolded examples to teach computational thinking concepts. *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 95–100). Colarado, US: ACM.

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

Weng, X., & Wong, G. K. (2017, December). Integrating computational thinking into English dialogue learning through graphical programming tool. *6th international conference on teaching, assessment, and learning for engineering (TALE)* (pp. 320–325). Hong Kong, China: IEEE.

Wilson, A., & Moffat, D. C. (2010, September). Evaluating Scratch to introduce younger schoolchildren to programming. *Proceedings of the 22nd annual psychology of programming interest group. Madrid, Spain*.

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

Wing, J. (2011). *Research notebook: Computational thinking—What and why.* The Link Magazine20–23.

Wohl, B., Porter, B., & Clinch, S. (2015, November). Teaching computer science to 5-7 year-olds: An initial study with scratch, cubelets and unplugged computing. *Proceedings of the workshop in primary and secondary computing education* (pp. 55–60). London, UK: ACM.

Wong, G. K., Cheung, H. Y., Ching, E. C., & Huen, J. M. (2015, December). School perceptions of coding education in K-12: A large scale quantitative study to inform innovative practices. *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (pp. 5–10). Zhuhai, China: IEEE.

Yardi, S., Krolikowski, P., Marshall, T., & Bruckman, A. (2008). An HCI approach to computing in the real world. *Journal on Educational Resources in Computing (JERIC), 8*(3), 9.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research, 53*(4), 562–590.