Contents lists available at ScienceDirect

Educational Research Review

journal homepage: www.elsevier.com

Review

Demystifying computational thinking

Valerie J. Shute[?a?, ?*], Chen Sun[?a], Jodi Asbell-Clarke[?b]

[a] *Florida State University, USA*
[b] *TERC, USA*

ARTICLE INFO

Article history:
Received 4 March 2017
Received in revised form 17 July 2017
Accepted 14 September 2017
Available online xxx

Keywords:
Computational thinking
Computational literacy
Problem solving
Programming

ABSTRACT

This paper examines the growing field of computational thinking (CT) in education. A review of the relevant literature shows a diversity in definitions, interventions, assessments, and models. After synthesizing various approaches used to develop the construct in K-16 settings, we have created the following working definition of CT: The conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts. This definition highlights that CT is primarily a way of thinking and acting, which can be exhibited through the use particular skills, which then can become the basis for performance-based assessments of CT skills. Based on the literature, we categorized CT into six main facets: decomposition, abstraction, algorithm design, debugging, iteration, and generalization. This paper shows examples of CT definitions, interventions, assessments, and models across a variety of disciplines, with a call for more extensive research in this area.

1. Introduction

In the middle ages, only select groups of people (e.g., priests and scribes) could read and write. But as the world evolved, increasingly more people needed these skills. Today, the rapid onset of computers in the 20th century is forcing an analogous revolution where digital literacy is now an essential skill to succeed in our complex, digital 21st century world. And although we don't all need to become software engineers, the majority of us do use computers daily and need to understand how to communicate with them to most effectively harness their computing power. Successful communication along these lines is called computational thinking (CT).

Over the past decade, CT has become a very hot topic in educational research and practice. Thousands of entries appear on a general Google search regarding its definition, instructional interventions, and assessment. Many of these entries suggest that CT relates to coding or programming, but considering CT as knowing how to program may be too limiting. According to the National Research Council (2010), *everyone* should acquire CT, not only programmers. CT skills include managing information effectively and efficiently with technologies in our data-driven era (Burke, O'Byrne, & Kafai, 2016; Kim, Kwon, & Lee, 2014; Lu & Fletcher, 2009; Sanford & Naidu, 2016; Wing, 2010). A workforce with individuals possessing CT skills increases the competitiveness of the United States in the world economic market (NRC, 2010).

Although many programs claim to teach coding skills, rarely do they look deeply at the ways of thinking used in CT. Analogous to inquiry as a way of thinking scientifically, CT is a set of practices that are entwined with ways of looking at problems that result in CT skills and understandings. Yet, there is no existing curriculum building a CT foundation of understanding for

* Corresponding author.
Email address: vshute@fsu.edu (V.J. Shute)

https://doi.org/10.1016/j.edurev.2017.09.003
1747-938/

young learners, as there is in Math or Science. Models are needed that help highlight CT within current classroom practices, because there are too few opportunities to fit new content within existing school curricula. Many problems presented in current curricula, however, can be approached with CT.

Possessing good CT skills may be a motivator for students to pursue computer science (Allan, Barr, Brylow, & Hambrusch, 2010) and other STEM-related majors (Sneider, Stephenson, Schafer, & Flick, 2014). CT has also been linked to creativity and innovation (Mishra, Yadav, & the Deep-Play Research Group, 2013; Repenning et al., 2015), and it has important applications in other STEM areas (Barr & Stephenson, 2011; Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013). An exact definition of CT, however, remains elusive (Barr, Harrison, & Conery, 2011; Grover & Pea, 2013).

In this paper, we discuss the various definitions of CT emerging from different disciplines, and we present a definition of CT in terms of how K-12 educators might think of building a solid foundation for CT in young learners. We break down CT into the components most often cited in the literature and propose a model for embedding CT learning and assessment within K-12 curricula.

1.1. Definitions of computational thinking

Computational thinking (CT) stems back to the constructionist work of Seymour Papert (Papert, 1980, 1991) and was first coined as a term in a seminal article by Wing (2006). She explained that CT entails "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). As such, it represents an ability to analyze and then solve various problems. Her arguments provided a fresh perspective on the relationship(s) between humans and computers, and gave rise to a wave of research on CT.

The most oft-cited definition of CT comes from Cuny, Snyder, and Wing (2010), noting that CT is a thinking process where "… solutions are represented in a form that can be effectively carried out by an information-processing agent" (as cited in Wing, 2010, p. 1). This relates not only to well-structured problems, but also ill-structured problems (i.e., complicated real-life problems whose solutions are neither definite nor measurable). Other researchers have come up with their own definitions relative to their particular research areas. For instance, Barr et al. (2011) concluded that in K-12, CT involves problem-solving skills and particular dispositions, such as confidence and persistence, when confronting particular problems. Berland and Wilensky (2015) defined CT as "the ability to think with the computer-*as*-tool" (p. 630) and suggested using "computational perspectives" as an alternative to "computational thinking" to emphasize that CT can be constrained by contexts. Additionally, CT has been defined as "… students using computers to model their ideas and develop programs" (Israel, Pearson, Tapia, Wherfel, & Reese, 2015, p. 264), explicitly linking CT to programming skills.

As described above, CT definitions vary in their operationalization of CT in certain studies, and are not particularly generalizable (e.g., Berland & Wilensky, 2015; Ioannidou, Bennett, Repenning, Koh, & Basawapatna, 2011; Israel et al., 2015). The definition of CT is evolving as researchers begin to aggregate knowledge about CT.

1.2. Goals and focus

In this paper, we intend to explore the scope and complexity of CT, and establish a clear definition and framework that will aid in the development of CT pedagogy and assessment, particularly for K-12. At the end, we provide current research examples and ideas for future research. Our review addresses the following questions: (1) What are the major characteristics and components of CT? (2) What interventions are (or may be) used to train/enhance CT? (3) What types of measures are used to assess CT? and (4) What are the main theoretical frameworks/models of CT? Again, our main goal is to derive a general model of CT that may be used as a framework towards assessing and supporting CT. Our model is derived from an extensive literature review and serves to guide the development of CT pedagogy and assessment in educational settings.

2. Method

2.1. Procedure

We began by collecting research papers relevant to CT, including peer-reviewed publications and proceedings, plus one academic report from a national workshop on CT. We searched various databases using the keywords "computational thinking" (quotation marks included), and specified that the term occurred either in the title or in the abstract.

The following online databases and web sites were employed in this search-collection effort:

- *ERIC*: The Educational Resources Information Center (ERIC) consists of *Resources in Education Index*, and *Current Index to Journals in Education*. ERIC is a broad and popular database containing educational reports, evaluations, and research.
- *PsycINFO:* This site is hosted by the American Psychological Association, which carries citations and summaries of scholarly journal articles, book chapters, books, and dissertations, in psychology and related disciplines.
- *JSTOR*: A database of back issues of core journals in the humanities, social sciences, and sciences. The gap between the most recently published issue of any journal and the date of the most recent issue available in JSTOR is from 2 to 5 years.

- *Google Scholar*: This web site was employed to search for and acquire specific references. Google Scholar is a web site providing peer-reviewed papers, theses, books, abstracts, and articles from academic publishers, professional societies, preprint repositories, universities, and other scholarly organizations.

We started the timeline in 2006, when Wing published her seminal article on CT, signaling the beginning of a corpus of research and projects on the topic. After collecting relevant papers, we screened the articles again, sorting them into conceptual papers and empirical studies. Conceptual papers discussed the general features of CT, providing a theoretical framework or suggesting instructional practices to integrate CT into education. Empirical studies tended to test and justify specific interventions and measure(s) of CT via qualitative and quantitative research designs. Finally, we integrated the findings from the literature review with empirical research to create our competency model[?1] for CT, aiming to facilitate and assess CT in educational settings.

2.2. Inclusion and exclusion criteria

In all, approximately 70 documents – empirical as well as theoretical papers – were initially collected. From this set, a total of more than 45 documents met the criteria for inclusion in the literature review. The inclusion criteria consisted of relevancy of the documents to the research topics in this article (e.g., computational thinking skills--characteristics and processes, models, assessments, and interventions). Both experimental and non-experimental studies were included. We created a table summarizing the full set of 70 papers collected, then deleted papers from the list for the following reasons: (a) poor quality research described in the paper (e.g., excessive statements and assumptions presented with inadequate support), (b) tangential or no focus specifically on CT, (c) empirical papers that measured something other than CT as the outcome, and (d) pilot studies that were low quality and/or reported a small sample size.

3. Results from the literature review

Our review consists of four main parts. We begin by addressing the distinct characteristics of CT. Next, we examine interventions to support CT followed by CT assessments used in K-12 and higher education. We end with a summary of CT research models.

3.1. Characteristics of computational thinking

In this section, we define CT and distinguish it from other types of thinking (e.g., systems thinking and mathematical thinking). We also discuss CT's relationship with computer science.

3.1.1. Components of CT

Wing (2006) argued that CT does not mean to think like a computer; but rather to engage in five cognitive processes with the goal of solving problems efficiently and creatively. These include:

1. *Problem reformulation* – Reframe a problem into a solvable and familiar one.
2. *Recursion* – Construct a system incrementally based on preceding information.
3. *Problem decomposition* – Break the problem down into manageable units.
4. *Abstraction* – Model the core aspects of complex problems or systems.
5. *Systematic testing* – Take purposeful actions to derive solutions.

Abstraction is the main element undergirding CT (Wing, 2008), where people glean relevant information (and discard irrelevant data) from complex systems to generate patterns and find commonalities among different representations (Wing, 2010). Abstraction has layers, so one must define each layer and clarify the relationships between layers. This involves: (a) abstraction in each layer, (b) abstraction as a whole, and (c) interconnection among layers. For instance, defining an algorithm is one kind of abstraction—the "abstraction of a step-by-step procedure for taking input and producing some desired output" (Wing, 2008, p. 3718).

In addition to abstraction and problem reformulation, Barr et al. (2011) argued that CT also consists of data organization and analysis, automation, efficiency, and generalization. Automation is making a process or system operate automatically; efficiency means creating optimal solutions; and generalization involves applying CT strategies to solve new problems. Barr and colleagues also included certain dispositions important to CT, such as confidence, persistence in relation to solving complex tasks, and the ability to work well in teams. Similarly, CT described by Bers, Flannery, Kazakoff, and Sullivan (2014) includes abstrac-

[1] A competency model refers to a collection of knowledge, skills, and other attributes that comprise a particular construct (such as CT). It answers the question: What do you want to say about the person at the end of the assessment? Variables in the competency model are usually called "nodes" and describe the set of variables on which inferences are based.

tion, generalization, and trial and error activities. They particularly emphasize the importance of debugging (i.e., identifying and fixing errors when solutions do not work as expected).

In a comprehensive report by the National Research Council, CT consists of five elements essential and universal across domains (NRC, 2010): (1) hypothesis testing, (2) data management, (3) parallelism, (4) abstraction, and (5) debugging. When solving a complex problem in any domain, one should generate and test hypotheses systematically to understand how the system works. It is impossible to test all possibilities, so selecting the right parameters to test is important. Data management involves gathering data from various sources, processing data patterns, and representing data in a meaningful way. Parallelism refers to simultaneously processing information from multiple sources or dimensions. Abstraction focuses on modeling the workings of a complex problem/system. Finally, debugging refers to finding and fixing errors after building up particular models. Recently, Anderson (2016) explicated five CT components: (1) problem decomposition, (2) pattern recognition, (3) abstraction (i.e., generalization of repeated patterns), (4) algorithm design for solutions, and (5) evaluation of solutions (i.e., debugging).

Based on the foregoing review, researchers have come up with similar CT constituent skills. The components common among researchers are: decomposition, abstraction, algorithms, and debugging. In our competency model, CT similarly consists of decomposition, abstraction, algorithms, debugging, as well as iteration and generalization (see detailed definitions, subcategorizations, and justifications in Section 4).

3.1.2. Differences between CT and other types of thinking skills

Researchers are also studying the differences and similarities between CT and other types of thinking (e.g., Barr et al., 2011; Grover & Pea, 2013). In this section, we compare CT with mathematical, engineering, design, and systems thinking.

Mathematical thinking involves the application of math skills to solve math problems, such as equations and functions (Sneider et al., 2014). Harel and Sowder (2005) defined mathematical thinking as global across many problems and "… governs one's ways of understanding" (p. 31). Mathematical thinking consists of three parts: beliefs about math, problem solving processes, and justification for solutions. The main commonality between CT and mathematical thinking is problem solving processes (Wing, 2008). Fig. 1 shows the full set of shared concepts of computational and mathematical thinking: problem solving, modeling, data analysis and interpretation, and statistics and probability.

Engineering involves skills needed to build or transform things in the world in order to construct better lives (Bagiati & Evangelou, 2016) as well as "applied science and math, solving problems, and making things" (Pawley, 2009, p. 310). The overlap between CT and engineering includes problem solving, along with understanding how complex systems work in the real world (Wing, 2008). However, unlike engineering, CT is intended to help humans understand complex phenomena through simulations and modeling, which can transcend physical constraints (Wing, 2010). To summarize, CT, mathematical thinking, and engineering stem from different disciplines. Their differences lie in specific applications in their own domain.

Design thinking requires one to solve problems by thinking as a designer (Razzouk & Shute, 2012). Computational thinking and design thinking both focus on problem solving. Design thinking, like engineering, focuses on product specification and the requirements imposed by both the human and the environment (i.e., practical problems). Again, CT is not limited by physical constraints, enabling people to solve theoretical as well as practical problems.

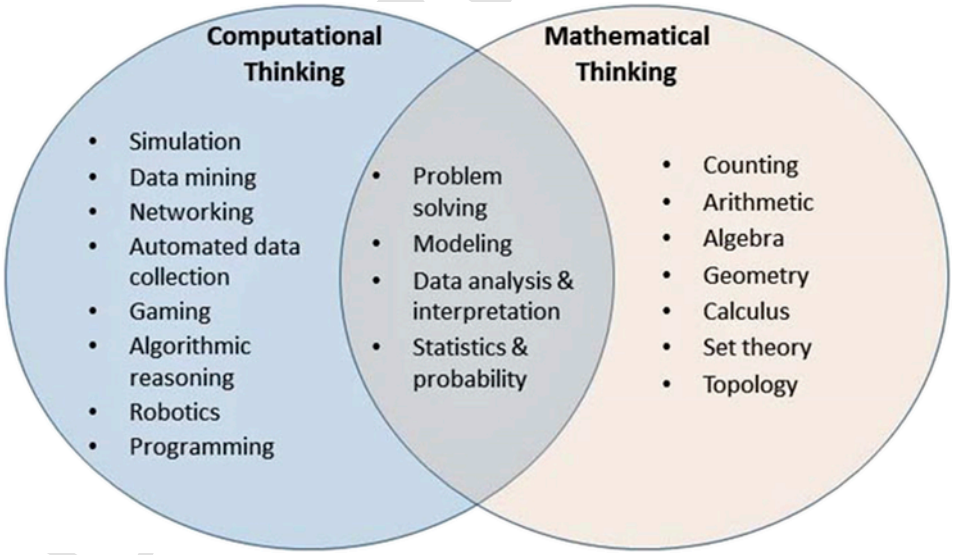

Fig. 1. Similarities and differences between CT and mathematical thinking. Adapted from Sneider et al. (2014).

Systems thinking refers to the ability to understand various relationships among elements in a given environment (Shute, Masduki, & Donmez, 2010). According to the competency model developed by Shute et al. (2010), people with system thinking skills should be able to: (a) define the boundaries of a problem/system, (b) model/simulate how the system works conceptually, (c) represent and test the system model using computational tools, and (d) make decisions based on the model. Although CT and systems thinking both involve understanding and modeling systems, CT is broader than systems thinking, which focuses on identifying and understanding the workings of a system as a whole. CT aims to solve problems efficiently and effectively, going beyond modeling and understanding to include algorithmic design, automation, and generalization to other systems/problems. In conclusion, CT is an umbrella term containing design thinking and engineering (i.e., efficient solution design), systems thinking (i.e., system understanding and modeling), and mathematical thinking as applied to solving various problems.

3.1.3. Relationship of CT with computer science and programming

Another area in need of clarification involves the relationships among CT, computer science, and programming (Czerkawski & Lyman, 2015). And although CT originates from computer science (Wing, 2006), it differs from computer science because it enables people to transfer CT skills to domains other than programming (Berland & Wilensky, 2015).

CT skills are not the same as programming skills (Ioannidou et al., 2011), but being able to program is one *benefit* of being able to think computationally (Israel et al., 2015). For instance, Shute (1991) examined the relationships among programming skills, prior knowledge, and problem-solving skills, within 260 college and technical school students. Participants who had no prior programming experience learned programming skills via an intelligent tutoring system. Several assessments were administered to measure learners' incoming knowledge (i.e., math and word knowledge), cognitive skills (i.e., working memory and information processing speed), and particular aspects of problem-solving skills (e.g., problem identification, sequencing, and decomposition). In addition, a criterion test was used to measure learners' programming skills and knowledge after the intervention. Results from a factor analysis and hierarchical regression showed that working memory, problem identification, and sequencing solutions are the best predictors of programming skill acquisition. Thus, CT and programming skills as well as problem solving are closely related.

In general, the field of computer science is broader than just learning about programming, and CT is broader than computer science (NRC, 2010; Wing, 2006) in that CT includes a way of thinking about everyday activities and problems. In line with this perspective, Lu and Fletcher (2009) proposed that teaching CT should not even use programming languages; instead the language should be based on notions that are familiar to most students to engender the acquisition of concepts like abstraction and algorithms. Like with most of the research targeting CT, its particular relationship to computer programming is evolving.

3.2. Interventions to develop computational thinking

Researchers have attempted to leverage programming tools, robotics, games/simulations, and non-digital interventions to teach CT knowledge and skills in various educational contexts. The target population ranges from kindergarten to undergraduates. Table 1 summarizes all of the research studies we reviewed, arrayed by intervention tools.

3.2.1. Research on CT using programming tools

Due to its close relationship with computing and programming, CT skills appear to be improved via computational tools, such as Scratch (MIT, 2003). Regarding the equivalence of programming skills to CT skills, Cetin (2016) compared the effects of employing Scratch (experimental group) with C language (control group) to teach programming concepts to pre-service IT teachers. This experiment lasted for six weeks, and the participants (n = 56) completed pre- and posttests relative to their achievement on and attitudes toward programming. Additionally, nine participants per group were randomly selected to attend semi-structured interviews. Results showed the experimental group performed significantly better than the control group in terms of programming knowledge and skills, but there were no between group attitudinal differences.

To promote algorithmic thinking via Scratch, Grover, Pea, and Cooper (2015) designed a seven-week Scratch-based CT course for 7th and 8th graders (n = 54). Similar to Cetin's (2016) study, CT gain was measured as pretest to posttest improvement on programming skills. The aim of this quasi-experiment was to see which approach (face-to-face instruction vs. face-to-face plus online) supported deep learning relative to computational concepts, such as algorithms, loops, conditionals, and decomposition. The two conditions were matched in terms of receiving comparable instruction for the same duration of time (i.e., four days per week, 55 min per day, across seven weeks). Findings revealed that both approaches lead to significantly higher CT gains, and students in the face-to-face plus online group performed significantly better than those in the face-to-face group. Moreover, both groups successfully transferred their programming knowledge and skills to text-based programming tasks.

The strength of Scratch is to help young people learn to think creatively, reason systematically, and work collaboratively, and thus is suitable to facilitate CT. It is easy to use with its drag-and-drop programming method, and provides a meaningful learning environment where learners engage in specific contexts. Alice (Carnegie Mellon University, 1999) functions similarly, equipped with ready-made code blocks. Compared with Scratch, it focuses on creating 3D programming projects. It also can be utilized to train CT. For example, Denner, Werner, Campe, and Ortiz (2014) randomly assigned 320 middle school students to either a dyadic work group or individual programming. Students' CT skills were measured by their ability to accomplish Alice tasks during one semester's course. Results demonstrated that students working collaboratively achieved significantly higher CT scores

Table 1
Articles reviewed sorted by intervention tools.

Study	Setting	Participants	Intervention		Content	Measures
			Program	Duration		
Cetin, 2016	Higher education	56 Undergrads (pre-service teachers)	Scratch; C language	6 weeks	Computer science	Multiple-choice; open-ended questions; surveys; interviews
Grover et al., 2015	Middle school	54 7th and 8th graders	Scratch	7 weeks	CT course	Multiple-choice; quizzes; assignments; text-based coding; projects
Denner et al., 2014	Middle school	320 students	Alice (solo vs. pairs)	One semester	Programming	Surveys; tasks
Werner et al., 2012	Middle school	311 students	Alice (solo vs. pairs)	One semester	Programming	Surveys; tasks
Atmatzidou & Demetriadis, 2016	High school	164 students	Lego Mindstorms Robotics	One year	CT course	Questionnaires; think-aloud; interviews
Berland & Wilensky, 2015	Middle school	78 8th graders	Lego vs. virtual robotics	5 days	Science	Log files; tests; questionnaires
Basu et al. (2017)	Middle School	98 6th graders	CTSiM platform	13 days	Ecology, CT	Questions; explanations; constructing algorithms
Bers et al., 2014	Kindergarten	53 students	TangibleK Robotics	20 h	CT course	Questionnaires
Jenkins, 2015	Middle school	51 students	Snap; Small Basic	3 h	Poem	Fill-in-the-blanks; questions;
Kim et al., 2013	Higher education	110 sophomores (pre-service elementary teachers)	PPS	15 weeks	Computer science	Multiple-choice; surveys
Yadav et al., 2014	Higher education	141 undergrads	CT module	One week	Educational psychology	Questionnaires

than students working alone. And collaboration was especially beneficial to students with minimal programming experience. These findings are consistent with those reported in an earlier experiment conducted by Werner, Denner, Campe, and Kawamoto (2012) testing 311 middle school students.

Basu, Biswas, and Kinnebrew (2017) similarly viewed CT constructs as programming-related concepts, such as sequencing, loops, and variables; and they considered iteration, problem decomposition, abstraction, and debugging as CT practices. They designed the CTSiM platform to integrate ecology and CT learning for 6th graders. Basu et al. employed a pretest/posttest design to test the effectiveness of scaffolding provided by a virtual agent embedded in CTSiM. Both the experimental (n = 52) and control (n = 46) groups learned CT and ecology via CTSiM, but only the experimental group received scaffolding. That is, when students failed a given task 3–5 times, scaffolding was triggered. In that case, the virtual agent provided conversation prompts, and the students answered by choosing one of the options, which in turn triggered a response from agent. Agent-student conversations were pre-programmed to help struggling students. In the control group, such functionality was disabled. The ecology tests required students to choose correct answers and provide rationales for their answers. The CT tests required students to predict outcomes after reading program segments, and build algorithms with CT constructs to model scenarios. Pre- and posttests showed significant gains in both groups on ecology and CT. And when pretest scores were used as a covariate, the experimental group (with scaffolding) significantly outperformed the control group in ecology learning gains and CT skills, with effect sizes of 0.36 and 0.31, respectively.

3.2.2. Research using robotics

Another fruitful area in which to develop CT skills is robotics, which is also closely related to programming. Lego robotics is particularly popular. For example, *Lego Mindstorms* (http://www.lego.com/en-us/mindstorms) was used to improve high school students' (n = 164) CT skills for a year (Atmatzidou & Demetriadis, 2016). The 44 two-hour sessions focused on developing the following CT skills: decomposing problems, abstracting essential information, generalizing the solution to different problems, creating algorithms, and automating procedures.

Solving robot programming problems revealed students' CT skills, which were measured via rubrics related to the quality of problem-solving performance. Quantitative data showed that all participants, regardless of their age or gender, improved similarly on their CT skills following the intervention. Qualitative data generated from interviews and think-aloud protocols confirmed the effectiveness of robotics to develop CT concepts and solve problems effectively.

Lego robotics is a physical activity. Berland and Wilensky (2015) compared the effects of Lego robotics (n = 34) versus virtual robotics (n = 44) among 8th graders. Pre- and posttests on pseudo-code programming measured CT gains. Both groups of students significantly improved their CT skills, and there was no significant posttest difference between the two groups. It is interesting to note that the two groups perceived the problems differently. That is, the virtual robotics group tended to perceive the problem as a whole and then attempt to decompose the problem down to its details; while the physical robotics group initially focused on the constituent parts.

How early can children learn CT skills? One study looked at teaching kindergarten students (n = 53) CT skills (Bers et al., 2014). These researchers developed the *TangibleK Robotics* curriculum (http://ase.tufts.edu/DevTech/tangiblek/) which included 20 h of instruction and one final project to measure students' development of CT in terms of debugging, sequencing, loops, and conditionals. However, repeated measures between tasks did not reveal any linear CT development in kindergarten students. The researchers speculated that either (a) kindergarten students are simply too young to develop CT skills, or (b) the robotics intervention was too difficult for them (Bers et al., 2014). Therefore, it is important to teach CT concepts in a way suitable for students' developmental stages.

3.2.3. Research using game design and other intervention tools

AgentSheets is an authoring tool that uses game design to teach CT, as well as to promote students' interest and skills in computer science (see Ioannidou et al., 2011 for details). AgentSheets provides a low entry bar for inexperienced students, yet does not restrain students with advanced abilities from creating complex game systems. In one study, teachers and community college students in two summer workshops learned how to animate interactions among objects via programming, using five AgentSheets design tasks (Basawapatna, Koh, Repenning, Webb, & Marshall, 2011). Unlike other studies, this research measured CT based on students' knowledge of relevant patterns, defined as object interactions, such as collision and absorption. These CT patterns represent the intersection of games and science simulations. After two weeks of intervention, the authors tested the participants' CT skills with eight questions, each representing a phenomenon sharing CT patterns across the five games (e.g., sledding collision). Participants needed to identify and justify usage of the CT patterns to create simulations of the phenomena. Generally, all participants did very well in correctly identifying CT patterns, as indicated by the percent correct of their responses. However, the authors noted that the results do not necessarily mean that students could transfer CT knowledge from game design to simulations of scientific phenomena.

Teaching CT does not necessarily require digital tools. A paper-and pencil programming strategy (PPS) was created for non-computer science majors (n = 110) (Kim, Kim, & Kim, 2013). It enabled participants (n = 55) in one group to choose their preferred way to visualize and represent programming ideas (e.g., via concept maps, tables, words, or symbols). Its effects were compared to another group, who engaged in typical course instruction (n = 55) for 15 weeks. Typical instruction in this case refers to the regular programming course, using LOGO (Softronics, Inc., 2002). Pre- and posttests revealed that students using the paper-and-pencil programming strategy showed significantly better understanding of CT and more interest in CS, than those learning via typical course instruction. The authors therefore argued that teaching CT without computers might be effective and efficient for non-computer science majors.

A lecture-based CT instructional module could also change pre-service teachers' understanding of and attitudes toward integrating CT in their classrooms (Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). The researchers tested a 100-min module focused on CT concepts such as decomposition, abstraction, algorithms, and debugging. After training, participants completed questionnaires with open-ended questions related to CT and its connections with various disciplines. Compared to the no-treatment control group (n = 154), a higher percentage of those in the experimental group (n = 141) who received the CT training considered CT a way of thinking to solve problems, with or without the help of computers. Moreover, they realized that CT was not restricted within the field of computer science. Conversely, the control group viewed CT narrowly—as using computers to complete specific computing tasks and restricted to subjects involving computers, like computer science and math.

3.2.4. CT skills undergirding programming, robotics, and game design

The rationale for using programming, robotics, and game design to improve CT skills is because each one of these areas emphasizes various CT components. For instance, writing computer programs requires the analysis of the problem (e.g., determining the goal to achieve), and then breaking the problem down to its constituent processes (e.g., identifying sub-goals and associated steps to achieve the goal). Writing efficient programs requires abstraction and generalization. For instance, if one step needs to be repeated four times, an efficient solution involves looping the step rather than writing the same code four times. Additionally, part of the programming code may be reused within similar problems, with minor adjustments, rather than rewriting a new program from scratch. And to test the correctness and efficiency of a program, debugging is necessary. That is why programming is frequently used to promote CT skills.

Similarly, robotics provides learners with tactile experiences to solve problems via CT skills. Learners need to identify the general problem/goal for the robot, then decompose the problem (e.g., figuring out the number of steps or sub-goals to accom-

plish the goal). Towards this end, learners develop algorithms for the robot so that it can follow the instructions and act accordingly. When the robot does not act as expected, debugging comes into play. Debugging requires the iterative processes of systematic testing and modifying.

Finally, as with programming and robotics, game design and gameplay entail various goals for players to solve. These can be smaller goals represented within levels of the game, and larger goals represented by boss levels and/or part of the narrative. To succeed, players need to derive solution plans, and if a plan fails, then a modified plan is developed. By systematically testing various plans, players find the most effective strategy to overcome challenges in the game. Moreover, skillful players are able to adopt strategies used before to solve new problems. Thus, problem decomposition, systemic testing and debugging, generalization, and iteration are skills required in gaming and are important components of CT.

3.3. Assessment of computational thinking

Because of the variety of CT definitions and conceptualizations, it's not surprising that accurately assessing CT remains a major weakness in this area. There is currently no widely-accepted assessment of CT. This makes it difficult to measure the effectiveness of interventions in a reliable and valid way (Grover & Pea, 2013; Kim et al., 2013; Settle et al., 2012). Having no standard CT assessment also makes it very difficult to compare results across various CT studies. Researchers tend to develop and apply their own CT measures in various studies, depending on their particular operationalization of CT (Kim et al., 2013).

Questionnaires and surveys are the most commonly used measure for knowledge of and/or attitudes towards CT (e.g., Atmatzidou & Demetriadis, 2016; Denner et al., 2014; Yadav et al., 2014). For instance, Cetin (2016) used surveys to measure preservice teachers' conceptual understanding of programming constructs, like arrays and loops, as well as attitudes towards computer science. Yadav et al. (2014) similarly used surveys to examine pre-service teachers' understanding of CT and their attitudes toward integrating CT into their teaching in the future. Kim et al. (2013) designed questionnaire items to specifically measure college sophomores' logical thinking, as well as conceptual understanding of, and interest in computer science. Viewing CT as the core of computational literacy, Jun, Han, Kim, and Lee (2014) administered questionnaires on computational literacy among elementary school students.

Other researchers have conducted interviews and observations with participants to understand the development of CT skills (e.g., Cetin, 2016; Israel et al., 2015). Some of the researchers assessed CT skills through projects and related tasks. For instance, Denner et al. (2014) and Werner et al. (2012) developed a performance-based assessment to measure CT by letting middle schoolers complete projects, such as a three-stage task designed with Alice. Students needed to solve particular problems (e.g., saving a fairy from a magic forest). Other researchers tested learning gains of subject area knowledge, after incorporating CT into disciplinary instruction (e.g., Cetin, 2016; Sengupta et al., 2013).

3.3.1. Scratch-based assessments

Several groups of researchers have used Scratch to develop systematic CT assessments. Brennan and Resnick (2012) claimed that Scratch-based projects could support long-term learning by making learning meaningful within a particular context. They also argued for a dynamic measure to assess Scratch users' (8–17 years old) CT abilities over time, revealing the progression of learning and measuring both conceptual understanding and application of CT skills. Their CT assessment included formative analysis, interviews, and design projects. The formative analysis involved evaluating users' portfolios to see the development of CT via projects completed over time. Interviews allowed the researchers to dig deeper into the thinking processes of users. There were also three sets of design projects, varying in difficulty, which were employed as performance measures. Each set included two tasks of the same difficulty level but with different representations. Users had to choose one task from each set to describe and debug.

To align with a particular Scratch-based CT course for middle schoolers, Grover et al. (2015) designed a system of formative and summative assessments to support and measure CT skills. Multiple-choice items and quizzes throughout the course resembled the pseudo code in Scratch. The items were used to consolidate students' knowledge of programming concepts, with timely feedback serving to explain various ideas and procedures. Course assignments were completed using Scratch and were evaluated by specific rubrics. The summative measure was a final test with multiple-choice items plus open-ended questions, along with a final project that was completed with a partner. In addition, the researchers developed a test based on AP programming exams to test whether students could transfer what they had learned to real text-based programming code.

Another Scratch-based assessment is called PECT (progression of early computational thinking), designed for elementary school students (Seiter & Foreman, 2013). This represents a guideline to measure three facets of CT. The first facet included *evidence* variables, which were collected directly from the command categories embedded in Scratch (e.g., operators and conditionals). The second facet was *pattern* variables (e.g., motion and interactivity), which are broader than the specific evidence from the pseudo code library in Scratch. The last facet was *CT concepts*, such as decomposition, abstraction, and algorithms. Students' proficiency levels (i.e., basic, developing, and proficient) per facet were measured according to rubrics.

3.3.2. Game/simulation-based assessment

Other researchers have proposed assessing CT via CT pattern analysis (CTPA) in K-12 schools. CT patterns are abstract programming concepts related to object interactions that students use to develop games and simulations (Basawapatna et al., 2011;

Ioannidou et al., 2011). Building object interactions in games required 12 CT patterns, defined by Ioannidou et al. (2011). Repenning et al. (2015) subsequently refined those 12 patterns to nine aspects, to measure transfer of CT from gaming to modeling scientific phenomena in STEM courses. CTPA is used to generate spider web-like graphic reports of the nine patterns, by comparing students' products with exemplars. This type of report provides visual information about the nature and degree of students' abilities, and which aspects need improvement. Reports over time can demonstrate students' progression of CT skills. Thus, teachers could provide instructional support given access to such graphic reports.

3.3.3. Validated CT scales for generic usage

The aforementioned assessments were designed solely for the purpose of a particular study based on a specific intervention. Recently, researchers have developed a scale to measure CT (Román-González, Pérez-González, & Jiménez-Fernández, 2016). This scale includes 28 items and takes about 45 min to complete. It focuses on computational concepts, like directions and sequences, loops, if conditionals, and simple functions. The reliability of this scale is 0.79. The authors tested its validity among 1251 Spanish students from 5th to 10th grades. The results showed significant correlations with other standardized tests on mental abilities and problem-solving skills. The problem-solving test explained 44.7% of the variance in the CT scale, while the mental ability test explained 27%. Thus, the authors concluded that CT is more similar to problem solving skills and less aligned with reasoning, spatial, and verbal abilities.

Similar efforts have been conducted and reported by Korkmaz, Çakir, and Özden (2017). The researchers developed a CT scale comprised of 29 items, measuring five factors (i.e., creativity, cooperation, algorithmic thinking, critical thinking, and problem solving). They first conducted exploratory factor analysis ($n = 726$) and then confirmatory factor analysis ($n = 580$) among Turkish college students. The results showed good data fit indices and evidence that the items measured targeted constructs. The reliabilities for the five factors ranged from 0.73 to 0.87. This scale measures CT differently from the previous one because the researchers operationalized CT differently, based on their own CT framework (described in Section 3.4.).

In conclusion, this review of CT assessments showcases the pressing need for a reliable and valid assessment of CT skills across educational settings, so that researchers can determine whether their CT interventions are effective or not, rather than just looking at learning gains or attitudinal changes via homegrown measures. The assessment of CT also calls for dynamic information that can reflect learners' abilities and progression over time.

3.4. Computational thinking models

Having reviewed CT components, interventions, and assessments, this section examines theoretical frameworks of CT. As with CT definitions and assessments, there are no agreed-upon models or frameworks for CT (Atmatzidou & Demetriadis, 2016). In this section, we examine four CT models proposed by researchers.

First, Atmatzidou and Demetriadis (2016) presented a simple, descriptive CT model, based on their operationalization of CT in previous studies. The model consists of five facets: abstraction, generalization, algorithms, modularity, and decomposition. It also provides examples of behaviors that students should demonstrate as evidence for each facet (see Atmatzidou & Demetriadis, 2016, p. 664 for details). Briefly, abstraction means distilling the core patterns from complicated systems; generalization involves applying problem-solving strategies to different contexts; algorithms refer to ordered steps/instructions to implement solutions; modularity means the automation of problem-solving solutions; and decomposition entails the breakdown of complex systems/things into manageable pieces.

The second model consists of CT concepts and abilities that should be incorporated into K-12 courses like math, science, social studies, and language arts (see Barr & Stephenson, 2011, p. 52). It defines core CT facets (i.e., data collection, data analysis, data representation, decomposition, abstraction, algorithms, automation, parallelism, and simulation) across various disciplines. These CT facets should have different representations in different subjects (e.g., representing data with charts or tables in math, and representing linguistic patterns in language arts). However, the specific demonstrations of CT facets within particular disciplines are not clearly stated in the paper. Moreover, the provided examples of teaching practices are too vague for teachers to actually employ them. For example, abstraction in science class is described only as modeling a physical entity. The weakness of this model stems from (a) no clear definitions per CT facet making operationalization very difficult, and (b) failure to distinguish concepts from abilities (e.g., abstraction is both a concept and an ability). Thus, this model has room for improvement before serving as a guideline for teachers in K-12 education.

Brennan and Resnick (2012) presented a CT framework within the context of using Scratch to facilitate CT. They categorized CT into three areas—concepts, practices, and perspectives. Table 2 shows our summary of their framework.

A review paper on integrating CT in K-12 settings by Lye and Koh (2014) was based on the Brennan and Resnick framework. Later, Zhong, Wang, Chen, and Li (2016) revised that model (see p. 565 for details) by adding instruction into the CT concepts, and iteration into the CT practices. They also rephrased CT perspectives to emphasize creativity and collaboration. However, they did not elaborate on those modifications, which makes it hard to interpret the revised model.

A fourth recent model aims to merge CT and regular classroom instruction (Weintrop et al., 2016). The researchers analyzed 34 lesson plans for high school math and science courses by coding teaching practices that related to CT facets. Then they categorized the specific facets into broader CT practices, and refined the categorization after consulting with lesson plan designers, in-service high school teachers, and experts in CT and curriculum design. Finally, they came up with a taxonomy containing four

Table 2
Summary of CT framework proposed by Brennan and Resnick (2012).

CT Concepts	*Sequences*: Instructions for computer to execute behaviors
	Loops: Repeat the same instruction for a specified number of times
	Parallelism: Concurrence of multiple instructions
	Events: Triggers for certain actions to happen to create interactive environments
	Conditionals: Constraints on execution of instructions, allowing for different outcomes
	Operators: Mathematical and string operations
	Data: Data storage, retrieval, and update
CT Practices	*Being incremental and iterative*: Iterative processes to design and implement solutions, step by step
	Testing and debugging: Trial and error processes to test and remove malfunctions as warranted
	Reuse and remix: Building reusable instructions; building new products on others' work
	Abstraction and modularity: Modeling complex systems with basic elements
CT Perspectives	*Expressing*: Perception of computation as a way of expression and creation
	Connecting: Perception of computation as a way of interacting and working with others
	Questioning: Raising questions and using technology to solve real life problems

CT categories with 22 CT practices (see Table 3). Their taxonomy is based on specific classroom activities and presented concrete examples of CT classroom activities, showing how lesson plans can be designed by following the taxonomy. This model is tailored to STEM courses in high school, and shows promise with regard to integrating CT in secondary education. However, more research is needed to validate this model.

Lacking a consistent model might cause problems in designing interventions to support CT learning and in assessing CT knowledge and skills in various educational settings. This leads to the next part of this paper; namely, our proposed competency model of CT, which can be used to guide assessment and support of CT skills.

4. Our computational thinking definition and framework

Drawing from the aforementioned definitions and models, we define CT as *the conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts*. As stated earlier, the components of CT that arise most often in the literature are: abstraction, decomposition, algorithms, and debugging. Our model attempts to understand the cognitive processes underlying each of these CT facets and the associated behaviors that can help us develop a competency model that can be used for the assessment of CT. We additionally identify iteration and generalization as two more skills that are important in the development of CT.

Our model of CT emphasizes the importance of approaching problems in a systematic way. *Problem decomposition* involves breaking a complex problem into smaller parts and using systematic processes to tackle each of those smaller problems. *Iterative*, systematic *debugging* ensures that each part of the smaller problems is solved efficiently and with no loose ends. *Abstraction* is the act of finding patterns within problems and solutions, and thus being in a position to *generalize* solutions for sets of

Table 3
Taxonomy of CT in STEM courses, adapted from Weintrop et al. (2016).

CT Category	CT Practice	Definition
Data Practices	Data collection	Gather data using multiple computational tools
	Data creation	Generate data for large and complex systems
	Data manipulation	Reorganize data in a meaningful way
	Data analysis	Use computational tools to analyze data and draw valid conclusions
	Data visualization	Communicate and present data in multiple ways
Modeling and Simulation	Conceptual understanding	Understand concepts deeply through modeling
	Testing solutions	Solve problems through hypothesis testing
	Model assessment	Evaluate the effectiveness of models
	Model design	Select essential elements for models
	Model construction	Implement new models or extend existing models
Computational Problem Solving	Solution preparation	Decompose and reframe problems using suitable computational tools
	Programming	Possess basic programming knowledge and skills
	Tools selection	Evaluate pros and cons of plausible computational tools
	Solution evaluation	Assess pros and cons of possible solutions
	Solution development	Develop solutions that can be applied to a wide range of problems
	Abstraction	Distill the most relevant information from a problem
	Debugging	Identify and fix errors
Systems Thinking	System investigation	Understand system functions as a whole
	Understanding relationships	Understand the operation and interrelationship of elements in a system
	Multilayered thinking	Think from multiple perspectives and levels
	Communication	Convey information effectively and efficiently
	System management	Define scope of systems and manage complexity

similar problems. Finally, *algorithm design* allows the development of re-usable tools/procedures for solving classes of problems.

Based on prior research, we have categorized CT into six main facets: decomposition, abstraction, algorithms, debugging, iteration, and generalization. The six facets of our CT model are described in Table 4.

This framing of CT underpins, and may be evidenced through, a variety of K-12 subject areas including Mathematics, Science, and even English Language Arts.

4.1. Comparison with other models

Our model focuses on the underlying conceptual foundation required to approach problems via a CT perspective and how that kind of CT perspective can be highlighted and supported in current K-12 subjects. This is in contrast to models that focus on just one specific subject area, such as Brennan and Resnick (2012) who limit their model of CT to concepts related to coding. The three facets in their model are hierarchical with the foundation comprised of programming concepts, like loops and conditionals. Their next layer relates to computing practices built on the conceptual knowledge, whereas the highest layer represents perspectives on computing (e.g., a way of expression, creation, and communication).

Weintrop et al., 's 2016 model is derived from high school teaching practices that occur in STEM courses. The four facets in their model represent a collection of possible classroom activities from which teachers can select the most relevant ones to facilitate CT acquisition. Thus, their model is restricted to high school STEM course settings, where we feel that a foundational basis for CT starts much earlier, analogous to scientific inquiry or mathematic reasoning.

4.2. Examples of emerging models of K-12 CT

Our competency-based model of CT knowledge and skills strives to inform assessment methods to measure the foundational understandings of CT in K-12 learners. Several research groups are just starting attempts to measure CT in K-12 settings. Here we present several examples, ending with an illustration of the co-authors' assessment-related work on the conceptual foundation of several important CT facets; namely problem decomposition, abstraction, and algorithm design.

As noted earlier, several research groups are studying learners' coding in Scratch. A tool called Dr. Scratch (http://www.drscratch.org/) is currently in Beta test, and used as the basis to analyze Scratch code by quantifying the usage of loops, sequences, and other logistical facets of coding outlined by Brennan and Resnick (2012).

Researchers at Stanford (Grover et al., 2015) designed and implemented a middle school curriculum called Foundations for Ad-vancing Computational Thinking (FACT). FACT strives to get a comprehensive picture of students' CT skills by examining cognitive, interpersonal, and intrapersonal variables. FACT uses pedagogical strategies to support transfer from block-based to text-based programming, along with formative and summative assessments (including quizzes and tests as well as open-ended programming assignments) related to the acquisition of computational thinking skills. Their findings show that students ages 11–14 using the FACT curriculum experience improved algorithmic learning, understanding of computing, and transfer of skills from Scratch to a text-based programming context. Building on this research, Grover (2017; see http://stemforall2017.videohall. com/presentations/872) suggests a framing of Variables, Expressions, Loops, and Algorithms (VELA) to prepare young learners for CT.

Research underway by EdGE at TERC[?2] involves studying the development of CT within the logic puzzle game, *Zoombinis*. EdGE studies how game-based behavior can predict implicit knowledge and help teachers support explicit STEM learning in the classroom (Rowe, Asbell-Clarke, & Baker, 2015). Implicit knowledge refers to knowledge that may not yet be formalized or expressed by a learner, but may be evident through actions and behaviors.

In Zoombinis, players engage in many cycles of CT strategy development that can be evidenced by their patterns of activity in the game. In the game, players are challenged to sort Zoombinis according to underlying logic rules (e.g., the puzzle named Allergic Cliffs will reject Zoombinis with certain features, or a combination of features in upper levels—see Fig. 2).

Players typically move from trial-and-error (testing arbitrary Zoombinis) to a systematic testing pattern, where they use problem-solving strategies, some of which researchers can identify as evidence of CT. For instance, players may consistently test if Zoombinis with one or more common features (e.g., those with glasses) help the student identify a pattern to solve the puzzle. This change, from trial-and-error to systematic testing provides evidence of problem decomposition. As players accumulate evidence, they may see common patterns and start testing towards a partial solution, eventually demonstrating their ability to abstract the patterns into an underlying rule of the puzzle. Successful players exhibit repeated strategies, or algorithms, implemented over a collection of puzzles. For example, once players adopt the strategy of trying one variable at a time, and then implementing singular pieces of information into a final solution, that pattern can be observed across multiple puzzles (and other problem-solving situations).

[2] TERC is a non-profit studying math and science education innovations since 1966. In 2009, the Educational Gaming Environments group (EdGE) was founded to study STEM learning in digital games.

Table 4
Summary of our CT facets and their definitions.

Facet	Definition
Decomposition	Dissect a complex problem/system into manageable parts. The divided parts are not random pieces, but functional elements that collectively comprise the whole system/problem.
Abstraction	Extract the essence of a (complex) system. Abstraction has three subcategories: (a) *Data collection and analysis*: Collect the most relevant and important information from multiple sources and understand the relationships among multilayered datasets; (b) *Pattern recognition*: Identify patterns/rules underlying the data/information structure; (c) *Modeling*: Build models or simulations to represent how a system operates, and/or how a system will function in the future.
Algorithms	Design logical and ordered instructions for rendering a solution to a problem. The instructions can be carried out by a human or computer. There are four sub-categories: (a) *Algorithm design*: Create a series of ordered steps to solve a problem; (b) *Parallelism*: Carry out a certain number of steps at the same time; (c) *Efficiency*: Design the fewest number of steps to solve a problem, removing redundant and unnecessary steps; (d) *Automation*: Automate the execution of the procedure when required to solve similar problems.
Debugging	Detect and identify errors, and then fix the errors, when a solution does not work as it should.
Iteration	Repeat design processes to refine solutions, until the ideal result is achieved.
Generalization	Transfer CT skills to a wide range of situations/domains to solve problems effectively and efficiently.

Fig. 2. Screen capture of the "Allergic Cliffs" level in Zoombinis.

To validate student learning of these facets of CT as a function of playing Zoombinis, EdGE and Empirical Games have designed a set of online assessment items that focus on several of the fundamental facets of CT: problem decomposition, abstraction, and algorithm design. Researchers are currently completing a validation study with 300 elementary and middle school learners. The assessments were conceptually and structurally the same for both elementary and middle school students, but differed in terms of difficulty (e.g., based on number of variables to consider in a pattern and size of the array for abstraction problems). The example items shown in this paper are from the elementary school version of the assessment.

The assessment for problem decomposition uses a series of progressively harder puzzles, similar to the game called Mastermind (the board game developed by Meirowitz in 1970; see https://boardgamegeek.com/boardgame/2392/mastermind) where

learners must use feedback from the game to figure out which values per graphical feature (color and/or shape and/or pattern) are required to solve a puzzle (see Fig. 3; left side is the initial problem, and right side is receiving feedback on the current test). The learner needs to drag one object at a time into the test box to find the correct color (and in more difficult levels—also shape and pattern) in as few tries as possible. Incorrect choices are placed in the "not solutions" box. Feedback per object is represented by a green check per correct color/shape/pattern, and a red X for an incorrect attribute.

For abstraction, pattern-matching puzzles require learners to identify an underlying rule. The learners drag objects from a pool on the right into the gray cells on the left to complete the pattern, and thus applying the inferred rule. Each colored shape can only appear once in the solution. See Fig. 4.

For algorithm design, players must align a sequence of arrows (in advance) to guide a character along a path in a maze that fits specified criteria. In Fig. 5, the sequencing task requires the learner to insert directional arrows, along with the number of iterations as warranted, to guide a leprechaun to a pot of gold, in the fewest steps possible, while avoiding certain obstacles.

Finally, the CT assessment for the Zoombinis study includes a set of general pattern recognition items (Raven's Progressive Matrices, RPM; Raven, 1981) as a baseline of learners' ability to infer and apply different patterns in increasingly complex situations. Raven (2000) pointed out that the RPM focuses on two components of general cognitive ability–eductive and reproductive ability. Eductive ability involves making sense out of chaos and generating high-level schema to handle complexity. Reproduc-

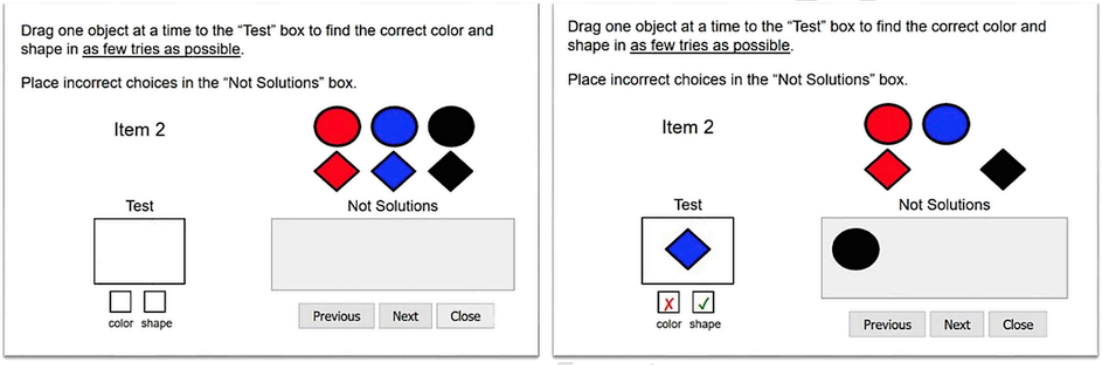

Fig. 3. Simple item in CT assessment for problem decomposition.

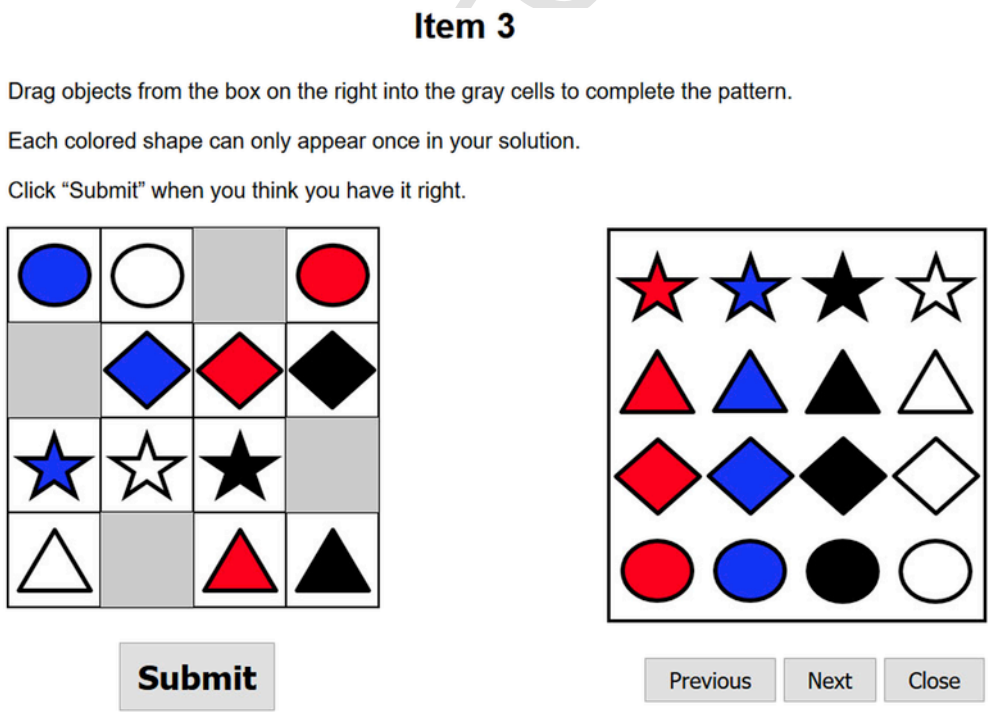

Fig. 4. Example item in CT assessment for abstraction.

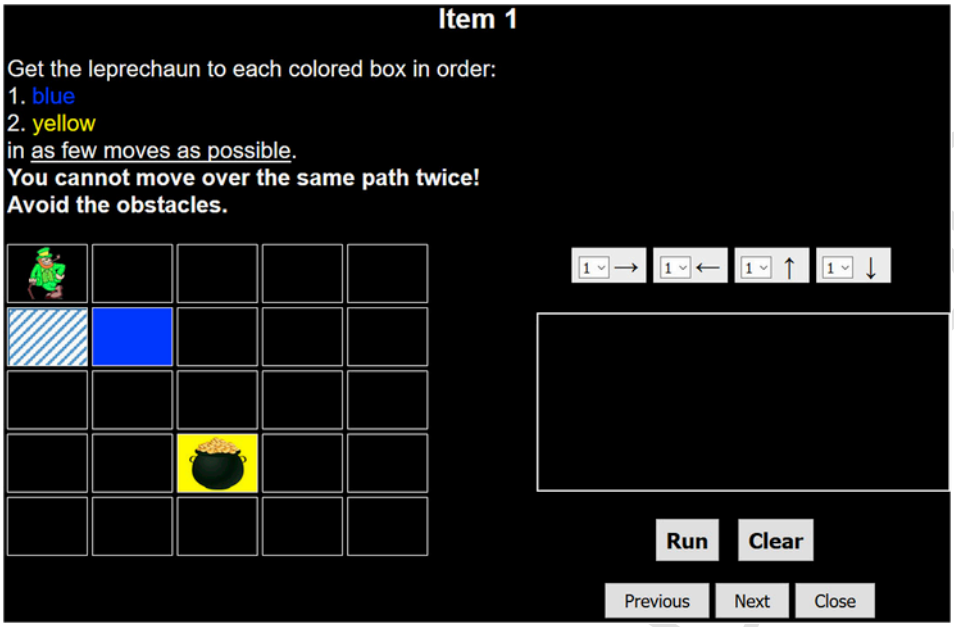

Fig. 5. Simple sequencing/algorithmic design item in CT assessment.

tive ability is the ability to recall and reproduce information that has been explicated. Rule application in general is the behavior of following the basic rules of a problem in order to solve the problem. It is an outward expression of the solvers' perception of the problems' features, via both eductive and reproductive abilities. An example is shown in Fig. 6. For each item, learners click

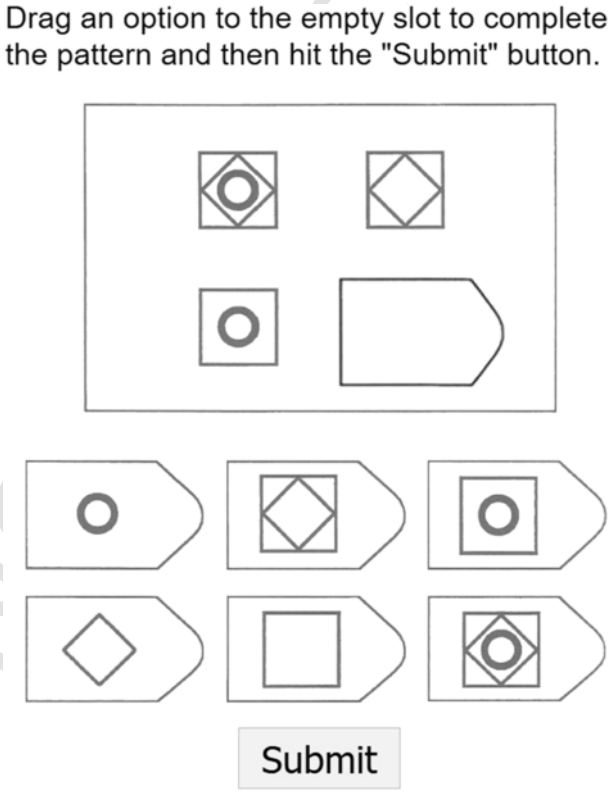

Fig. 6. Example CT assessment item from Raven's for pattern identification.

and drag the correct option among the eight possible options at the bottom to complete the pattern at the top—where the solution applies horizontally, vertically, and diagonally.

These assessments strive to examine the fundamental cognitive underpinnings of CT in an abstract and non-contextual manner. EdGE is using these assessments in combination with Educational Data Mining of game behaviors in Zoombinis, and observations of classroom CT learning to get a full picture of CT learning of students in grades 3–8.

5. Discussion

In this review, we analyzed existing definitions of CT, with a focus on literature that could help describe the conceptual foundation of CT that should guide learning and assessment in K-12 settings. We discussed how CT is similar to, and distinct from other ways of thinking (e.g., mathematical thinking and systems thinking), and described the relationships between CT and computer science. We reviewed interventions and assessments used to develop and measure learners' CT skills, ranging from kindergarten to college level. Finally, we presented examples of research striving to measure the development CT in experiences for K-12 learners. Our goal with this review paper was to synthesize the literature in order to provide a general framework of CT, including its definition, facets, and elaboration per facet, to develop a model of CT that guides K-12 CT learning and assessment.

This review of the literature has shed light on a number of gaps in CT research. For instance, a generally agreed-upon definition of CT is missing in the literature, along with a specification of its components (Wing, 2008). Inconsistent term usage occurs in many papers, such as conflating computer science, computer programming, and CT (Czerkawski & Lyman, 2015; Israel et al., 2015).

The immaturity of the field that results in this ambiguity is compounded when looking at education. Teachers are generally unfamiliar with CT and have difficulty finding connections between CT and their current curricula. Designing and developing a reliable and valid CT assessment is key to successful education of CT embedded in multiple disciplines (Grover & Pea, 2013). But lacking a standard definition to operationalize CT consequently leads to research where measurements vary greatly across studies, which makes the results less convincing and certainly difficult to compare.

Another issue that needs to be resolved concerns the difficulty in judging whether in-class CT interventions actually achieve their desired results (Settle et al., 2012). Again, researchers have pointed out the difficulty of assessing CT in classrooms, and have called for real-time assessment to provide learning progression data per student to support teachers' instruction (Bers et al., 2014; Wolz, Stone, Pearson, Pulimood, & Switzer, 2011). The framework we have proposed can be a guideline to develop assessment tasks that elicit evidence for specific CT skills. The examples we present of how CT is being measured in Scratch, Zoombinis, and FACT are intended to highlight these possibilities.

Studies on transfer of CT are also needed (Bers et al., 2014). One problem is how to identify the application of CT in other domains (Czerkawski & Lyman, 2015). Efforts have been made by a few researchers, such as Ioannidou et al. (2011), who tried to examine the transfer of CT acquired from games to math and science courses. Similarly, Repenning et al. (2015) aimed to investigate transferred CT skills from games to the creation of simulations of scientific phenomena. Grover et al. (2015) found that students could apply computational concepts learned from Scratch to specific text-based programming languages, which is quite promising. In any case, the long-term retention of CT skills, along with the application of CT skills to other contexts and domains, is under-researched.

A final area deserving research attention involves gender differences in the development of CT. Females are often underrepresented in STEM related subjects, particularly once they reach college. Researchers may consider utilizing CT to motivate learners, especially females, to major in science fields (e.g., Grover & Pea, 2013; Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012; Repenning et al., 2015). However, the results of studies examining gender differences are inconsistent. Atmatzidou and Demetriadis (2016) reported that girls' CT skills improved significantly after intervention, and that ultimately girls and boys reached the same skill level. Girls tend to spend significantly more time learning online after school, resulting in better performance than boys (Grover et al., 2015). However, no gender differences were reported by Werner et al. (2012) and Yadav et al. (2014).

Computational thinking needs to be demystified (e.g., Barr & Stephenson, 2011). Towards this end, we have developed a definition and shown examples of a model that: (a) considers CT as a logical way of thinking, not simply knowing a programming language; (b) particularly focuses on conceptual development required to engage in problem decomposition, abstraction, algorithmic design, debugging, iteration, and generalization; (c) examines performance-based competencies related to each of these facets of CT, and (d) can be strengthened and emphasized within existing (e.g. STEM) curricula. Our CT model presented in this paper is intended to be broadly applicable, while specific enough to inform measurement of CT learning—overall and at the facet level (for diagnostic purposes). Current research is in progress to validate this claim. We anticipate the next few years will bring many empirical studies to help refine this model for use across wide-ranging contexts.

Uncited reference

Angeli et al., 2016.

Unlisted reference

Jenkins, 2015

Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. Co-Author XXX gratefully acknowledges funding from the US National Science Foundation, and intellectual support from the EdGE at TERC team. This work is most closely related to grant NSF/DRL/DRK12/#0150228.

References

Allan, V., Barr, V., Brylow, D., Hambrusch, S., 2010, March. Computational thinking in high school courses. In: Proceedings of the 41st ACM technical symposium on computer science education. ACM, pp. 390–391.

Anderson, N.D., 2016. A call for computational thinking in undergraduate psychology. Psychology Learning & Teaching 15, 226–234. https://doi.org/10.1177/1475725716659252.

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. Educational Technology & Society 19 (3), 47–57.

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. https://doi.org/10.1016/j.robot.2015.10.008.

Bagiati, A., Evangelou, D., 2016. Practicing engineering while building with blocks: Identifying engineering thinking. European Early Childhood Education Research Journal 24 (1), 67–85. https://doi.org/10.1080/1350293X.2015.1120521.

Barr, D., Harrison, J., Conery, L., 2011. Computational thinking: A digital age skill for everyone. Learning & Leading with Technology 38 (6), 20–23.

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, 48–54. https://doi.org/10.1145/1929887.1929905.

Basawapatna, A., Koh, K.H., Repenning, A., Webb, D.C., Marshall, K.S., 2011, March. Recognizing computational thinking patterns. In: Proceedings of the 42nd ACM technical symposium on computer science education. ACM, pp. 245–250.

Basu, S., Biswas, G., Kinnebrew, J.S., 2017. Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. User Modeling and User-adapted Interaction 27 (1), 5–53.

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, 628–647. https://doi.org/10.1007/s10956-015-9552-x.

Bers, M., Flannery, L., Kazakoff, E., Sullivan, A., 2014. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. Computers & Education 72, 145–157. https://doi.org/10.1016/j.compedu.2013.10.02.

Brennan, K., Resnick, M., 2012, April. New frameworks for studying and assessing the development of computational thinking. In: Proceedings of the 2012 annual meeting of the American educational research association, (Vancouver, Canada).

Burke, Q., O'Byrne, W.I., Kafai, Y.B., 2016. Computational participation: Understanding coding as an extension of literacy instruction. Journal of Adolescent & Adult Literacy 59, 371–375. https://doi.org/10.1002/jaal.496.

Cetin, I., 2016. Preservice teachers' introduction to computing: Exploring utilization of scratch. Journal of Educational Computing Research https://doi.org/10.1177/0735633116642774, Advance online publication.

Cuny, J., Snyder, L., Wing, J.M., 2010. Demystifying computational thinking for non-computer scientists. Unpublished manuscript, referenced in http://www.cs.cmu.edu/∼CompThink/resources/TheLinkWing.pdf.

Czerkawski, B.C., Lyman III, E.W., 2015. Exploring issues about computational thinking in higher education. TechTrends 59 (2), 57–65. https://doi.org/10.1007/s11528-015-0840-3.

Denner, J., Werner, L., Campe, S., Ortiz, E., 2014. Pair programming: Under what conditions is it advantageous for middle school students?. Journal of Research on Technology in Education 46, 277–296. https://doi.org/10.1080/15391523.2014.888272.

Grover, S., 2011, April. Robotics and engineering for middle and high school students to develop computational thinking. In: Paper presented at the annual meeting of the American educational research association, New Orleans, LA.

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

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, 199–237. https://doi.org/10.1080/08993408.2015.1033142.

Harel, G., Sowder, L., 2005. Advanced mathematical-thinking at any age: Its nature and its development. Mathematical Thinking and Learning 7 (1), 27–50. https://doi.org/10.1207/s15327833mtl0701_3.

Ioannidou, A., Bennett, V., Repenning, A., Koh, K.H., Basawapatna, A., 2011, April. Computational thinking patterns. In: Paper presented at the annual meeting of the American educational research association, New Orleans, LA.

Israel, M., Pearson, J., Tapia, T., Wherfel, Q., Reese, G., 2015. Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. Computers & Education 82, 263–279. https://doi.org/10.1016/j.compedu.2014.11.022.

Jun, S., Han, S., Kim, H., Lee, W., 2014. Assessing the computational literacy of elementary students on a national level in Korea. Educational Assessment, Evaluation and Accountability 26, 319–332. https://doi.org/10.1007/s11092-013-9185-7.

Kazimoglu, C., Kiernan, M., Bacon, L., Mackinnon, L., 2012. A serious game for developing computational thinking and learning introductory computer programming. Procedia-social and Behavioral Sciences 47, 1991–1999.

Kim, B., Kim, T., Kim, J., 2013. Paper-and-pencil programming strategy toward computational thinking for non-majors: Design your solution. Journal of Educational Computing Research 49, 437–459. https://doi.org/10.2190/EC.49.4.b.

Kim, Y.C., Kwon, D.Y., Lee, W.G., 2014. Computational modeling and simulation for learning an automation concept in programming course. International Journal of Computer Theory and Engineering 6, 341–345. https://doi.org/10.7763/IJCTE.2014.V6.886.

Korkmaz, , Çakir, R., Özden, M.Y., 2017. A validity and reliability study of the computational thinking scales (CTS). Computers in Human Behavior https://doi.org/10.1016/j.chb.2017.01.005.

Lu, J.J., Fletcher, G.H., 2009, March. Thinking about computational thinking. ACM SIGCSE Bulletin 41 (1), 260–264.

Lye, S., Koh, J., 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior 41, 51–61. https://doi.org/10.1016/j.chb.2014.09.012.
Mishra, P., Yadav, A., 2013. Of art and algorithms: Rethinking technology & creativity in the 21st century. TechTrends 57 (3), 10–14.
National Research Council, 2010. Committee for the Workshops on Computational Thinking: Report of a workshop on the scope and nature of computational thinking. National Academies Press, Washington, D.C.
Papert, S., 1980. Mindstorms. Children, computers and powerful ideas. Basic books, New York.
Papert, S., 1991. Situating constructionism. In: Papert, S., Harel, I. (Eds.), Constructionism. MIT Press, Cambridge, MA.
Pawley, A., 2009. Universalized narratives: Patterns in how faculty members define "engineering". Journal of Engineering Education 98, 309–319.
Raven, J.C., 1981. Manual for Raven's progressive matrices and vocabulary scales. Research supplement No.1: The 1979 british standardisation of the standard progressive Matrices and mill hill vocabulary scales, together with comparative data from earlier studies in the UK, US, Canada, Germany and Ireland. Harcourt Assessment, San Antonio, TX.
Raven, J.C., 2000. The Raven's progressive matrices: Change and stability over culture and time. Cognitive Psychology 41, 1–48.
Razzouk, R., Shute, V., 2012. What is design thinking and why is it important?. Review of Educational Research 82, 330–348.
Repenning, A., Webb, D.C., Koh, K.H., Nickerson, H., Miller, S.B., Brand, C., … Repenning, N., 2015. Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. ACM Transactions on Computing Education (TOCE) 15 (2), 1–31. https://doi.org/10.1145/2700517.
Román-González, M., Pérez-González, J.C., Jiménez-Fernández, C., 2016. Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. Computers in Human Behavior
Rowe, E., Asbell-Clarke, J., Baker, R.S., 2015. Serious games analytics to measure implicit science learning. In: Loh, C.S., Sheng, Y., Ifenthaler, D. (Eds.), Serious games analytics: Methodologies for performance measurement, assessment, and improvement. Springer International Publishing, Switzerland, pp. 343–360.
Sanford, J.F., Naidu, J.T., 2016. Computational thinking concepts for grade school. Contemporary Issues in Education Research (CIER) 9 (1), 23–31. https://doi.org/10.19030/cier.v9i1.9547.
Seiter, L., Foreman, B., 2013, August. Modeling the learning progressions of computational thinking of primary grade students. In: Proceedings of the ninth annual international ACM conference on international computing education research. ACM, pp. 59–66.
Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G., Clark, D., 2013. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. Education and Information Technologies 18, 351–380. https://doi.org/10.1007/s10639-012-9240-x.
Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., et al., 2012, July. Infusing computational thinking into the middle-and high-school curriculum. In: Proceedings of the 17th ACM annual conference on innovation and technology in computer science education. ACM, pp. 22–27.
Shute, V.J., 1991. Who is likely to acquire programming skills?. Journal of Educational Computing Research 7 (1), 1–24.
Shute, V.J., Masduki, I., Donmez, O., 2010. Conceptual framework for modeling, assessing, and supporting competencies within game environments. Technology, Instruction, Cognition, and Learning 8, 137–161.
Sneider, C., Stephenson, C., Schafer, B., Flick, L., 2014. Computational thinking in high school science classrooms. The Science Teacher 81 (5), 53–59.
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. https://doi.org/10.1007/s10956-015-9581-5.
Werner, L., Denner, J., Campe, S., Kawamoto, D.C., 2012, February. The fairy performance assessment: Measuring computational thinking in middle school. In: Proceedings of the 43rd ACM technical symposium on computer science education. ACM, pp. 215–220.
Wing, J.M., 2006. Computational thinking. Communications of the ACM 49 (3), 33–35.
Wing, J.M., 2008. Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society a: Mathematical, Physical and Engineering Sciences 366, 3717–3725. https://doi.org/10.1098/rsta.2008.0118.
Wing, J.M., 2010. Computational thinking: What and why?. Unpublished manuscript Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, Retrieved from https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.
Wolz, U., Stone, M., Pearson, K., Pulimood, S.M., Switzer, M., 2011. Computational thinking and expository writing in the middle school. ACM Transactions on Computing Education (TOCE) 11 (2), 1–22.
Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., Korb, J.T., 2014. Computational thinking in elementary and secondary teacher education. ACM Transactions on Computing Education (TOCE) 14 (1), 1–16. https://doi.org/10.1145/2576872.
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, 562–590. https://doi.org/10.1177/0735633115608444.