TechTrends (2018) 62:574–584
https://doi.org/10.1007/s11528-018-0290-9

ORIGINAL PAPER

Computational What? Relating Computational Thinking to Teaching

Ugur Kale[1] · Mete Akcaoglu[2] · Theresa Cullen[3] · Debbie Goh[4] · Leah Devine[5] · Nathan Calvert[6] · Kara Grise[7]

Published online: 18 April 2018
© Association for Educational Communications & Technology 2018

Abstract
Computational thinking is one of the skills critical for successfully solving problems posed in a technology driven and complex society. The limited opportunities in school settings to help students develop computational thinking skills underscores the need for helping teachers integrate it in their practices. Besides developing the knowledge of technology, content, and pedagogy, teachers need to recognize the relevance of computational thinking to their teaching, a factor influencing their future practice with it. Drawing from the literature on problem-solving and TPACK framework, this paper discusses strategies, including content-specific examples, problem-solving nature of computational thinking, and the methods of teaching problem-solving for enabling teachers to make the connections between computational thinking and their practices.

Keywords Computational thinking · Problem solving, teacher education

Background

Computational thinking is a means to understand and solve complex problems through using computer science concepts and techniques (Wing 2008) such as decomposition, pattern recognition, abstraction, and algorithms (Grover and Pea 2013). As it may relate to individuals' abilities to use technology in everyday life, computational thinking becomes even more critical to be successful in society. The World Economic Forum predicts a loss of 7.1 million jobs by 2020 as robots and automation displace jobs across industries and geographic regions. At the same time, there will be 2.1 million new jobs created in the areas of computing, math, architecture, and engineering (The Fourth Industrial Revolution n.d.). The rapid change in skills requirements across all jobs underscores the urgency of developing computational thinking and code literacy in our children to future-proof them for an economy and society that run on complex computing technologies such as artificial intelligence, robotics, the Internet of Things, and analytics.

As noted above, problem solving is the essential skill underlying computational thinking, and it is perhaps the most important skill differentiating humans (van Merriënboer 2013). Given the problems we face increase in complexity, the new generation of learners are ever more required to have the abilities to solve complex and dynamic problems (Sonnleitner et al. 2014). This makes it essential that schools and teachers work to support teaching of such skills to young children (Greiff et al. 2014; Sonnleitner et al. 2014). Unfortunately, however, formal

✉ Ugur Kale
ugur.kale@mail.wvu.edu

Mete Akcaoglu
makcaoglu@georgiasouthern.edu

Theresa Cullen
tacullen@ou.edu

Debbie Goh
debbiegoh67@gmail.com

Leah Devine
ldevine@k12.wv.us

Nathan Calvert
ghostyroastytoasty@gmail.com

Kara Grise
kgrise13@gmail.com

[1] Learning Sciences and Human Development Department, West Virginia University, 504-F Allen Hall, PO Box 6122, Morgantown, WV 26505, USA

[2] Georgia Southern University, Statesboro, GA, USA

[3] University of Oklahoma, Norman, OK, USA

[4] California University of Pennsylvania, California, PA, USA

[5] West Virginia Virtual Schools, Charleston, WV, USA

[6] Hollywood Elementary School, Hollywood, MD, USA

[7] Mercyhurst Preparatory School, Erie, PA, USA

schooling is often far from teaching students the necessary skills to solve complex problems of real life (Authors 2016; Resnick 1987). This is also indicated by the results of recent findings from the Programme for the International Assessment of Adult Competencies (Rampey et al. 2016): adults in the United States performed lowest, in comparison to adults ages 16–65, in seventeen other nations in their ability to engage in problem-solving in technology rich environments.

Efforts to address the need for promoting problem-solving and computational thinking skills in today's complex computing society are also reflected in recent initiatives by the U.S. Government and Department of Education, such as Digital Promise, MakerEd, and "Nation of Makers" movement by the White House. For example, in 2016, President Obama launched the *Computer Science for All* initiative to fund computer science education in schools so that American students could formally learn computer science and acquire computational thinking skills. The National Science Foundation (NSF) is also investing $120 million over the next five years to promote computer science education in school curricula nationwide.

Such initiatives built on the earlier research and development projects, however, mostly targeted large and affluent school districts in big cities (Guzdial 2016) and are more unique than common in the current public education system. In schools, competing curriculum priorities and lack of funds still limit the opportunities for teachers to learn how to teach computer science concepts and computational thinking in their classrooms (Google Inc. and Gallup Inc. 2016). Time constraints and a need for teachers to focus on increasing their content knowledge is also likely to make them hesitant to learn a new field.

A consequence of the above factors is that students in classrooms will have limited opportunities to gain computational thinking skills. Limited participation in the fields of STEM and computing will even widen the existing inequalities in the transition into a complex computing-based society with women and those in developing or underdeveloped settings most at risks of falling behind. More opportunities are, thus, essential in schools to enable teachers and students to practice computational thinking. While the increasing number of computer science classes focusing on computer programming can be promising (Google Inc. and Gallup Inc. 2016), teachers need to know more than how to code in order to integrate computational thinking in their teaching. Teachers must be shown how computational thinking can enhance their teaching and their students' understanding of their content area.

Teaching Computational Thinking

Successful integration of a technological innovation in teaching requires a nuanced understanding of not only technology, but also the content (what to teach) and pedagogy (how to teach) (Mishra and Koehler 2006), which is referred to as Technological Pedagogical Content Knowledge (TPACK). Accordingly, teaching computational thinking should entail the knowledge of using computational thinking tools (technology), knowing which instructional strategies to use to teach computational thinking and the subject matter (pedagogy), and understanding of computational thinking and the subject matter (content). Findings from a survey study with 636 computer science teachers in Greece (Giannakos et al. 2015) indicated that even though teachers rated their content, pedagogical, and technological knowledge individually high, their understanding of technology in relation to teaching computer science concepts was rather limited. In other words, knowing technology and content did not translate to knowing how to use technology to teach the content.

Not being able to relate the affordances of technology to the teaching of specific content would be even more problematic when it comes to promoting computational thinking in other subjects because teachers would need not only to become knowledgeable of the subject they teach but also to understand computer science concepts. For instance, to help students develop computational thinking in a science course, teachers may engage their students in creating a simulation of an ecosystem. Besides the understanding of the ecosystem, they would need to develop an algorithm and define variables to represent the interactions among its inhabitants. To develop the simulations, they would also need to operationalize the algorithm in the programming environment.

While such various kinds of knowledge and understanding are essential to a well-developed TPACK for promoting computational thinking, teaching with technological innovations requires the belief that these innovations are beneficial for teaching and learning (Ertmer et al. 2012). Teachers need to recognize the educational value of computational thinking regarding how it connects to their own teaching. Defined as the degree to which a task is "useful or relevant for other tasks or aspects of an individual's life" (Hulleman et al. 2010, p.881), utility value influences the perception of meaningfulness that shows the individual how the content connects to its application. Value perceptions are good predictors of future success and engagement (Hulleman et al. 2010). As such, recognizing the utility value of computational thinking is essential to teachers' interest in future practices with it.

Existing initiatives on computational thinking emphasize coding skills in computer science classes. For instance, as part of the national CS4All efforts, computer science tends to be taught as a standalone subject (Herold 2017), as opposed to being integrated in other subjects (Google Inc. and Gallup Inc. 2016). A sole programming focus of such practices may not be perceived relevant to all teachers who teach various content areas. Without recognizing what computational thinking may mean in their classrooms regarding technology use, content learning, and pedagogical approaches, teachers may find it meaningless, and consequently become reluctant to teach it.

As observed in a recent research project surveying over three thousand primary school teachers in Italy, the teachers had an incomplete conception about computational thinking (Corradini et al. 2017).

The purpose of this paper is to discusses strategies for enabling teachers to make the connections between computational thinking and their teaching. Specifically, we argue that (a) reviewing content-specific examples of computational thinking tools, (b) recognizing the problem-solving nature of computational thinking processes and identifying the kinds of knowledge to support computational thinking, and (c) applying methods of teaching problem-solving to promote computational thinking can help teachers recognize the educational value of computational thinking and connect it to their own teaching. In the next sections, we will discuss each strategy for such connections depicted in Fig. 1.

Content-Specific Computational Thinking Tools

The content focus in teacher professional development has been observed to be one of the key factors enabling teachers to transfer their newly gained knowledge to classroom settings (Garet et al. 2001). However, professional development programs to support teachers' knowledge and skills of integrating technologies tend to focus on tools and pedagogy in isolation from content (Goh and Kale 2015). Further, using computational thinking in teaching other subjects besides programming has been underexplored (Grover and Pea 2013). Thus, an explicit connection of computational thinking technology to content areas of teaching is an essential step toward helping teachers recognize its relevance to their practices.

In 2016, the International Society for Technology in Education (ISTE) included computational thinking as one of its seven standards for students. These standards are accepted

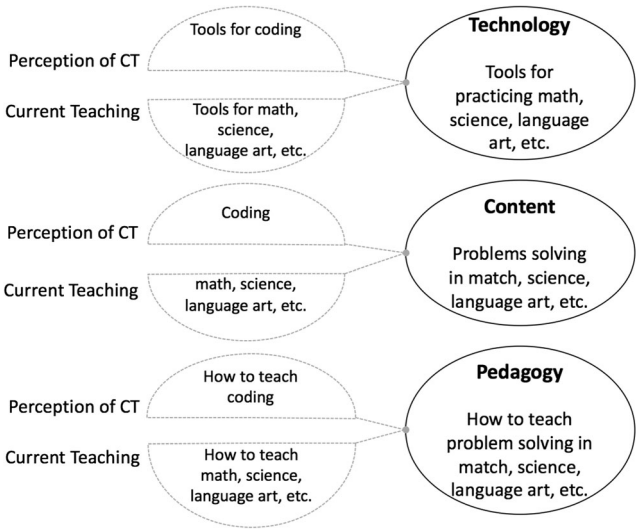

Fig. 1 Connecting CT with current practices

across the United States and reflect the growing inclusion of coding and computational thinking in schools (ISTE 2016). As a means to help promote computational thinking in K12 settings, ISTE and the Computer Science Teachers Association (CSTA) collaborated to identify ideas and examples of what computational thinking concepts and processes may look like in activities across various content areas. Barr and Stephenson (2011) outlined the results of these efforts, a simplified version of which is shown in Table 1.

While such classifications are useful to identify content-specific activities, technologies to foster computational thinking in the different subjects requires selection. Tools that are easy to use but also powerful enough to produce sophisticated algorithms are considered ideal such as Alice, Scratch, or Kodu, which help learners focus on designing through graphical programming while avoiding issues with coding syntax (Grover and Pea 2013). On the other hand, how such tools can benefit students' content learning beyond coding skills is still necessary.

In a semester-long recent workshop provided on computational thinking through Kodu, teacher candidates tended to focus on the coding aspects while having difficulties in connecting computational thinking to the content area that they teach (Authors 2016). In another professional development program on computational thinking through Scratch, only less than half of the participant teachers highlighted the content learning they helped facilitate in their classrooms (Duncan et al. 2017). Developing projects samples or prompting teachers to search ones specific to their content areas can be a useful strategy to avoid such issues. For instance, the ScratchED website (Scratch n.d.-a, -b) – an online community of parents, teachers, and students using Scratch, hosts sample projects, lessons, and relevant materials, which can be searched according to the grade levels, material types, and content areas.

Besides finding such resources to get lesson ideas, the ability to remix existing projects or to modify their scripts allows for making versions that can be tailored to specific lessons. According to Scratch user statistics (Scratch, n.d.-a, -b), in the first two months of 2017, around 500,000 projects were remixed and over 1,000,000 new projects were created, which is a promising trend for exchanging and creating lessons in various content areas that incorporate the use of computational thinking tools.

Problem-Solving Nature of Computational Thinking

Another key component of helping teachers connect computational thinking to their teaching practices is to help them recognize its problem-solving nature. Computational thinking is more than just computer programming: it involves both constructing and analyzing processes (Wing 2008). Through creating and presenting solutions to given situations, it engages students consciously in making artifacts (e.g., design

Table 1 * Examples of computational thinking processes in various content areas

CT processes	Mathematics	Science	Language arts
Decomposition	Apply order of operations in an expression	Do a species classification	Write an outline
Pattern recognition	Use histogram, pie chart, bar chart to find patterns	Identify patterns in experiment data	Represent patterns of different sentences
Abstraction	Identify underlying structures in a word problem	Build a model of a physical entity	Write a story with branches
Algorithm	Do long division and factoring	Do an experimental procedure	Write instructions
Automation	Use excel or star logo	Use probeware	Use a spell checker

* See Barr and Stephenson's work (2011) for other domains and CT concepts

systems) in situated environments. As a computing process, computational thinking starts with being confronted by problems, the solution to which involves decomposition, pattern recognition, abstraction, automation, algorithms, and analysis (Barr et al. 2011; Grover and Pea 2013; Wing 2008). These processes mirror the components of problem-solving, namely understanding and representing, planning and monitoring, executing, self-regulation (Mayer and Wittrock 2006; OECD 2003). Table 2 compares the processes involved in computational thinking and problem-solving.

By recognizing the problem-solving nature of computational thinking (Voogt et al. 2015), teachers can focus on problematizing their content for students to "solve", and determine the problem-solving processes needed to facilitate their computational thinking. Given the similarities between both processes, the kinds of knowledge required and the methods to facilitate computational thinking should benefit from those for teaching problem-solving. In other words, the knowledge base on how to teach problem-solving can inform the pedagogy for how to teach computational thinking.

Knowledge for Problem-Solving to Support Computational Thinking

Based on research on problem-solving, Mayer and Wittrock (2006) identified the kinds of knowledge to support the cognitive processes required for solving problems. These included (a) Factual/Conceptual, (b) Strategic, (c) Procedural, and (d) Metacognitive knowledge.

Factual knowledge refers to knowledge of facts such as "there are twelve months in a year", and conceptual knowledge involves knowledge of concepts, their classifications, and relations such as knowing how raindrops form (Mayer and Wittrock 2006). Facts and concepts provide the knowledge needed to *understand* and *represent* a given problem. Strategic knowledge is the knowledge of general methods regarding how to approach or decompose a problem such as synthesizing articles to identify the key ideas (Mayer and Wittrock 2006). Strategic knowledge is essential to *planning* and *monitoring* solutions to a given problem. Procedural knowledge is about knowing a set of particular procedures for how to do or complete a task, such as the steps to follow in order to change a tire or do long division (Mayer and Wittrock 2006). Procedural knowledge helps carry out the planned solutions (*Execution*) in the problem-solving process. Metacognitive knowledge, on the other hand, refers to the awareness of one's own thinking process such as "I don't know how to move to the next step." *Self-regulation* in the problem-solving process mostly depends on metacognitive knowledge (Mayer and Wittrock 2006). Given the problem-solving nature of computational thinking as depicted in Table 2, these kinds of knowledge may also be needed to support the processes involved in computational thinking (See Fig. 2).

Table 2 Comparison of problem solving and computational thinking

Problem-solving		Computational thinking	
Understand a situation, information, main features/mechanism	Understand & Represent	Confrontation	Define and understand a problem encountered
Identify constraints, parts, variables, relations	Plan & Monitor	Decomposition	Logically break it down into smaller parts
		Pattern recognition	Identify patterns among the smaller parts
Represent alternatives, relations, functions,		Abstraction	Filter out details and identify underlying characteristics or rules
Propose			
Carry out planned operations	Execute	Algorithm/ Automation	Generate and automate steps to solve the problem
Check & reflect on the decision, analysis & design, diagnosis/solution	Self-regulate	Analysis	Analyze possible solutions Test/debug/troubleshoot
			Analyze the appropriateness of the abstractions made

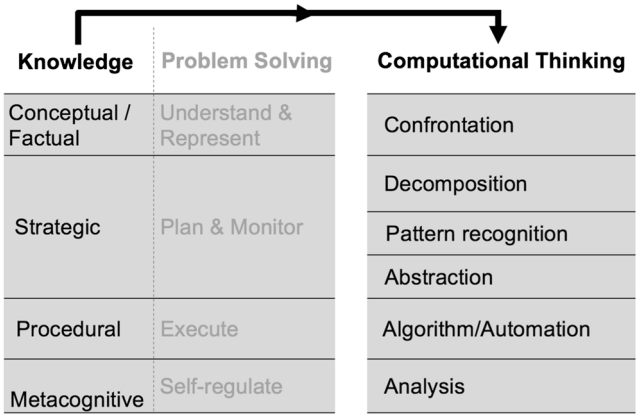

Fig. 2 Kinds of knowledge supporting problem solving and computational thinking

Teaching Problem-Solving to Facilitate Computational Thinking

Mayer and Wittrock (1996, 2006) outlined instructional methods that develop the kinds of knowledge that facilitate problem-solving processes, which can also benefit computational thinking (Akcaoglu 2014). These include Load-reducing, Schema-activation, Structured-based, Generative, Guided-discovery, Modeling, and Teaching Thinking Skills. Below, we described these strategies in details and provided examples for each to demonstrate the connections made between teaching problem solving and students' computational thinking. The examples came from the classrooms of three of the authors, who are currently K12 teachers. They specifically developed and implemented learning activities in various content areas including Spanish, geometry, and financial literacy as part of a graduate level course focusing on Scratch as a learning tool and problem solving as a teaching method for promoting computational thinking.

Load-Reducing *Load-reducing* methods focus on preventing working memory from being overloaded to facilitate information processing during problem-solving. These methods involve both automaticity and constraint removal strategies to help develop the procedural knowledge (Mayer and Wittrock 2006). Automaticity encompasses the teaching of specific low-level component cognitive skills pertaining to a task so that in ensuing tasks, more effort can be placed upon executing higher-level problem-solving skills (Samuels 1979). For instance, students who practice the basics of coding skills enough would then need to pay little attention to it and focus more on the planning and execution of their solutions during programming. In constraint removal strategy, the goal is to remove constraints in the problem space to ease the process of new knowledge acquisition during the process of problem-solving. The goal is to "create problem skills tasks that do not require attention demanding tasks" (Mayer and Wittrock 2006, p.292). By using visual coding programs, students would not need to worry about syntaxes in text-based programing.

Example in Spanish Designed for upper middle school students (7th–8th graders), this activity is contextualized within reading and writing in Spanish. The main learning outcome for students is to read and understand basic Spanish words, and to create interactive learning games in Spanish by using Scratch. The activity starts with reviewing Spanish commands in Scratch. Then, students play an interactive game that requires written input in Spanish to "travel" to Spanish speaking countries and answer trivia questions to earn points, which is followed by the creation of students' own interactive games.

Remixing can serve as a *load-reducing* method in this activity helping the *algorithm* process of computational thinking. For instance, instead of creating a new Scratch project, students remix the example interactive game created by the teacher. To remix, they modify the existing codes in the algorithm and observe the changes made as the game runs (See Fig. 3). This process allows students to analyze and understand the structure of the existing game, through which they can see if the underlying model used fits with their own goals in creating a new one.

Scheme-Activation *Schema-activation* methods focus on enabling learners to make sense of the new content by relating it to their existing understanding (Mayer and Wittrock 2006), and support the conceptual/factual knowledge. For instance, advance organizers, diagrams, or graphic organizers (e.g., concept maps) can highlight the relationship among both the familiar and unfamiliar concepts key to understanding and representing a problem confronted (Jonassen 2004). Through a chart that outlines the behavior of a system (e.g., a simulation or game), learners can identify the main elements (decompose), their interactions, and conditions in which they occur (pattern). Using analogies is another means to activate schema, in which "learners solve a new problem by using what they know about a related problem that they can [already] solve" (Mayer and Wittrock 1996, p.55). Using analogous problems, students can find similarities in the underlying structures in two separate problems that differ on the surface (e.g., a predator-prey problem using rabbits and wolves vs. using two imaginary alien species) (Gick and Holyoak 1980, 1983). During design tasks, analogies can be provided through introducing scenarios that are different on the surface, but essentially built on same underlying principles.

Example in Geometry Designed for 5th graders, this activity focuses on two-dimensional shapes and their angles. The main learning outcome for students is to graph two dimensional composite shapes on the coordinate plane by identifying the number of lines and determining the angles of the basic

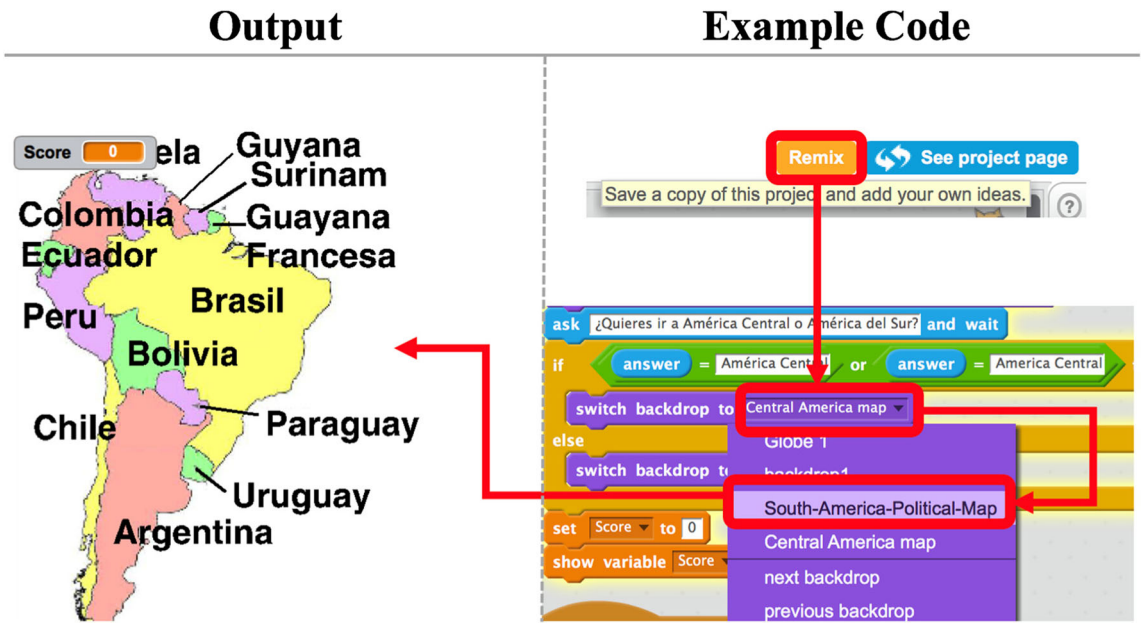

Fig. 3 Process of remixing an existing project

shapes. The activity starts with plotting coordinates on a paper grid to create basic shapes such as a square. This is followed by a task to draw the same shape in Scratch, which requires familiarization with its interface and identifying the necessary commands such as "pen down" (to start drawing), and "go to x:_ y:_" (to start the drawing from specific coordinates), "move:_ steps" (to determine the distance to move forward) "turn: _ degrees" (to change the direction of the movement), and "repeat" (to automate the steps according to the number of lines required).

Drawing shapes that students familiar with can activate their schema, which would help make sense of the elements of the problem *confronted*. For instance, the problem posed in this activity is to draw a composite shape in the Scratch interface such as a drawing of a simple house. A composite shape consists of multiple basic shapes, which students may be familiar with drawing on paper. The major task becomes drawing the similar shapes in Scratch interface through the necessary commands (See Fig. 4).

Generative In *generative* methods, as the name suggests, students are asked to generate (e.g., write out) connections between what they know and what they learned. This process is believed to promote teaching for understanding (Mayer and

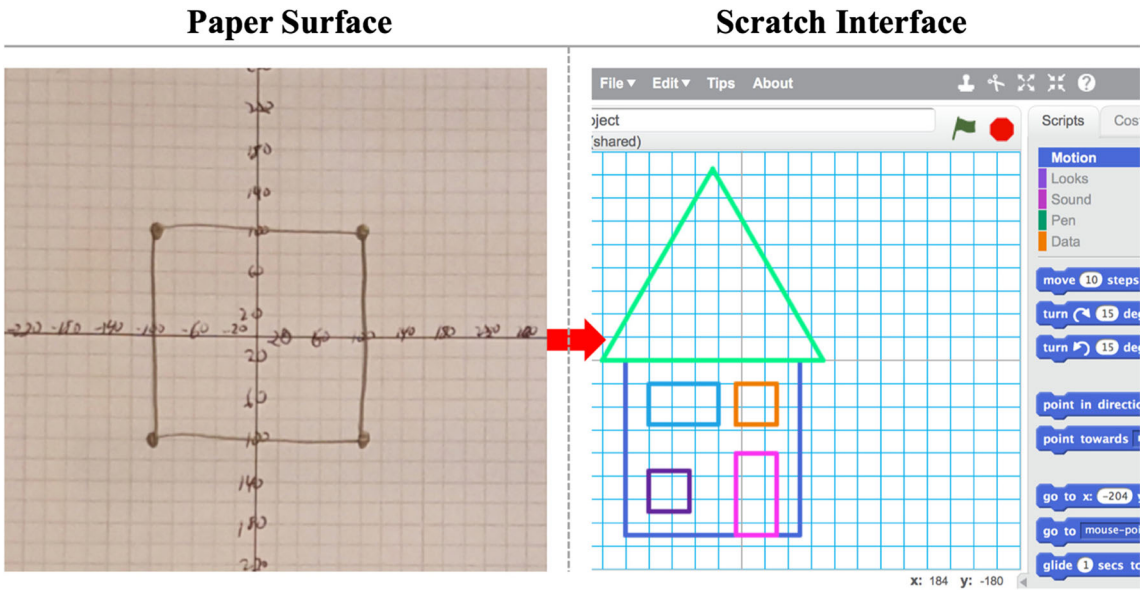

Fig. 4 Practice drawing on paper before scratch interface

Wittrock 2006). Although this sounds similar to schema activation, the focus here is rather on the learners generating the connections as opposed to being presented ones. The generative methods support conceptual/factual knowledge by requiring learners to explain the new information and how it is related to what they already know (Linden and Wittrock 1981). For instance, when asked to describe the meaning of a variable in familiar situations, learners may liken it to a physical "container". These methods also support metacognitive knowledge by engaging learners in a self-explanation process (Wittrock 1989, 1991). For instance, prompting learners to explain or take notes for themselves regarding the set of coding they develop for a given task can enable them to analyze their own problem-solving processes.

Example in Financial Literacy The activity, part of an 11th grade financial planning course, focuses on stock price factors. The main learning outcome for students is to define and calculate stock price factors such as current yield, total return, and earnings per share, and to analyze these calculations to determine whether to purchase, sell, or hold the stocks. After a presentation about stock prices, students practice calculating simple problems on a worksheet that has question prompts. They continue practicing the calculations with more questions on Scratch where they have to define variables for the formulas and develop algorithms for the calculations of given scenarios.

Question-prompts can serve as a generative method in this activity to facilitate the *decomposition* process of computational thinking by having students make the connections between what they know and what they learned. For instance, students are given a worksheet that not only lists the problem but also provides questions that prompt them to explain and break down the problem (e.g., formulas) into smaller components such as the variables (e.g., annual dividend and current market value) to calculate the current yield (See Fig. 5).

Structured Methods *Structured* methods, supporting strategic knowledge, focuses on enabling learners to manipulate concrete objects for better understanding of more abstract concepts or rules (Mayer and Wittrock 2006), such as using bundles of sticks or beads to teach mathematics concepts. Design tasks through coding can serve as external representations of problems, which can serve as "concrete" objects helping students understand the abstract rules behind complex problems (White 1993).

Example in Geometry As described earlier, for this activity, 5th graders are to graph two dimensional composite shapes on the coordinate plane in Scratch interface. They identify the number of lines and determine the angles of the basic shapes. Manipulating the content in this activity can help them *identify patterns*. For instance, students can draw a triangle first, then a square, a pentagon, or the like. As they draw a new shape or modify a previously drawn one, they make changes in their coding, which would reflect the relationship between the number of lines and the angles for each kind of shape (e.g., turning 90 degrees to draw a square, or 60 degrees for a triangle). Based on the relationship and pattern identified, an algorithm can draw a kind of shape based on the number of lines entered as the input (See Fig. 6).

Guided-Discovery *Guided-discovery* methods, supporting strategic knowledge, emphasize a combination of enough guidance to and a level of freedom for learners to explore the problem so that they can identify relations and patterns, and to discover the underlying rules and principles on the problem focused (Mayer and Wittrock 2006). Guided discovery methods are known to be superior to pure discovery methods (for a review see: Kirschner et al. 2006; Mayer 2004). For example, students can be provided with practice opportunities on their own where they are also guided as needed to discover the reasons for the observed outcomes as they code during a problem-solving task.

Example in Geometry The previous example focused on structured methods where 5th graders identify the number of lines and determine the angles of the basic shapes by manipulating the shapes and changing the codes. Despite its positive impact on understanding of abstract concepts, manipulating concrete

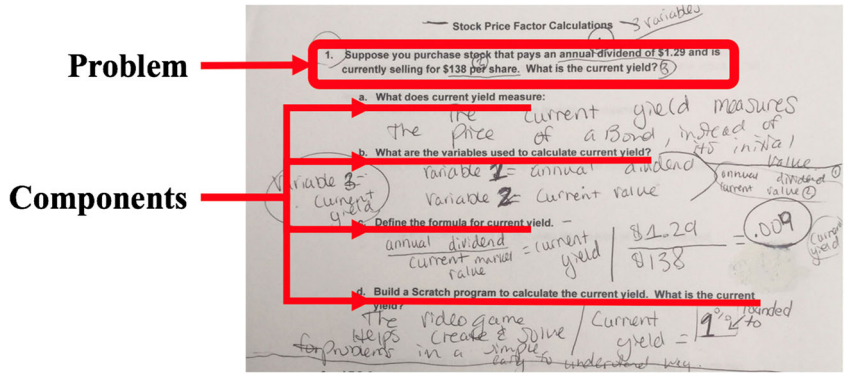

Fig. 5 Student worksheet with question prompts breaking problem into components

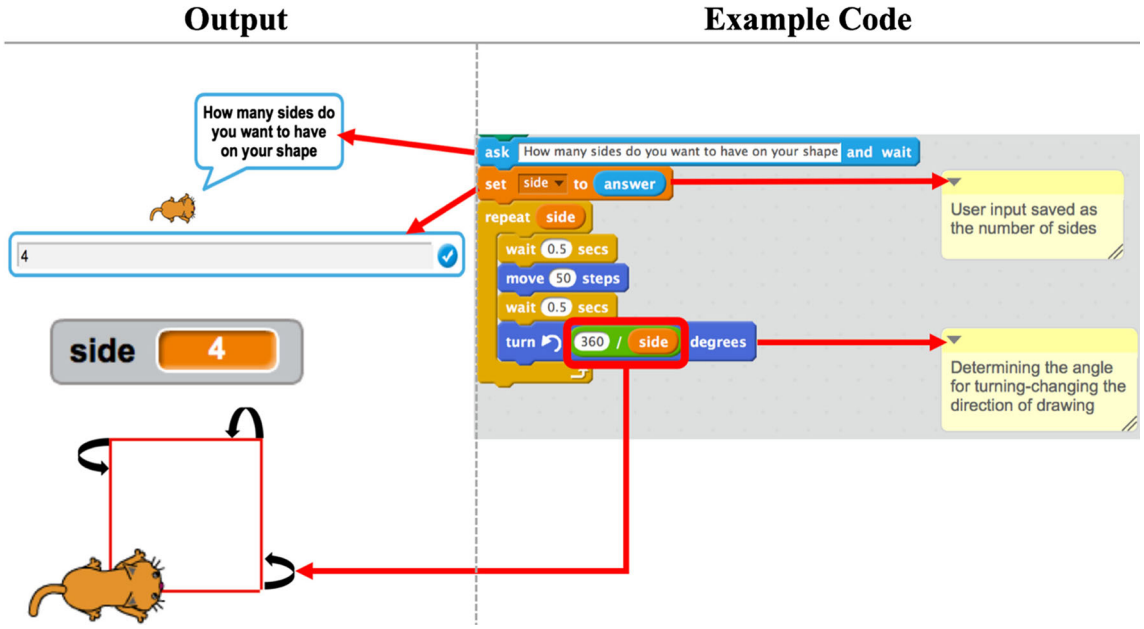

Fig. 6 Example code and output for drawing various shapes in Scratch

objects may overload learning (Kilpatrick et al. 2001), especially if it is completed in a pure discovery method. In the activity, just because students draw different shapes and modify their coding on their own does not mean they would automatically grasp the relationship between the elements of each shape and the codes. Guidance, however, in the form of question-prompts for students to focus on each shape at a time, and to explain the changes they notice in relation to the number of lines and angles for the new shape can help them *identify patterns* that they might not be able to recognize on their own.

Modeling *Modeling* methods emphasize demonstrating or explaining the steps involved in solving problems (Mayer and Wittrock 2006), and can support conceptual, strategic, and procedural knowledge. Worked-out examples and apprenticeships are common strategies to model. In work-out examples, explaining the steps and their justifications can provide the learners with a successful problem solver's reasoning (Quilici and Mayer 2002). This can aid learners' three key cognitive processing in problem-solving (Mayer and Wittrock 2006) – relating a newly encountered problem to the one exemplified (conceptual), abstracting a method (strategic), and applying it to solving the new problem (procedural). For instance, a completed program that has annotations explaining the key codes and their rationales can support learners' such cognitive processes. The apprenticeship strategy also highlights examples. However, the main focus is on novices working with more experienced problem solvers (Lave and Wenger 1991). Reciprocal teaching and cooperative learning are also other strategies to model (Palinscar and Brown 1984). In reciprocal teaching, instructors and learners can take turns to discuss their approaches to a given coding task, and during cooperative learning, learners with varying abilities work together on similar tasks where they can share their varying perspectives to solving the problem.

Example in Spanish For this activity, as outlined earlier, 7th and 8th graders are to read and understand basic Spanish words, and to create interactive learning games in Spanish by using Scratch where they create interactive games. Demonstrations and resources in this activity can serve as a method to model the steps that students can *abstract* from and apply to their learning tasks. For instance, regarding developing interactive games, the teacher models to students how to work with the necessary codes, change backgrounds, or add characters, which can be in the form of in-class demonstrations or video tutorials that can be accessed at any time (See Fig. 7):

As students review such resources or observe their teachers' demonstrations in class, they can start to see the underlying functions of the codes such as "if and then conditions", which then can be used in any scenarios (abstracting) when it comes to designing their own interactive games.

Teaching Thinking Finally, *Teaching Thinking Skills Directly* is the process of designing instruction with the goal of explicitly teaching specific metacognition to students (Mayer and Wittrock 1996, 2006). Having learners focus on and be conscious about the problem-solving process (e.g., through reflecting) through reflections is the key tenet of teaching thinking skills. Jonassen (2004) summarized key strategies that support students' reflection on the problem-solving process, such as peer instruction, think-aloud pairs, and coding protocols. These are not domain specific strategies and can be applied to teaching computational thinking. For example, in

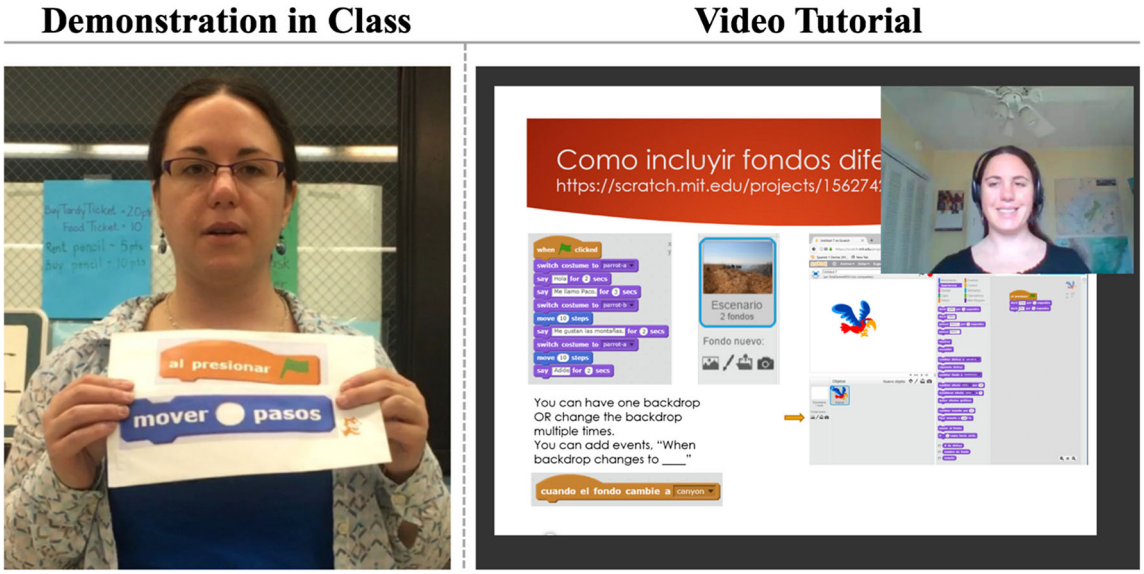

Fig. 7 Teacher modeling the use of codes in scratch in two modes

peer instruction, learners can be prompted to not only describe their coding for a given task but also convince another learner of the correctness of their solution. For the think-aloud pair strategy, learners are assigned two roles – problem solver and listener. The problem solver can be asked to verbalize her thoughts as she generates the codes while the listener listens to it, catches mistakes, and prompts the problem solver to verbalize further as needed. Coding protocols, another strategy to teaching thinking skills, help learners learn how to solve problems by having them study how others solved them. Providing learners with a transcript of how a program is completed to generate certain outcomes can allow them to analyze the successful problem solvers' thought process.

Example in Financial Literacy As described earlier, for this activity, 11th graders are to practice the calculations of stock price factors such as current yield, total return, and earnings per share. They define variables for the formulas and develop algorithms for the calculations of given scenarios in Scratch.

Having students be aware of the problem-solving process through reflection can help them *analyze* their decisions. In the activity, students are taught to develop and run the algorithms that calculate the associated prices such as the Total Return for different scenarios. While working in pairs, they can be asked to justify their decisions to each other regarding buying, selling, or holding a particular stock based on the analysis of their codes. For instance, a stock with a higher dividend may not necessarily result in a higher Total Return depending on the values in other variables taken into account for the calculations.

Figure 8 summarizes the lists of methods discussed so far for teaching problem-solving, and the computational thinking process that they can facilitate. The specific methods and the computational thinking processes that

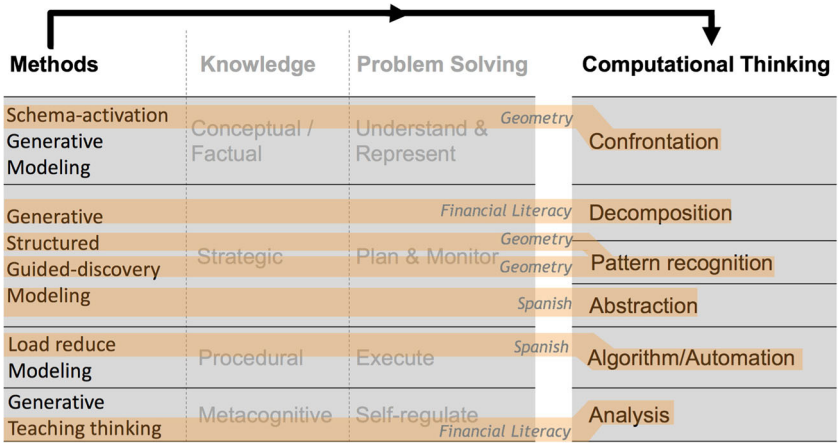

Fig. 8 Methods for teaching problem solving and facilitating computational thinking

were connected in the examples above were also highlighted with references to the examples.

Conclusion

Referred to as individuals' abilities to solve problems through using computer science concepts, computational thinking is considered a critical skill in contemporary and future society (Wing 2008). However, there are limited opportunities in public education for students to develop computational thinking skills (Google Inc. and Gallup Inc. 2016; Guzdial 2016), which will be likely to widen existing inequalities in the transition into a complex computing society. While more opportunities are needed in school settings focusing on computational thinking, teachers need to know more than how to code in order to facilitate their students' learning. Teachers need to recognize the relevance of computational thinking to what they teach before they can intend to incorporate it their practices.

In this paper, we highlighted three strategies that can help teachers make the connections between computational thinking and their teaching. The first one emphasized content-specific examples regarding the use of computational thinking tools. Given teachers' tendencies to focus on coding aspects of computational thinking (Authors 2016), seeing how technologies and resources can benefit students' content learning is key to their perception of computational thinking. The second strategy was based on problem-solving nature of computational thinking. By recognizing the similarities between both kinds of learning processes, teachers can frame their content as "problems" for their students to "solve", while promoting computational thinking. The third strategy building on the first two ones was the main theme of the paper. It is focused on the methods of teaching problem-solving as a means to support the kind of knowledge necessary to the processes involved in computational thinking.

By describing the methods of teaching problem-solving and providing subject-specific examples, we operationalized the teaching of problem-solving in the context of teaching computational thinking, which can serve as a practical framework to guide the efforts in promoting computational thinking in content-specific learning activities. Our approach here reflected a main argument that the methods to teach problem solving should benefit the teaching of computational thinking due to similarities between the two processes. However, future research is needed to examine the effectiveness of these methods especially in emerging technology-rich tinkering context such as makerspaces where teachers and students design and create artifacts by inquiring, creating, and solving problems. Exploring how these methods can inform (and be informed by) the inquiry and making processes to facilitate computational thinking would be an essential area of future studies.

Compliance with Ethical Standards

Conflict of Interest The authors declare that they have no conflict of interest.

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research & Development., 62*(5), 583–600. https://doi.org/10.1007/s11423-014-9347-4.

Authors et al. (2016). Details removed for peer review.

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.

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

Corradini, I., Lodi, M., & Nardelli, E. (2017). Conceptions and Misconceptions about Computational Thinking among Italian Primary School Teachers. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 136–144). ACM.

Duncan, C., Bell, T., & Atlas, J. (2017). What do the teachers think?: Introducing computational thinking in the primary school curriculum. In *Proceedings of the Nineteenth Australasian Computing Education Conference* (pp. 65–74). ACM.

Ertmer, P. A., Ottenbreit-Leftwich, A. T., Sadik, O., Sendurur, E., & Sendurur, P. (2012). Teacher beliefs and technology integration practices: A critical relationship. *Computers and Education, 59*(2), 423–435. https://doi.org/10.1016/j.compedu.2012.02.001.

Garet, M. S., Porter, A. C., Desimone, L., Birman, B. F., & Kwang Suk, Y. (2001). What makes professional development effective? Results from a national sample of teachers. *American Educational Research Journal, 38*(4), 915–945.

Giannakos, M. N., Doukakis, S., Pappas, I. O., et al. (2015). Investigating teachers' confidence on technological pedagogical and content knowledge: An initial validation of TPACK scales in K-12 computing education context. *Journal of Computers in Education., 2*(1), 43–59. https://doi.org/10.1007/s40692-014-0024-8.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology, 12*, 306–355.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology, 15*, 1–38.

Goh, D., & Kale, U. (2015). The urban-rural gap: Project-based learning with technology among west Virginian teachers. *Technology Pedagogy, and Education.* https://doi.org/10.1080/1475939X.2015.1051490.

Google Inc., & Gallup Inc. (2016). Trends in the state of computer science in U.S. K-12 schools. In *Retrieved from* http://goo.gl/j291E0.

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.

Greiff, S., Wüu, S., WuS, S., Csapó, B., Demetriou, A., Hautamäki, J., Graesser, A. C., & Martin, R. (2014). Domain-general problem solving skills and education in the 21st century. *Educational Research Review, 13*, 74–83.

Guzdial, M. (2016). State of the states: Progress toward CS for all. Communications of the ACM. Retrieved from http://cacm.acm.

org/blogs/blog-cacm/198790-state-of-the-states-progress-toward-cs-for-all/fulltext

Herold, B. (2017). Computer science for all in San Francisco schools: 7 Early takeaways [Blog post]. Retrieved from http://blogs.edweek.org/edweek/DigitalEducation/2017/04/computer_science_for_all_san_francisco_7_takeaways.html?cmp=SOC-SHR-twitter

Hulleman, C. S., Godes, O., Hendricks, B. L., & Harackiewicz, J. M. (2010). Enhancing interest and performance with a utility value intervention. *Journal of Educational Psychology, 102*(4), 880–895. https://doi.org/10.1037/a0019506.

International Society for Technology in Education (ISTE). (2016). ISTE national educational technology standards (NETS) for students. Eugene, OR :International Society for Technology in Education.

Jonassen, D. H. (2004). Learning to solve problems: An instructional design guide (Vol. 6). San Francisco, CA: Wiley.

Kilpatrick, J., Swafford, J., & Findell, B. (2001). *Adding it up: Helping children learn mathematics*. Washington, DC: National Academy Press.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist, 41*(2), 75–86.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge university press.

Linden, M., & Wittrock, M. C. (1981). The teaching of reading comprehension according to the model of generative learning. *Reading Research Quarterly*, 44–57.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction. *The American Psychologist, 59*(1), 14–19. https://doi.org/10.1037/0003-066X.59.1.14.

Mayer, R. E., & Wittrock, M. C. (1996). Problem-solving transfer. In D. C. Berliner & R. C. Calfee (Eds.), *Handbook of educational psychology* (pp. 47–62). New York, NY: Macmillan Library Reference.

Mayer, R. E., & Wittrock, M. C. (2006). Problem solving. In P. A. Alexander, P. H. Winne, P. A. Alexander, P. H. Winne (Eds.), Handbook of educational psychology (pp. 287–303). Mahwah, NJ, US: Lawrence Erlbaum Associates Publishers.

van Merriënboer, J. J. G. (2013). Perspectives on problem solving and instruction. *Computers & Education, 64*, 153–160. https://doi.org/10.1016/j.compedu.2012.11.025.

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A new framework for teacher knowledge. *Teachers College Record, 108*, 1017–1054.

OECD. (2003). PISA 2003 assessment framework: Mathematics, reading, science and problem solving knowledge and skills. *OECD Publishing*. Retrieved from http://www.oecd.org/edu/school/programmeforinternationalstudentassessmentpisa/33694881.pdf

Palinscar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction, 1*(2), 117–175.

Quilici, J. L., & Mayer, R. E. (2002). Teaching students to recognize structural similarities between statistics word problems. *Applied Cognitive Psychology, 16*(3), 325–342.

Rampey, B.D., Finnegan, R., Goodman, M., Mohadjer, L., Krenzke, T., Hogan, J., and Provasnik, S. (2016). Skills of U.S. unemployed, young, and older adults in sharper focus: Results from the program for the international assessment of adult competencies (PIAAC) 2012/2014: First look (NCES 2016-039). *U.S. Department of Education. Washington, DC: National Center for Education* Statistics Retrieved from http://nces.ed.gov/pubsearch

Resnick, L. B. (1987). The 1987 presidential address: Learning in school and out. *Educational Researcher, 16*(9), 13–20.

Samuels, S. J. (1979). The method of repeated readings. *The Reading Teacher, 32*(4), 403–408.

Scratch. (n.d.-a). Monthly project shares. Retrieved March 28, 2017, from https://scratch.mit.edu/statistics/

Scratch ED. (n.d.-b). *Search resources.* Retrieved March 28, 2017, from http://scratched.gse.harvard.edu/resources

Sonnleitner, P., Brunner, M., Keller, U., & Martin, R. (2014). Differential relations between facets of complex problem solving and students' immigration background. *Journal of Educational Psychology, 106*(3), 681–695. https://doi.org/10.1037/a0035506.

The Fourth Industrial Revolution: What it means and how to respond. (n.d.). Retrieved from: http://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond .

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. https://doi.org/10.1007/s10639-015-9412-6.

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A, 366*, 3717–3725. https://doi.org/10.1098/rsta.2008.0118.

White, B. Y. (1993). ThinkerTools: Causal models, conceptual change, and science education. *Cognition and Instruction, 10*(1), 1–100.

Wittrock, M. C. (1989). Generative processes of comprehension. *Educational Psychologist, 24*(4), 345–376.

Wittrock, M. C. (1991). Generative teaching of comprehension. *The Elementary School Journal, 92*(2), 169–184.