Technology, Knowledge and Learning
https://doi.org/10.1007/s10758-019-09423-8

ORIGINAL RESEARCH

An Emerging Technology Report on Computational Toys in Early Childhood

Megan Hamilton[1] · **Jody Clarke-Midura**[1] · **Jessica F. Shumway**[2] · **Victor R. Lee**[3]

Abstract
This emerging technology report describes computational toys as tools for learning and building computational thinking (CT) skills in young children. We present both a framework to categorize computational toys as well as a separate framework to evaluate the toys' effectiveness for teaching CT skills. We then apply our frameworks to thirty computational toys targeting children ages 5 and 6 years old. By identifying physical and ideational features of computational toys, educators and researchers will be able to apply our findings to practice and expand upon CT pedagogical research in young learners. Our future research goals include to investigate how CT skills can be observed and measured in early childhood education.

Keywords Computational thinking · Problem solving · Programming · STEM toys · Early childhood

1 Introduction

An influx of commercial STEM education toys, like Cubetto, are being marketed as tools to teach computer programming and problem-solving skills. These computational toys offer a variety of programming interfaces. For example, some use block-based programming, some utilize pictorial-based programming, and others are comprised of physical building blocks that provide a more tactile experience for the learner. Manufacturers promote computational toys with claims that children will improve their thinking and coding skills by interacting with them. For instance, Primo Toys advertises Cubetto (Fig. 1a, b), a wooden Montessori-based robot, as a learn-to-code toy that is "tangible, fun and age-appropriate," and as children interact with the toy they are able to learn the "fundamentals of coding including algorithms, functions, and debugging" ("Cubetto Universe," n. d.). Yet, there is little evidence to support these assertions. Oftentimes,

✉ Megan Hamilton
megan.hamilton@usu.edu

[1] Instructional Technology and Learning Sciences, Utah State University, Logan, UT, USA

[2] School of Teacher Education and Leadership, Utah State University, Logan, UT, USA

[3] Graduate School of Education, Stanford University, Stanford, CA 94305, USA

Published online: 31 August 2019

Fig. 1 **a** and **b** Sarah and Ryan making a connection between their code instructions (plastic pieces on the panel board) to the position of their robot and the distance to the goal on the map

educational technologies are developed so quickly that they bypass the development of research-informed pedagogical practices associated with them (Manches and Plowman 2017). There is a need for research to unpack how children interact with computational toys, what skills children are improving, and what, if any, impact these interactions have on childhood development and learning. Among the first steps toward these research goals, it is necessary to better understand the design elements and features of these computational toys.

In this emerging technology report, we discuss our early research on categorizing computational toys and present a framework to summarize the features of the toys for teaching computational thinking (CT) skills, or the "range of analytical mental tools that are inherent to the field of computer science" such as abstraction and heuristic reasoning (Bers 2018, p. 3). In the following sections, we provide context for our work, introduce a framework for categorizing computational toys, and provide findings from an analysis of thirty different toys commonly found in the commercial marketplace.

2 Background

Ideas involving CT date as far back as the 1950s (Tedre and Denning 2016). Papert (1980) was among the first to describe CT in his work related to the Logo programming language and the Logo turtle, an educational robot. In the early 2000s, CT was revitalized by Jeannette Wing as she refined its definition and urged for CT to become a part of every child's abilities (Wing 2006, 2008). Wing (2006) defined CT as the systematic thought process learners utilize as they are "solving problems, designing systems, and understanding human behavior by drawing on the concepts fundamental to computer science" (Wing 2006, p. 33). Prior to Wing (2006), early contributions from scholars strengthened the early foundations of CT by discussing how computer programming can enhance problem-solving abilities (Clements and Gullo 1984; DiSessa 2001; Harel and Papert 1990). In the following sections, we further contextualize CT by (1) discussing CT and its implications in K-12 education, (2) examining definitions of CT that have informed our work, and (3) exploring various tools used to teach CT in informal and formal learning environments.

2.1 CT in K-12 Education

National Research Council (NRC 2010) conducted workshops on CT and subsequently released an NRC report on cognitive and educational implications of CT, including applications of CT across disciplines and the relationship of CT to mathematics and engineering (NRC 2010). A number of participants of the NRC's workshop on CT agreed that a logical next step of future workshops would be to focus on pedagogical aspects of CT (NRC 2010). As a result, educational leaders' early discussions of CT sparked conversations as to what CT should look like in K-12 classrooms (Barr and Stephenson 2011). Consequently, the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) created a task force of leaders from education and industry to develop a K-12 computer science (CS) framework (CSTA Standards Task Force 2011) as well as an operational definition of CT (CSTA/ISTE 2011). According to the CSTA/ISTE (2011), computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables the use of a computer and other tools to solve them
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem-solving process to a wide variety of problems

Grover and Pea (2013) conducted a review of CT in K-12 education. After examining a broad range of definitions and perspectives on CT, they argued for a dramatic shift from defining CT to operationalizing and putting CT into practice (Grover and Pea 2013). Operationalizing CT continues to be supported by organizations like the CSTA and the Association for Computing Machinery (ACM) as they strive to develop and disseminate CT resources and educator toolkits across multiple K-12 disciplines. CSTA and ACM (2016) assembled an additional task force to revise CS standards to reflect recommended changes as well as align them with other K-12 standards such as the Common Core for Math and Science (CSTA 2016) as well as the Next Generation Science Standards (NGSS Lead States 2013). An updated version of the CSTA K-12 Computer Science Standards was released in 2017 (CSTA 2017). Since then, some states (e.g. Florida and Massachusetts) have drafted their own versions of computer science (CS) standards to reflect the 2017 CSTA CS standards, while a few states (e.g. the state of Washington) have outright adopted CSTA standards as their own (State of Computer Science Education 2018). In our analysis of coding toys' features, we use the constructs coming out of this work on standards to consider how features may support students' development of CT.

2.2 Defining CT

Meanwhile, educational leaders have yet to reach a consensus on a shared definition of CT (Denning 2017). Certain definitions of CT remain closely connected to computing

disciplines, namely computer science. For example, Brennan and Resnick (2012) provided a framework for studying and assessing the development of CT in young learners as they programmed using Scratch, an online community and authoring environment. Their CT definition consists of three key dimensions including: (1) computational concepts (2) computational practices and (3) computational perspectives (Brennan and Resnick 2012). Over a span of several years and various Scratch workshops, Brennan and Resnick (2012) identified common computational concepts (e.g. sequences, loops, parallelism, events, conditionals, operators, and data) young learners utilized as they designed and programmed in a virtual programming environment.

Some definitions for CT have been created within the context of other curricular disciplines outside of CS. For instance, Weintrop et al. (2016) performed a review of CT literature and interviewed experts in mathematics and science in order to develop a taxonomic definition of CT consisting of four categories: (1) data practices (2) modeling and simulation practices (3) computational problem-solving practices and (4) systems thinking practices. Others have linked CT with engineering education (Ehsan and Cardella 2017; Wing 2006). For example, Ehsan and Cardella (2017) identified CT competencies such as abstraction, algorithm and procedure, debugging/troubleshooting, pattern recognition, and simulation while observing first-grade students complete an engineering design task in an informal learning context.

While the above definitions have been linked to specific disciplines, other CT definitions have been developed using more of a cross-disciplinary approach. As a case in point, Shute et al. (2017) examined theoretical and empirical studies related to CT in K-12 education, and compared many characteristics and definitions of CT across a variety of disciplines. Based on their analysis, Shute et al. (2017) developed a definition for CT as the "conceptual foundation required to solve problems effectively and efficiently" (p. 151). Included with their definition, Shute et al. (2017) analyzed various CT concepts and categorized commonly identified CT skills into six main facets: decomposition, abstraction, algorithm design, debugging, iteration, and generalization. We used the definitions and facets by Shute et al. (2017) to support our categorization and analysis of coding toys.

2.3 Tools for Teaching CT

A variety of media, or tools, have been utilized to teach computational thinking skills. Some of the first tools used to teach computational thinking were related to Papert's work with the Logo educational programming language. Papert and his collaborators, Cynthia Solomon and Wallace Feurzeig, developed Logo as a child's tool for programming and commanding a robot turtle to physically move around a given space (Solomon and Papert 1976). Newer versions of the Logo programming environment allowed for a digital version of the robot turtle to be commanded within a programming environment. Papert's work with the Logo turtle helped to inspire many of the educational programming languages children use today, including Scratch (Resnick et al. 2009) and Scratch Jr. (Flannery et al. 2013).

Papert's work also inspired various other technological tools including educational toys and apps designed for young children. A wide spectrum of computationally-themed manipulatives and toys, ranging from digital versions to completely screen-less options, are available to teach young learners (Sullivan and Heffernan 2016). For instance, Primo Toy advertises on their website that Cubetto is not only based on Montessori philosophies, but it was also inspired by the Logo turtle ("Cubetto Universe," n. d.). In addition to her

collaborative work with Scratch Jr., Bers helped to co-found KinderLab Robotics, where they currently work to provide tangible, screen-free robotics kits (e.g. KIBO) for young children (Bers 2010).

3 Framework and Coding Scheme

Scholars and educators welcome the availability and diversity of computational devices, including programming tools like Dash and Dot, Robot Turtles, and Cubetto, yet much can be done to appropriately and effectively integrate these devices within classroom settings and K-12 curriculum (Manches and Plowman 2017). Moreover, further research is needed in order to determine whether or not such activities are developmentally appropriate and how they facilitate computational thinking in young children (Bers et al. 2014).

Yu and Roque (2018) surveyed physical, virtual, and hybrid computational kits (e.g. Cubetto, Scratch Jr., KIBO, etc.) and examined how each kit supported various computational thinking concepts and practices using Brennan and Resnick's (2012) CT framework. Ehsan et al. (2017) examined several digital media, specifically educational Apps (e.g. Daisy the Dinosaur, Kodable, Scratch Jr., etc.), and evaluated how each App supported CT competencies such as abstraction, debugging/troubleshooting, pattern recognition, simulations, etc. In this paper, we present our framework for analyzing computational-themed toys (or physical objects for children to play with) and the CT skills they are designed to teach.

In previous work, we evaluated the physical and ideational features of twenty different computational-themed toys (Hamilton et al. 2018). In this report, we extend our previous work by (1) expanding our categories of physical features to include dichotomous coding schemes as well as (2) adding CT skills to our classification scheme of ideational features. For this report and future research, we adopt and build upon Shute et al.'s (2017) definition of CT. To develop our definition of CT skills, we modified Shute et al.'s (2017) categorization of CT skills to include additional CT facets such as parallelism (Brennan and Resnick 2012) and efficiency.

3.1 Physical Features

Physical features of computational toys encompass visually-perceived aspects of toys that can be physically manipulated (e.g. manipulative controls on the body of the robot). We categorized physical features as how computational and programming skills are physically instantiated (Hamilton et al. 2018). For example, Cubetto has physical tiles that afford grasping and placing tiles within specific locations on the instruction board. Once the child hits the blue (execution) button, the robot will follow the order of the instruction tiles in sequence as it moves from start to finish. Not all computational toys have manipulable tiles, however. Some toys, like Robot Mouse, have physical buttons on the actual device that can be pressed in order for the user to "program" the robot to move in specified directions.

For this paper, we continue to use our first (and above mentioned) definition of physical features. However, we have amended our categorization scheme to include a 'dichotomous key' for evaluating CT toys' features based on coding the physical components of the toys (Table 1). In nature, living things exhibit a varying assortment of characteristics, and as a way to categorize and identify living things scientists have developed dichotomous keys to identify an organism at the taxonomic level. Similar to organisms, real-world objects can

Table 1 Dichotomous key for computational toys' physical features

1	a. The toy(s) are made of electronic components	Go to statement #2
	b. The toy(s) are made of non-electronic components	The toy belongs to the "Board Games and Books" category (e.g. Robot Races)
2	a. The toy(s) are robotic in nature, meaning it is a mechanical device capable of being programmed to carry out a series of actions	Go to statement #3
	b. The toy(s) are not robotic in nature	The toy belongs to the "Non-Robotic Electronics" category (e.g. Puzzlets)
3	a. The robotic toy(s) are screenless and have tangible features	Go to statement #4
	b. The robotic toy(s) are controlled remotely using an app or screen-based technology	The toy belongs to the "Screen-Based Robot" category (e.g. Dash and Dot)
4	a. The robotic toy(s) have buttons or tactile controls found on the body of the robot itself	The toy belongs to the "Button-Operated Robot" category (e.g. Bee-Bot)
	b. The robotic toy(s) or kit has external manipulative blocks, tiles, and/or pieces that control the robot	The toy belongs to the "Robot with Tangible Interface" category (e.g. Cubetto)

Disclaimer: If a toy is found to have characteristics of multiple toy categories, it belongs to the "Blended" category (i.e. Blue-bot)

be grouped together based on their characteristics. Computational toys also exhibit a great deal of variation; therefore, we propose a guide to classifying computational toys based on their physical features (Table 1).

Thus, we group computational toys into the following physical categories: (1) Board Games and Books, (2) Non-Robotic Electronics, (3) Screen-Based Robot, (4) Button-Operated Robot, (5) Robot with Tangible Interface, and (6) Blended. For example, let us consider a toy called Bee-bot. Bee-bot is a robotic floor robot shaped like a bee. It has directional keys, or manipulative controls, on its back. Therefore, we categorize the Bee-bot as a 'Button-Operated Robot.' However, Blue-bot, a newer generation of the Bee-bot, has bluetooth access and can be programmed using app-based technology (e.g. iPad). Blue-bot also has manipulative controls on its body. Because Blue-bot has physical features of multiple categories we would classify it as a "Blended" computational toy.

Our physical feature classification allows us to group toys based on whether or not they have an internal or external interface, and if they are controlled by an external app. Our classification applied to thirty toys is presented in Table 2.

In Fig. 2, we present the results of our evaluation of thirty computational toys based on their physical features. The thirty toys were selected for review based on the following criteria: (1) the toy is marketed to target five- and 6-year-old children and (2) the toy is marketed as teaching computational or programming skills. We searched for publications in the Education Source, ERIC, and PsychINFO using keywords such as "computational toys," "programming," "early childhood," "education," and "STEM" to search for research projects affiliated with computational toys. We searched for additional computational toys using online retail platforms such as Amazon and Target. We used the following keywords, as recommended by Amazon, to search for toys: "coding toys," "programming toys," and "STEM toys." Additionally, we also searched for computational toys funded by Kickstarter campaigns (e.g. Unruly Splats).

Figure 2 shows that the majority of computational toys (ten in total) that we reviewed were categorized as 'Screen-Based Robots' because the majority of computational toys targeting 5- and 6-year old children are robots that can be programmed by a secondary external technological device (e.g. tablet). The category to represent the least number of computational toys (two in total) was that of 'Button-Operated Robot' toys (e.g. Bee-Bot).

3.2 Ideational Features

Physical features are characteristics that can be observed visually. However, we need to go beyond arbitrarily describing what we can observe by using merely our senses. After all, doing so may lead us to generalize or make unnecessary assumptions. For instance, early taxonomists primarily relied upon the physical traits of organisms to classify living things into different categories, but anatomical features alone did not account for complex biological relationships or evolutionary processes that led to such characteristics. Therefore, in addition to examining physical features, we explore beyond superficial learning features and examine the internal knowledge, specifically ideational CT skills and competencies, which children are developing as they interact with computational toys.

During the design process, idea generation, or ideation, occurs when designers work to create solutions when tackling a problem (Norman 2002). Not only do educational tools have physical features (as discussed above), but they also have design features, or ideational features, intended to teach a particular idea or skillset. Therefore, as children interact with features of CT toys, they are also generating possible solutions to computational

Table 2 Thirty toys classified by physical features

Name of toy	Physical feature classification
Augie AR Coding Robot	Screen-Based Robot
Bee-Bot Robot	Button-Operated Robot
Botley the Coding Robot	Robot with Tangible Interface
Blue-Bot Robot	Blended
Code and Go Robot Mouse	Button-Operated Robot
Code Monkey Island	Board Games and Books
Coder Bunnyz	Board Games and Books
Coji Robot	Screen-Based Robot
Cubetto	Robot with Tangible Interface
Dash	Screen-Based Robot
Dot	Screen-Based Robot
Finch Robot	Screen-Based Robot
FurReal Maker: Proto Max	Screen-Based Robot
Future Coders Bunny Trails	Blended
Future Coders Robot Races	Board Games and Books
Harry Potter Kano Coding Kit	Non-Robotic Electronics
KIBO	Robot with Tangible Interface
Let's Go Code Activity Set	Board Games and Books
Makeblock Codey Rocky Robot	Screen-Based Robot
Makeblock mBot Robot Kit	Screen-Based Robot
Mindware Code Hopper	Board Games and Books
Osmo Coding Awbie	Non-Robotic Electronics
Ozobot Bit	Blended
Puzzlets	Non-Robotic Electronics
Robot Turtles	Board Games and Books
Root the Robot	Screen-Based Robot
Siggy Scooter	Screen-Based Robot
Thames and Kosmos Coding and Robotics	Robot with Tangible Interface
Think and Learn Code-a-pillar	Robot with Tangible Interface
Unruly Splats	Non-Robotic Electronics

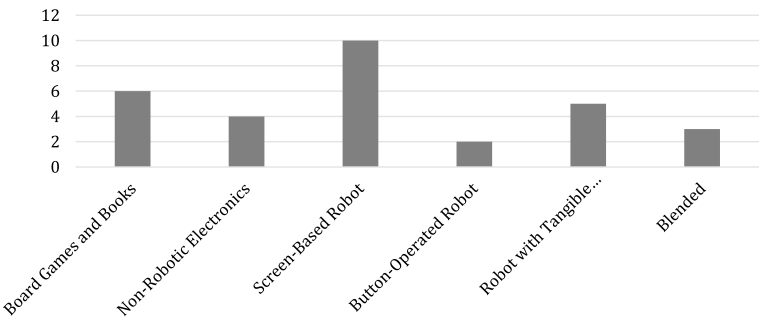

Fig. 2 Number of computational-themed toys assigned to physical feature classification

challenges affiliated with the toy's design. In terms of computational toys, we refer to ideational features as the design features intended to provoke CT skills (e.g. pattern recognition and algorithmic thinking; Hamilton et al. 2018). CT skills are the skills that emerge as a result of children's interactions with both the physical and ideational features of CT toys. Thus, we developed a framework for evaluating CT skills children use as they interact with CT toys. Previous research has identified CT competencies (e.g. Ehsan and Cardella 2017; Shute et al. 2017) as well as elements of CT (Angeli et al. 2016). We modified these facets (Ehsan and Cardella 2017; Shute et al. 2017; Angeli et al. 2016) to create our skill-based definitions. Our skill-based definitions focus on CT competencies that are measurable and observable (see Table 3).

Children who engage in playful programming activities with computational toys utilize CT skills as they participate in computational challenges. Children use algorithmic thinking (Selby and Woollard 2013; Angeli et al. 2016) as they create a sequence of instructions for Cubetto to follow. Additionally, as children program Cubetto to move from one location to the next, they continue to break down a complex problem into smaller parts, commonly referred to as a problem decomposition (NRC 2010; Angeli et al. 2016). Debugging, another CT competency, is used frequently as both children fix errors within their program (Selby and Woollard 2013; Angeli et al. 2016). The designed features of various other computational toys also allow for children to use and develop their CT skills. In the next section, we show how these CT competencies can be turned into measureable CT skills to be utilized by practitioners in classroom settings.

In addition to evaluating toys based on their physical features, we evaluated toys based on types of CT skills (Table 3) children use as they interact with CT toys. We have amended our initial evaluation of CT toys and skills (Hamilton et al. 2018) to include additional CT skills: parallelism, efficiency, automation, and iteration. In Fig. 3, we present the results of our evaluation of the thirty toys identified in Table 2 based on their ideational CT skills. We found that when children interacted with all thirty of the examined toys they used algorithmic thinking skills. The CT toys also emphasize problem decomposition, scaffolded debugging, and pattern recognition. The least of the skills to be emphasized include higher-order thinking skills like parallelism, efficiency, and iteration.

Many of the toys we reviewed focus on lower order computational thinking skills such as sequencing and debugging (Hamilton et al. 2018). The *Bebras* community, an association of organizations involved with informatics and computing education, annually arranges an international problem-solving challenge which includes tasks related

Table 3 Summary of measurable ideational CT skills and their definitions

CT skill	Definition
Algorithmic thinking	Creating and following sequence of instructions to complete a task
Parallelism	Carrying out tasks or steps at the same time to improve efficiency
Efficiency	Designing a solution to have fewest number of steps
Automation	Modifying or remixing portions of an existing program to solve similar problems
Scaffolded debugging	Finding and fixing goal-deviant errors while developing a solution
Problem decomposition	Breaking down a problem or task into smaller, more manageable parts
Abstraction	Identifying main ideas and define reusable routines
Pattern recognition	Identifying patterns or trends within data or information
Iteration	Repeating design processes to refine solutions

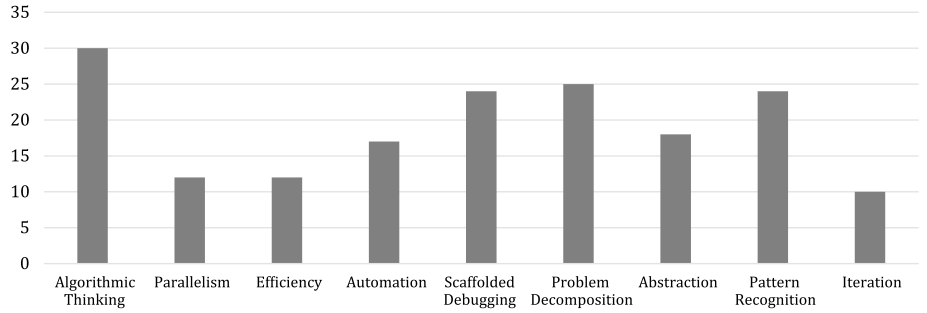

Fig. 3 Distribution of CT skills of examined computational-themed toys

to programming, algorithms, and computational thinking. Similar to our earlier findings, Izu et al. (2017) analyzed *Bebras* challenges focused on computational tasks and found that lower order computational thinking skills such as algorithmic thinking and data representation were represented amongst different age groups (grades K-12) of international students. Izu et al. (2017) recommend to the *Bebras* challenge creators to expand on the other CT categories and keep a library of the tasks as they are classified. Similarly, we encourage toy manufacturers to also expand upon higher-order computational thinking skills such as abstraction, as young children are quite capable of thinking abstractly (Gibson 2012).

Technological tools and technology-supported activities are becoming more prevalent in early childhood programs. For this report, we evaluated many of the latest computational toys, and yet, such technologies continue to emerge within the marketplace quite rapidly. Because such technologies are "here to stay" (NAEYC 2012, p. 2), we must make an effort to understand these complex learning interactions and any impacts they have upon childhood development and learning. Not only should such technologies be aligned with learners' developmental needs, but their use should complement curricular goals while further developing foundational skills (Haugland 2000). In addition to curricular alignment, early childhood educators should be provided with necessary resources, support, and training in order to realize effective technology integration (NAEYC 2012).

Our emerging technology report adds to a growing body of research regarding computational manipulatives and the affordances they provide for young learners (Sullivan and Heffernan 2016). Further research is needed in developing and evaluating computational thinking skills in early childhood (Manches and Plowman 2017; Bers et al. 2014). In future work, we plan to expand upon pedagogical research in early childhood and further investigate how CT skills can be observed and measured in early childhood settings. Some of our goals include to develop resources and provide pedagogical support for integration of computational toys in early classroom settings. We aim to understand how early computational seeds of learning manifest themselves and how educators can continue to nurture these seeds as children grow and develop.

Funding This research was supported in part by an internal Research Catalyst Grant from the Research and Graduate Studies Office at Utah State University and in part by the National Science Foundation (Award #1842116).

References

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.

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.

Bers, M. U. (2010). The TangibleK robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice (ECRP), 12*(2), 1–20.

Bers, M. U. (2018). Coding and computational thinking in early childhood: The impact of Scratch Jr. in Europe. *European Journal of STEM Education, 3*(3), 8. https://doi.org/10.20897/ejsteme/3868.

Bers, M. U., Flannery, L., Kazakoff, E. R., & 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.2013.10.020.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*. Retrieved from http://web.media.mit.edu/»kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf. Accessed 11 Feb 2019.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051–1058. https://doi.org/10.1037/0022-0663.76.6.1051.

Computer Science Teachers Association. (2016). *Interim CSTA K–12 computer science standards*. Retrieved from CSTA website https://cdn.ymaws.com/www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf. Accessed February 11, 2019.

Computer Science Teachers Association. (2017). *Revised CSTA K–12 computer science standards*. Retrieved from CSTA website https://www.doe.k12.de.us/cms/lib/DE01922744/Centricity/Domain/176/CSTA%20Computer%20Science%20Standards%20Revised%202017.pdf. Accessed February 11, 2019.

Computer Science Teachers Association (CSTA) Standards Task Force. (2011). *In CSTA K-12 computer science standards*. Retrieved from CSTA website http://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf. Accessed February 11, 2019.

Computer Science Teacher Association (CSTA), & International Society for Technology in Education (ISTE). (2011). *Operational definition of computational thinking for K-12 education*. Retrieved from CSTA website http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf. Accessed February 12, 2019.

Cubetto Universe – New Maps and Adventures. (n.d.). https://www.kickstarter.com/projects/primotoys/cubetto. Accessed February 12, 2019.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM, 60*(6), 33–39. https://doi.org/10.1145/2998438.

DiSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Ehsan, H., Beebe, C., & Cardella, M. E. (2017). Promoting computational thinking in children using apps. Paper presented at 2017 ASEE annual conference and exposition, Columbus, Ohio. Retrieved from https://peer.asee.org/28772. Accessed 11 Feb 2019.

Ehsan, H., & Cardella, M. E. (2017). Capturing the computational thinking of families with young children. Paper presented at 2017 ASEE annual conference and exposition. Columbus, Ohio. Retrieved from https://peer.asee.org/capturing-the-computational-thinking-of-families-with-young-children-in-out-of-school-environments. Accessed 11 Feb 2019.

Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013). Designing Scratch Jr.: Support for early childhood learning through computer programming. In *Proceedings of the 12th international conference on interaction design and children*. https://doi.org/10.1145/2485760.2485785.

Gibson, J. P. (2012). Teaching graph algorithms to children of all ages. In *Proceedings of the 17th ACM annual conference on innovation and technology in computer science education* (pp. 34–39). ACM. https://doi.org/10.1145/2325296.2325308.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Hamilton, M. M., Clarke-Midura, J., Shumway, J. F., & Lee, V. R. (2018). An Initial Examination of Designed Features to Support Computational Thinking in Commercial Early Childhood Toys. In J.

Kay & R. Luckin (Eds.), *Rethinking Learning in the Digital Age: Making the Learning Sciences Count*, 13th International Conference of the Learning Sciences (ICLS) 2018 (Vol. 3, pp. 1739-1740). London: ISLS.

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments, 1,* 1–32. https://doi.org/10.1080/1049482900010102.

Haugland, S. W. (2000). What role should technology play in young children's learning? Part 2. Early childhood classrooms in the 21st century: Using computers to maximize learning. *Young Children, 55*(1), 12–18.

Izu, C., Mirolo, C., Settle, A., Mannila, L., & Stupuriene, G. (2017). Exploring Bebras tasks content and performance: A multinational study. *Informatics in Education, 16*(1), 39–59.

Manches, A., & Plowman, L. (2017). Computing education in children's early years: A call for debate. *British Journal of Educational Technology, 48*(1), 191–201.

NAEYC. (2012). *Position statement of the National Association for the Education of Young Children and the Fred Rogers Center for Early Learning and Children's Media at Saint Vincent College*. https://www.naeyc.org/resources/topics/technology-and-media. Accessed February 11, 2019.

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking*. http://www.nap.edu/catalog.php?record_id=12840. Accessed February 12, 2019.

NGSS Lead States. (2013). *Next generation science standards: For states, by states*. Washington, DC: The National Academies Press. Retrieved from https://www.nextgenscience.org/. Accessed February 12, 2019.

Norman, D. (2002). *The design of everyday things*. New York: Basic Books.

Papert, S. (1980). *Mindstorms*. Brighton: Harvester Press.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60. https://doi.org/10.1145/1592761.1592779.

Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition*. Retrieved from http://eprints.soton.ac.uk/356481. Accessed 11 Feb 2019.

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

Solomon, C. J., & Papert, S. (1976). A case study of a young child doing turtle graphics in LOGO. In *Proceedings of the June 7–10, 1976, national computer conference and exposition on—AFIPS'76* (p. 1049). New York, New York: ACM Press. https://doi.org/10.1145/1499799.1499945.

State of Computer Science Education. (2018). https://advocacy.code.org/. Accessed February 12, 2019.

Sullivan, F. R., & Heffernan, J. (2016). Robotic construction kits as computational manipulatives for learning in the STEM disciplines. *Journal of Research on Technology in Education, 48*(2), 105–128.

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. In *Proceedings of the 16th Koli Calling international conference on computing education research (Koli Calling'16)* (pp. 120–129). Koli, Finland: ACM Press, https://doi.org/10.1145/2999541.2999542.

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.

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

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of The Royal Society, 366,* 3717–3725.

Yu, J., & Roque, R. (2018). A survey of computational kits for young children. In *Proceedings of the 17th ACM conference on interaction design and children* (IDC'18) (pp. 289–299). Trondheim, Norway: ACM Press. https://doi.org/10.1145/3202185.3202738.