Computational Thinking in K–12: A Review of the State of the Field

Shuchi Grover[1] and Roy Pea[1,2]

Jeannette Wing's influential article on computational thinking 6 years ago argued for adding this new competency to every child's analytical ability as a vital ingredient of science, technology, engineering, and mathematics (STEM) learning. What is computational thinking? Why did this article resonate with so many and serve as a rallying cry for educators, education researchers, and policy makers? How have they interpreted Wing's definition, and what advances have been made since Wing's article was published? This article frames the current state of discourse on computational thinking in K–12 education by examining mostly recently published academic literature that uses Wing's article as a springboard, identifies gaps in research, and articulates priorities for future inquiries.

Keywords: computational thinking; computing education; computational literacy; computers and learning; K–12 curricula; learning environments; problem solving; STEM learning; student cognition; technology

Introduction

Six years ago, Jeannette Wing's succinct and influential article, "Computational Thinking," appeared in the Viewpoint section of the March 2006 edition of the *Communications of the ACM* with the pronouncement: "It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (p. 33).

Wing's arguments caught the attention of a broad academic community. Prompted by her article and a growing community of researchers, educators, and policymakers, computational thinking (or CT) as a concept and associated research agenda has witnessed increasing attention and research. The tailwinds in the larger environment have fanned this growing interest. The issue of Computer Science (CS) Education in K–12 took center-stage following a stark report titled *Running on Empty: The Failure to Teach K–12 Computer Science in the Digital Age* (Wilson, Sudol, Stephenson, & Stehlik, 2010) revealed precipitously low numbers for women in computing and that more than two thirds of the country had few computer science standards at the secondary school level. Concerns about these statistics deepen given projections from the Bureau of Labor Statistics (http://www.bls.gov/ooh/) that computing is one of the fastest-growing job markets through 2018. This CS education imperative has dovetailed with the science policy attention to science, technology, engineering, and mathematics (STEM) learning in the United States since the turn of the 21st century. With CT being viewed as at the core of all STEM disciplines (Henderson, Cortina, Hazzan, & Wing, 2007) it appears that computing in K–12 is an idea whose time has come.

Of course, the idea of CT is not new. Back in the 1960s, Alan Perlis argued for the need for college students of all disciplines to learn programming and the "theory of computation" (Guzdial, 2008). However, in the context of K–12 education, computing first gained popular traction around Seymour Papert's MIT work in the 1980s. Papert pioneered the idea of children developing procedural thinking through LOGO programming (Papert, 1980, 1991). This recent resurgence takes a fresh, "21st century" perspective on the topic, and Wing's 2006 article forms a logical starting point for our critical examination of the current state of the field of CT in K–12 education. The following sections examine mostly recently published, salient, academic literature that has used Wing's article as a springboard. The article will also report on key efforts around computing education in K–12.

Given the definitional confusion that has plagued CT as a phrase and how imperative it is for school education, the next section looks deeply at the varied perspectives and evolving definitions of CT, the rationale for building CT among school children, and common criticisms against CT in schools. The article then surveys recent research investigating CT (including some that do not use the phrase *computational thinking* per se but nonetheless examine computational competencies in children), the various environments and tools that are believed to foster CT development, and studies attempting to assess CT are appraised. Finally, the article lays out priorities for broadening the K–12 CT discourse on the basis of the gaps in current research.

[1]Stanford University School of Education, Stanford, CA, USA
[2]H-STAR Institute, Stanford, CA, USA

Educational Researcher, Vol. 42 No. 1, pp. 38–43
DOI: 10.3102/0013189X12463051

The What and Why of Computational Thinking

According to Wing (2006), "computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p. 33). CT's essence is thinking like a computer scientist when confronted with a problem.

Wing's call to action for CT in school education served as the starting point for two National Academy of Sciences workshops convening leading researchers from education, learning sciences and computer science departments, and leaders from the computing industry, to explore "the nature of computational thinking and its cognitive and educational implications" (National Research Council [NRC], 2010, p. viii) and the pedagogical aspects of computational thinking (NRC, 2011). In the first workshop, early notions of CT that focused on procedural thinking and programming (Papert, 1980, 1991), though still considered valid, were revisited and broadened to encompass several core concepts of computer science that take it beyond "just programming." The workshop, however, threw into sharp relief the lack of consensus that seems to have bedeviled this space. Some of the central questions left unanswered by the workshop included the following: How can CT be recognized? What is the best pedagogy for promoting CT among children? Can programming, computers, and CT be legitimately separated? (NRC, 2010). Some of these questions were reexamined in the follow-up workshop that focused on better defining the space by gathering and synthesizing insights from educators addressing CT in their work with K–12 teachers and learners. The aim of the workshop was to share examples and best practices of pedagogies and environments for teaching CT and revealed a plethora of perspectives that reflected several tools and pedagogies that are legitimate candidates for use in developing these competencies.

Wing (2011) revisited the topic and clarified, "Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent." Aho (2012) simplified this further by defining CT as the thought processes involved in formulating problems so "their solutions can be represented as computational steps and algorithms" (p. 832).

Recently, the Royal Society (2012) also offered a succinct and tractable definition that captures the essence of CT—"Computational thinking is the process of recognising aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes" (p. 29).

A valuable perspective that breaks down the meaning of CT, especially for high school curricula, comes from the CS Principles course being piloted by the College Board and the National Science Foundation (NSF) (http://www.csprinciples.org/). The course focuses on the *practices* of computational thinking and is based on the seven "big ideas" of computing:

1. Computing is a creative human activity
2. Abstraction reduces information and detail to focus on concepts relevant to understanding and solving problems
3. Data and information facilitate the creation of knowledge
4. Algorithms are tools for developing and expressing solutions to computational problems
5. Programming is a creative process that produces computational artifacts
6. Digital devices, systems, and the networks that interconnect them enable and foster computational approaches to solving problems
7. Computing enables innovation in other fields, including science, social science, humanities, arts, medicine, engineering, and business.

Following workshops organized by the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE), Barr and Stephenson (2011) provided a similar "operational definition of CT" aimed at K–12 teachers that comprised an explanatory checklist for what CT means along with an enumeration of core CT concepts and capabilities, and examples of how they might be embedded in activities across multiple disciplines.

It is worth noting here that the potent idea of "computational literacy" (diSessa, 2000) pre-dates Wing's charter for CT for all. Although the essence of both concepts targets this new digital age competency, diSessa separates the "material" tools such as programming environments, from the "cognitive" and the "social" aspects of computational literacy. Furthermore, diSessa underscores the use of "computing as a medium" for exploring other domains such as math and science, much like Kay and Goldberg (1977) explored math, science, and art via programming in Smalltalk. This notion is often neglected in popular definitions of CT. The term *computational literacy* is perhaps susceptible to confusion with earlier ones like *computer literacy, information literacy*, and *digital literacy* that have assumed various meanings over the years and fall well short of what diSessa demands of computational literacy. Although the phrase and notion of *computational thinking* now seems to be preferred over *computational literacy*, in research and practice today the two phrases are often used interchangeably.

Procedural literacy is another avatar of CT that was first proposed in 1980 by B. A. Sheil at Xerox PARC. In our reading, there is little to distinguish between procedural literacy and CT applied mostly to creating video games and other computational media artifacts or, more broadly, the practice of CT in the context of new media art and design.

Researchers and CS educators for the most part now work broadly with the aforementioned recent descriptions of CT. The value of *abstraction* as CT's keystone (distinguishing it from other types of thinking) is undisputed. Abstraction is "defining patterns, generalizing from specific instances," and a key to dealing with complexity (Wing, 2011). The following elements are now widely accepted as comprising CT and form the basis of curricula that aim to support its learning as well as assess its development:

- Abstractions and pattern generalizations (including models and simulations)
- Systematic processing of information
- Symbol systems and representations
- Algorithmic notions of flow of control

- Structured problem decomposition (modularizing)
- Iterative, recursive, and parallel thinking
- Conditional logic
- Efficiency and performance constraints
- Debugging and systematic error detection

Programming is not only a fundamental skill of CS and a key tool for supporting the cognitive tasks involved in CT but a demonstration of computational competencies as well. Noteworthy efforts like CS Unplugged (http://csunplugged.org/) that introduce computing concepts without the use of a computer, while providing valuable introductory activities for exposing children to the nature of CS, may be keeping learners from the crucial computational experiences involved in CT's common practice.

Finally, although there is broad acknowledgement that computing pervades all aspects of the global economy, its place as a mandatory part of the school curriculum is far from secure. Many criticisms have revolved around these multiple interpretations of CT and a lack of clarity among educators on CS as a discipline. Another valid concern is whether there is a compelling rationale for all children, including those who allege no interest in pursuing CS and STEM careers, to develop computational competencies in school. In the zero-sum school curriculum map, how should curriculum policymakers make room in already packed school curricula? There is also lack of agreement on whether CT should ultimately be incorporated into education as a general subject, a discipline-specific topic, or a multidisciplinary topic (NRC, 2011). Lastly, there is some question whether CT is distinct enough from other forms of thinking that children are developing. Advocates of CT concede that although it shares elements with mathematical, engineering, and even design thinking, and draws on a rich legacy of related frameworks, it also extends each of those thinking skills in a unique way (Lee et al., 2011). Denning and Freeman (2009) observe that although the computing paradigm "contains echoes of engineering, science, and mathematics, it is distinctively different because of its central focus on information processes" (p. 30) and that Wing's CT interpretation embeds well into this system of practice.

We claim that the approach to problem solving generally described as CT is a recognizable and crucial omission from the expertise that children are expected to develop through routine K–12 Science and Math education (although CT has finally been mentioned, albeit briefly, in the 2012 NRC K–12 Science Education framework). If basic literacy in Math and Science can be considered essential for all children to understand how our world works, why should school education not lift the hood on all-pervasive computing devices as well? We believe that those in possession of computational competencies will be better positioned to take advantage of a world with ubiquitous computing. Early experiences with this way of problem solving will not only alleviate problems in introductory CS courses undergraduates have been known to face but also generate interest and prime students for success in this growing field rife with opportunity.

Recent news from media and industry suggest that the move to make programming a more commonplace skill for everyone and introducing 'rithms (short for algorithms) as the fourth "r" for 21st-century literacy is gaining momentum globally. Israel has long boasted an exemplary mandatory high school CS curriculum. Countries such as Russia, South Africa, New Zealand, and Australia have already made room for CS in the K–12 curriculum. More recently, the United Kingdom has piloted programs to teach computing to all schoolchildren following a bold 2012 policy charter from the Royal Society.

Summary of Pertinent Research on CT in K–12

With broadly agreed on definitions of CT in K–12 education, focus has recently shifted to tackling the more practical questions of how to promote and assess the development of CT. There is extensive literature from the last three decades tackling issues of teaching and learning programming and CS. The bulk of CS education research, however, is set in the context of undergraduate classrooms. Although there is much to learn about CT in K–12 both from studies of kids and programming in the 1980s (using languages such as LOGO and BASIC) as well as early programming and CS experiences of college students, the space constraints imposed by the essay as well as a focus on the recent resurgence of CT force the review to be limited to *recent research involving 21st-century tools and school-age children.*

Environments and Tools That Foster CT

The idea of "low floor, high ceiling" as one of the guiding principles for the creation of programming environments for children has been around since the days of LOGO. It essentially means that though it should be easy for a beginner to cross the threshold to create working programs (low floor), the tool should also be powerful and extensive enough to satisfy the needs of advanced programmers (high ceiling). Computationally rich environments and effective CT tools for school children must have low threshold and high ceiling, scaffold, enable transfer, support equity, and be systemic and sustainable (Repenning, Webb, & Ioannidou, 2010). Several programming tools fit these criteria to varying degrees. Popular among these are graphical programming environments such as Scratch, Alice, Game Maker, Kodu, and Greenfoot; Web-based simulation authoring tools such as Agentsheets and Agentcubes; and robotics kits and tangible media such as Arduino and Gogo Boards. Graphical programming environments are relatively easy to use and allow early experiences to focus on designing and creating, avoiding issues of programming syntax. By allowing novices to build programs by snapping together graphical blocks that control the actions of different dynamic actors on a screen, environments like Scratch, MIT's popular offering, quite literally make programming a snap.

Several of these introductory computational experiences use the three-stage "use–modify–create" progression to help the learner go from user to modifier to creator of computational artifacts (Lee et al., 2011), a progression first broadly used in Apple's Hypercard application in the mid-1980s to early 1990s. Curricular activities such as game design and robotics have typically served well as a means for the iterative exploration of CT, making them ideal not only for motivating and engaging school children but for introducing them to computer science. Visual and tangible programming experiences are often followed by

exposure to high-level programming languages such as Python, Java, and Scheme.

Recommendations for engaging girls through computing in context (Margolis & Fisher, 2002; also see Cooper & Cunningham, 2010) provide a compelling rationale for tools that strive to bridge the gender gap in the computing field. Emerging computational environments are poised to provide more opportunities for engagement in CT in formal and informal settings while also engaging girls as well. E-textiles and other "computational craft" kits that use small, powerful hardware, such as the Lilypad Arduino, allow children to combine traditional arts and crafts such as sewing and sketching with computation and electronics. MIT App Inventor, a visual programming environment that uses Scratch-like graphical blocks of code for building Android mobile apps, is more gender neutral and complete than most tools. It sets a low floor for allowing creative app building (something all teens, including girls, are eager to do) while still engaging with complex CT concepts including procedural and data abstraction, iterative and recursive thinking, structured task breakdown, conditional and logical thinking, and debugging.

Despite its growing popularity for promoting many 21st-century competencies in K–12 (NRC, 2012), video gaming as a platform for examining CT among children has been underutilized in recent research. Holbert and Wilensky (2011) successfully developed and tested a prototype video game, FormulaT, which aimed to serve as a platform for learning principles of kinematics as well as "systematic computational strategies." FormulaT used NetLogo, a computational environment for agent-based modeling. The activities of abstracting pertinent behaviors into agents, applying rules, and evaluating the results via modeling and simulation are key ways of engaging in CT. Blikstein (2010) demonstrates leveraging Netlogo computational models for science learning in secondary-level classrooms. Agent-based modeling, however, remains relatively underused in CT research.

Not surprisingly, current computational tools vary in their effectiveness in allowing for engagement with the various component elements of CT. Maloney, Peppler, Kafai, Resnick, and Rusk (2008) reported demonstration of several CT elements such as conditional logic, iterative and parallel thinking, and data abstraction in Scratch programs created by urban youth in afterschool settings. However Scratch lacks the means to abstract functionality into functions and procedures, prompting a version called Snap! from Berkeley that seeks to address this. Perhaps an imperative for CS in K–12 will fuel the development of new tools built expressly for fostering CT among school-age children. These should not only embody all the characteristics of effective CT tools and promote the development of all the competencies now identified as elements of CT but also be guided by recent research on commonsense human understanding of computing and how children explain their approaches to problem solving (Pane, Ratanamahatana, & Myers, 2001; Simon, Chen, Lewandowski, McCartney, & Sanders, 2007).

Lastly, despite the variety of environments in which current CT research is situated, many promising spaces are still untapped; Fab Labs, Makerspaces, and DIY movements, such as Maker Faire and Instructables, that promote construction of tangible computational artifacts, informal "hacker" events for kids, as well as ubiquitous and powerful smartphones, all present exciting possibilities.

Assessment of CT

Without attention to assessment, CT can have little hope of making its way successfully into any K–12 curriculum. Furthermore, to judge the effectiveness of any curriculum incorporating CT, measures that would enable educators to assess what the child has learned need to be validated.

Most recent research addressing questions of CT assessment, such as Werner, Denner, Campe, and Kawamoto's (2012) *Fairy Assessment* in Alice, has used either student-created, or predesigned programming artifacts to evaluate students' understanding and use of abstraction, conditional logic, algorithmic thinking, and other CT concepts to solve problems. Ideas of deconstruction, reverse engineering, and debugging to assess children's understanding in computational contexts have long enjoyed educational appeal. Fields, Searle, Kafai, and Min (2012) evaluated students' engineering and programming skills as they debugged prebuilt faulty e-textile projects. Han Koh, Basawapatna, Bennett, and Repenning (2010) attempted with some success to assess the thorny issue of transfer to answer questions like "Now that the student can program Space Invaders, can the student program a science simulation?"

In the past two decades, "academic talk" has been leveraged for promoting and assessing math and science literacy. The development in the student use of the vocabulary and language of CS over the course of engaging in computationally rich activities provides an additional instrument for measuring the growth of CT (Grover, 2011).

Computing Education in K–12

Wilson and Guzdial (2010) maintain that although the national urgency for strengthening STEM in K–12 has translated into billions of dollars in funding, research explicitly in computing education remains underfunded. NSF initiatives such as CPATH, BPC, and most recently, CE21 have gone a long way in energizing projects aimed at bringing CT/CS concepts to the secondary level. An additional boost for guiding interested middle and high school students into CS careers comes from DARPA's initiatives such as CS-STEM and Carnegie Mellon University's FIRE (Fostering Innovation through Robotics Exploration).

Although ongoing research in development of CT will help inform computing curricula throughout K–12, preparing teachers for computing education and ensuring gender equity remain huge challenges. The NSF's CS10K initiative aims to add 10,000 new CS teachers in U.S. high schools by 2015. The Georgia Computes! alliance is at the forefront of nationwide efforts for teacher preparation, development of CT/CS K–12 curricula as well as motivating female students in CS. Georgia Tech's Guzdial argues in his blog (http://computinged.wordpress.com/) that challenges to meeting the CS10K deadline include answering questions like the following: What do teachers need in order to develop into successful computer science teachers? What kind of pedagogy will fit into the lives of in-service high school teachers? What is Computer Science Pedagogical Content Knowledge?

In terms of curriculum, besides CS Principles for AP CS, the Exploring CS curriculum (http://www.exploringcs.org) is

intended to be a 1-year college preparatory curriculum for high school students. Other initiatives aimed at introducing CS into schools include CS4HS (http://www.cs4hs.com/) and Computing in the Core (http://www.computinginthecore.org/)—both of which represent collaborations between academia, national bodies, and organizations such as Microsoft and Google. CSTA's Model Curriculum for K–12 Computer Science provides curricular suggestions to help build interest, engage, and motivate students in CS. In addition, Google's Exploring Computational Thinking website (www.google.com/edu/computational-thinking) has a wealth of links to CT resources on the web. ACM has also recently introduced a new thread, EduBits, in its ACM Inroads quarterly that highlights principal educational activities within ACM and affiliated organizations.

Broadening the Scope of the Discourse and Priorities for Empirical Inquiry

It is thus quite evident that much of the recent work on CT has focused mostly on definitional issues, and tools that foster CT development. Some strides have been made in the realm of defining curricula for nurturing computational competencies, and assessing their development. Large gaps, however, still exist that call out for empirical inquiries.

In a view that was echoed by Alfred Aho, Wing argued, "an application of the science of learning research in designing grade- and age-appropriate curricula for computational thinking is necessary to maximize its impact on and significance for K–12 students" (NRC, 2011, p. 4). Barring some recent studies, such as Fadjo, Lu, and Black (2009) and Berland and Lee (2011), few others have taken into account contemporary research in the learning sciences in socio-cultural and situated learning, distributed and embodied cognition, as well as activity, interaction and discourse analyses. Cognitive aspects of children and novices learning computational concepts were studied extensively in the 1980s—issues such as development of thinking skills (Kurland, Pea, Clement, & Mawby, 1986); debugging (Pea, Soloway, & Spohrer, 1987); problems with transfer (Clements & Gullo, 1984; Pea & Kurland, 1984); use of appropriate scaffolds for successful transfer (Klahr & Carver, 1988), to name a few. That body of literature should be brought to bear on 21st-century CT research.

Also underinvestigated is the idea of computing as a medium for teaching other subjects—dovetailing the introduction of CT at K–12 with transfer of problem-solving skills in other domains. Past work includes demonstrations of children successfully designing LOGO software to teach fractions (Harel & Papert, 1990) and science (Kafai, Ching, & Marshall, 1997), and using modeling software in science (Metcalf, Krajcik, & Soloway, 2000).

Empirical studies on CT in schoolchildren could leverage extensive research on the types of problems beginner CS undergraduates face in their early programming experiences that go beyond syntactical issues: Are there well-defined hurdles or targets of difficulty that exist in the path of developing some elements of CT in children (e.g., recursion)? If so, what are these and how can they be addressed?

Also largely untapped is the territory of dispositions for, attitudes toward, and stereotypes concerning CT and CS, and how they relate to the development of learner identity (Mercier, Barron, & O'Connor, 2006). How crucial are these as we strive to provide both girls and boys with learning experiences that aim to nurture CT competencies? Recent incipient work on surveys of student attitudes toward computing represents a start in gaining a better understanding of this.

Clearly, much remains to be done to help develop a more lucid theoretical and practical understanding of computational competencies in children. What, for example, can we expect children to know or do better once they've been participating in a curriculum designed to develop CT and how can this be evaluated? These are perhaps among the most important questions that need answering before any serious attempt can be made to introduce curricula for CT development in schools at scale. It is time to redress the gaps and broaden the 21st-century academic discourse on computational thinking.

ACKNOWLEDGMENT

We gratefully acknowledge grant support of the LIFE Center from the National Science Foundation for this work (NSF-0835854).

REFERENCES

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

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.

Berland, M., & Lee, V. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning*, *1*(2), 65–81.

Blikstein, P. (2010). Connecting the science classroom and tangible interfaces: the bifocal modeling framework. In *Proceedings of the 9th International Conference of the Learning Sciences*, Chicago, IL, 128–130.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognitions. *Journal of Educational Psychology*, *76*, 1051–1058.

Cooper, S., & Cunningham, S. (2010). Teaching computer science in context. *ACM Inroads*, *1*, 5–8.

Denning, P., & Freeman, P. (2009). Computing's paradigm. *Communications of the ACM*, *52*(12), 28–30.

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge: MIT Press.

Fadjo, C. L., Lu, M., & Black, J. B. (2009, June). *Instructional embodiment and video game programming in an after school program*. Paper presented at the World Conference on Educational Multimedia, Hypermedia & Telecommunications, Chesapeake, VA.

Fields, D. A., Searle, K. A., Kafai, Y. B., & Min, H. S. (2012). Debuggems to assess student learning in e-textiles. In *Proceedings of the 43rd SIGCSE Technical Symposium on Computer Science Education*. New York, NY: ACM Press.

Grover, S. (2011, April). *Robotics and engineering for middle and high school students to develop computational thinking*. Paper presented at the annual meeting of the American Educational Research Association, New Orleans, LA.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27.

Han Koh, K., Basawapatna, A., Bennett V., & Repenning, A. (2010). Towards the automatic recognition of computational thinking for adaptive visual language learning. In *Proceedings of the 2010 Conference*

on Visual Languages and Human Centric Computing (VL/HCC 2010) (pp. 59–66). Madrid, Spain: IEEE Computer.

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments*, *1*, 1–32.

Henderson, P. B., Cortina, T. J., Hazzan, O., and Wing, J. M. (2007) Computational thinking. In *Proceedings of the 38th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*, 195–196. New York, NY: ACM Press.

Holbert, N. R., & Wilensky, U. (2011, April). *Racing games for exploring kinematics: a computational thinking approach*. Paper presented at the annual meeting of the American Educational Research Association, New Orleans, LA.

Kafai, Y. B., Ching, C. C., & Marshall, S. (1997). Children as designers of educational multimedia software. *Computers & Education*, *29*, 117–126.

Kay, A., & Goldberg, A. (1977). Personal dynamic media. *IEEE Computer*, *10*, 31–41.

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology*, *20*, 362–404.

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*, 429–458.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . .Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*, 32–37.

Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. In *Proceedings of SIGCSE '08*. New York, NY: ACM Press.

Margolis, J., & Fisher, A. (2002). *Unlocking the clubhouse: Women in computing*. Cambridge: MIT Press.

Mercier, E. M., Barron, B., & O'Connor, K. M. (2006). Images of self and others as computer users: The role of gender and experience. *Journal of Computer Assisted Learning*, *22*, 335–348.

Metcalf, J. S., Krajcik, J., & Soloway, E. (2000). Model-It: A design retrospective. In M. J. Jacobson & R. B. Kozma (Eds.), *Innovations in science and mathematics education* (pp. 77–115). Mahwah, NJ: Lawrence Erlbaum.

National Research Council. (2010). *Committee for the Workshops on Computational Thinking: Report of a workshop on the scope and nature of computational thinking*. Washington, DC: National Academies Press.

National Research Council. (2011). *Committee for the Workshops on Computational Thinking: Report of a workshop of pedagogical aspects of computational thinking*. Washington, DC: National Academies Press.

National Research Council. (2012). *A framework for K–12 science education: Practices, crosscutting concepts, and core ideas*. Washington, DC: National Academies Press.

Pane, J. F., Ratanamahatana, C. A., & Myers, B. A. (2001). Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, *54*, 237–264.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism*. (pp. 1–11). Norwood, NJ: Ablex.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*, 137–168.

Pea, R. D., Soloway, E., & Spohrer, J. C. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics*, *9*, 5–30.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*, 265–269. New York, NY: ACM Press.

Royal Society. (2012). Shut down or restart: The way forward for computing in UK schools. Retrieved from http://royalsociety.org/education/policy/computing-in-schools/report/

Simon, B., Chen, T., Lewandowski, G., McCartney, R., & Sanders, K. (2007, March). *Commonsense computing: What students know before we teach (Episode 1: Sorting)*. Paper presented at the Second International Workshop on Computing Education Research, Canterbury, UK.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The Fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*, 215-220. New York, NY: ACM.

Wilson, C., & Guzdial, M. (2010). How to make progress in computing education. *Communications of the ACM*, *53*(5), 35–37.

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). *Running on empty: The failure to teach K-12 computer science in the digital age*. New York, NY: The Association for Computing Machinery and the Computer Science Teachers Association.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–36.

Wing, J. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine*, Spring. Carnegie Mellon University, Pittsburgh. Retrieved from http://link.cs.cmu.edu/article.php?a=600

AUTHORS

SHUCHI GROVER is a doctoral candidate at Stanford University School of Education, 485 Lasuen Mall, Stanford, CA 94305-3096; *shuchig@stanford.edu*. Her research focuses on helping children become computationally literate—studying social, cultural, and cognitive processes that help in developing computational competencies—and on tools and environments that nurture such development.

ROY PEA is the David Jacks Professor of Education & Learning Sciences at Stanford University, School of Education, and Computer Science (Courtesy), and Director of the H-STAR Institute, Wallenberg Hall, 450 Serra Mall, Bldg. 160, Stanford, CA 94305; *roypea@stanford.edu*. His work in the learning sciences focuses on advancing theories, findings, tools, and practices of technology-enhanced learning of complex domains.

Manuscript received April 14, 2012
Revisions received June 13, 2012, and July 19, 2012
Accepted September 6, 2012