

Assessing Algorithmic and Computational Thinking in K-12: Lessons from a Middle School Classroom

Shuchi Grover

Abstract As educators move to introduce computing in K-12 classrooms, the issue of assessing student learning of computational concepts, especially in the context of introductory programming, remains a challenge. Assessments are central if the goal is to help students develop deeper, transferable computational thinking (CT) skills that prepare them for success in future computing experiences. This chapter argues for the need for multiple measures or “systems of assessments” that are complementary, attend to cognitive and noncognitive aspects of learning CT, and contribute to a comprehensive picture of student learning. It describes the multiple forms of assessments designed and empirically studied in *Foundations for Advancing Computational Thinking*, a middle school introductory computing curriculum. These include directed and open-ended programming assignments in Scratch, multiple-choice formative assessments, artifact-based interviews, and summative assessments to measure student learning of algorithmic constructs. The design of unique “preparation for future learning” assessments to measure transfer of CT from block-based to text-based code snippets is also described.

Keywords Computational thinking • Middle school computer science • K-12 computer science education • Deeper learning • Systems of assessments • Algorithmic thinking

“...assessment provides a powerful lever. Assessments shape the public mind, and everything else flows from that”—Schwartz and Arena (2013)

S. Grover (✉)
SRI International, Menlo Park, CA 94025, USA
e-mail: shuchi.grover@sri.com

© Springer International Publishing AG 2017
P.J. Rich, C.B. Hodges (eds.), *Emerging Research, Practice, and Policy on Computational Thinking*, Educational Communications and Technology: Issues and Innovations, DOI 10.1007/978-3-319-52691-1_17

269

Making the Case for Assessments of Computational Thinking

With the bold call for “Computer Science For All” in January 2016, the President of the United States echoed the beliefs of many—that all children from “kindergarten through high school need to learn computer science and be equipped with the computational thinking skills they need to be creators in the digital economy, not just consumers, and to be active citizens in our technology-driven world” (Whitehouse.gov, 2016). This announcement came on the heels of a decade of efforts among researchers, educators, and advocacy groups nationwide working in concert with organizations such as the NSF, ACM CSTA, and [Code.org](#) that involved building a shared understanding around what it means for children to learn computational thinking (CT) and computer science (CS) in formal K-12 educational settings (Grover & Pea, 2013; Wing, 2006). There is now consensus in the national education discourse that computational problem-solving, logical and algorithmic thinking, abstraction, and modeling of real-world phenomena—all undisputed aspects of CT—should be taught not only in CS classrooms but also in the context of STEM and other subjects.

Many of the introductory CT experiences for K-12 settings are being designed around programming using block-based programming environments such as Scratch, Alice, Blockly, MIT App Inventor, and Snap!, among others. These have focused on providing kids with an engaging exposure to the creation of computational artifacts and largely ignored issues of assessment. Consequently, assessments of CT remain underdeveloped and under-researched (Yadav et al., 2015) and this issue has been called out as a key future CS education research imperative (Cooper, Grover, Guzdial, & Simon, 2014). They are conspicuously missing from the early introductory programming curricular offerings rolled out online by entities such as [Code.org](#) and Khan Academy. Without sufficient attention to thoughtful assessment, CT can have little hope of scaling in K-12 education (Grover & Pea, 2013). While the goal of assessments is mostly to measure student learning, it need not necessarily result in awarding student grades, but rather, in providing a useful means to highlight gaps in student understanding that can in turn inform refinements in curriculum and/or pedagogy. Measures that enable educators to assess student learning, both formatively and summatively, need to be created, tested, and validated in various settings with diverse learners.

Furthermore, few efforts, if any, have looked at the issue of transfer of CT skills beyond the immediate curriculum. Transfer of learning is an aspect of assessment that deserves attention since computational experiences at various levels of K-12 aim to serve as bridges to future computational work. New approaches to transfer such as *Preparation for Future Learning* (PFL; Bransford & Schwartz, 1999; Schwartz, Bransford, & Sears, 2005) have shown promise in the context of science and mathematics learning at the secondary level (Chin et al., 2010; Dede, 2009; Schwartz & Martin, 2004). Interventions in CS education could similarly benefit from these emergent ideas in the learning sciences.

This chapter describes the design, use, and study of a comprehensive suite of assessments created for an introductory CS and programming curriculum for middle

school, *Foundations for Advancing Computational Thinking*, or FACT. The goals of the assessments were to assess student learning of algorithmic thinking (mainly) in introductory programming and to evaluate the curriculum that was also designed as part of this research effort. The assessment design and use were influenced and informed by prior research on assessments described in the literature review below.

(Lack of) Assessments of CT in K-12: A Literature Review

Some progress has been made in creating assessments for established high school curricula such as Exploring Computer Science (ECS; Goode, Chapman, & Margolis, 2012) and AP CS Principles (Astrachan et al., 2011) being used nationally. SRI International (2013)'s Principled Assessments of Computational Thinking use evidence-centered design (ECD; Mislevy, Steinberg, & Almond, 2003) to create assessments that support valid inferences about CT practices (Bienkowski, Snow, Rutstein, & Grover, 2015) for the ECS curriculum. Learning in the AP CS Principles course is assessed using through-course assessments involving "performance tasks" in addition to the end-of-course AP Exam comprising multiple-choice questions (College Board, 2014).

The few research efforts that have specifically targeted tackling the issue of CT assessment especially in the context of activities involving programming (e.g., Fields, Searle, Kafai, & Min, 2012; Koh, Nickerson, Basawapatna, & Repenning, 2014; Meerbaum-Salant, Armoni, & Ben-Ari, 2010; Werner, Denner, Campe, & Kawamoto, 2012; Werner, Denner, & Campe, 2015) suggest that assessing the learning of computational concepts and constructs in popular programming environments is a challenge.

Manually checking students' completed projects is a customary form of assessment. However, it is subjective and time-consuming, especially with large student populations, an issue that is being addressed to some extent through tools such as *Dr. Scratch*, a Scratch-specific tool for assessing the complexity of a program (Moreno-León, Robles, & Román-González, 2015). However the existence of computational constructs in the code may not always provide an accurate sense of students' computational competencies (Brennan & Resnick, 2012). Kurland and Pea (1985) reported that students aged 11 and 12 years who had logged more than 50 h of LOGO programming experienced under "discovery learning" conditions were able to write and interpret short, simple programs but had much difficulty on programs involving fundamental programming concepts. In interviews, students revealed many incorrect conceptions about how programs work. These findings clearly point to problems inherent in completed student programs alone as a measure of their understanding of CT concepts. Artifact-based interviews can help provide a more accurate picture of student understanding of their programming projects as shown by Barron, Martin, Roberts, Osipovich, and Ross (2002). Such procedures, while valuable for research and for providing a holistic view of student learning, may not be practical to scale, especially in curricula used in regular school classrooms.

Werner et al. (2012) assessed student learning through a specially designed *Fairy Assessment* created in Alice that required middle school students to code parts of a predesigned program to accomplish specific tasks to demonstrate understanding of algorithmic thinking, abstraction, and code. They found student performance to be the highest on the simplest task which measured comprehension and lowest on the task which measured complex problem-solving skills using debugging. The task, however, was specific to Alice, and rubric-based scoring of student projects was reported to be cumbersome.¹

There is thus a need for assessment instruments that will illuminate student understanding of specific computing concepts and other CT skills such as debugging, code tracing, problem decomposition, and pattern generalization. Black & Wiliam (1998), Glass & Sinha (2013) contend that well-designed multiple-choice assessments can be used to further learners' understanding and provide learners with feedback and explanations in addition to simply testing student understanding. Cooper created a multiple-choice instrument for measuring learning of Alice programming concepts (Moskal, Lurie, & Cooper, 2004), but it has not been used to measure student learning in K-12 education. Lewis et al. (2013) used simple quizzes that get at students' understanding of Scratch blocks. Meerbaum-Salant et al. (2010) used assessments designed to measure understanding of programming concepts in Scratch before, during, and after the intervention.

Other innovative forms of assessment involve the design of gaming environments that teach and assess aspects of CT (e.g., Lee, Ko, & Kwan, 2013) or the design of simulations that assess CT in the context of Science classrooms (e.g., Weintrop et al., 2014). Some limited progress has been made in the development and use of automated tools to assess evidence of CT in programs created in block-based languages such as Scratch, Blockly, and Alice (e.g., Fields, Quirke, Amely, & Maughan, 2016; Grover, et al., 2016; Werner, McDowell, & Denner, 2013). Many of these "learning analytics" efforts are still nascent. Other efforts to assess CT have included studying growth and use of the CS vocabulary and "CT language" (Fletcher & Lu, 2009) to measure CT as children engage in computationally rich activities (Grover, 2011).

Large-scale efforts to roll out introductory computing curricula at the middle school level such as the UK national effort and the Israel Ministry of Education's Science and Technology Excellence Program that includes a national CS curriculum and exam (Zur Bargury, 2012) provide ideas for how CT is being assessed in introductory CS settings internationally. The UK curriculum includes formative assessments, Scratch programming assignments, and a final project of the student's choosing (Scott, 2013). The use of multiple-choice assessments and attendant rubrics to measure learning at scale in the Israeli effort (Zur Bargury, Pärvi, &

¹In an ongoing NSF-funded collaborative research effort, SRI International and Carnegie Mellon University are examining ways of automating assessment using log data from the Fairy Assessment in Alice captured by Denner and Werner. We are employing a combination of computational learning analytics/educational data mining techniques and the ECD framework to study students' programming *process* and automate the assessment of programming tasks such as the Fairy Assessment. Grover, Basu, & Bienkowski (2017) & Grover et al. (2017) provide a glimpse of our work in progress using this computational psychometrics approach.

Lanzberg, 2013) is particularly pertinent to the work described in this chapter. Their nationwide exam in 2012 comprised nine questions (with roughly 30 sub-questions). Arguably such multiple-choice measures are easier to implement on a large scale than open-ended student projects. They use Bloom’s taxonomy to classify the questions and inferences that can be drawn about the appropriate learning level of the associated computing concepts. The research and design on CT assessments in FACT described in this chapter builds on many of these prior efforts.

Research Framework and Methodology

*Foundations for Advancing Computational Thinking:
Curriculum and Assessment Design*

The introductory programming units of FACT, the middle school curriculum at the center of this research, were designed as a structured in-classroom learning experience that leveraged pedagogical ideas for deeper understanding and transfer, and also included various forms of formative and summative assessments. FACT included elements designed to build awareness of computing as a discipline while promoting engagement with foundational computing concepts in introductory programming, mainly, elements of algorithmic flow of control, and CT practices such as code reading, writing pseudo-code, and debugging. The various units in FACT focused on computing in our world, the basics of computational problem-solving, algorithmic thinking, and programming in Scratch (Table 1). The curricular units were deployed as a course on Stanford University’s OpenEdX platform to facilitate a 7-week blended in-class learning experience. The online materials comprised short Khan Academy-style videos ranging between 1 and 5 min in length that led learners through the thinking involved in the construction of computational solutions using the Scratch programming environment. This was inspired by the use of worked examples that have been found to reduce cognitive load initially for novices encountering conceptually challenging tasks in programming (Morrison, Margulieux, & Guzdial, 2015). The videos were interspersed with directed as well as open-ended programming activities to be completed individually or in pairs, peer

Table 1 FACT curriculum unit-level breakdown

Unit 1	Computing is everywhere!/What is CS?
Unit 2	What are algorithms and programs? Precise sequence of instructions in programming
Unit 3	Iterative/repetitive flow of control in a program: loops and iteration
Unit 4	Representation of information (data and variables)
Unit 5	Boolean logic and advanced loops
Unit 6	Selective flow of control in a program: conditional thinking
	Final project (student’s own choice; could be done individually or in pairs)

discussions, and online “quizzes” that used automated grading with feedback, and explanations of solutions. The course ended with an open-ended game design project of choice.

Transfer and PFL for text-based computing contexts (from the block-based Scratch environment) was mediated through expansive framing (Engle et al., (2012) and providing learners opportunities to work with analogous representations (Gentner et al., 2003) of the computational solutions—plain English, pseudo-code, in addition to programs coded in Scratch. Details of the curriculum design and pedagogical underpinnings of FACT are described in Grover, Pea, and Cooper (2015). The curriculum and assessments were refined over two iterations of design-based research (DBR) with students in middle school classrooms (N = 26 in Study#1; N = 28 in Study#2). Student’s learning outcomes, classroom experiences, and feedback on the curriculum and assessments were used to inform revisions. Grover and Pea (2016) detail the DBR process that influenced design decisions. This chapter focuses specifically on the refined assessments used in the second iteration (Study#2) of teaching FACT in a middle school classroom setting.

The “Deeper Learning” Lens

“Deeper learning” (Pellegrino & Hilton, 2013) is increasingly seen as an imperative for helping students develop robust, transferable knowledge and skills for the twenty-first century. The phrase acknowledges the cognitive, intrapersonal, and interpersonal dimensions of learning while also underscoring the need for learners to be able to transfer learning to future contexts. Ideas of deeper learning find resonance in *How People Learn* (Bransford, Brown, & Cocking, 2000)—the seminal treatise that explicated the need for learning environments to be assessment-centered in addition to learner-, knowledge-, and community-centered.

Assessments are designed artifacts of the learning environment aimed at providing feedback to the learner and to the teacher about student understanding. FACT’s assessment design was guided by Barron and Darling-Hammond’s (2008) assertion that robust assessments for meaningful learning must include (1) intellectually ambitious performance assessments that require application of desired concepts and skills in disciplined ways, (2) rubrics that define what constitutes good work, and (3) frequent formative assessments to guide feedback to students and teachers’ instructional decisions. Furthermore, Conley and Darling-Hammond (2013) assert that in addition to assessments that measure key subject matter concepts, assessments for *deeper learning* must measure (1) higher-order cognitive skills as well as skills that support transferable learning and (2) abilities such as collaboration, complex problem-solving, planning, reflection, and communication of these ideas through the use of appropriate vocabulary of the domain in addition to presentation of artifacts to a broader audience.

These assertions point to the need for different measures of learning or “systems of assessments” that are complementary, encourage and reflect deeper learning, and contribute to a comprehensive picture of student learning. *In light of these recommendations for assessments of deeper learning, multiple and innovative measures of assessment were used in FACT to help create a multifaceted picture of student*

learning as described in the sections below. The following research questions were probed through empirical inquiry: (1) What is the variation across learners in learning of algorithmic flow of control (serial execution, looping constructs, and conditional logic) through the FACT curriculum? (2) Does FACT promote an understanding of algorithmic concepts that goes deeper than tool-related syntax details as measured by “preparation for future learning” (PFL) transfer assessments? (3) What is the change in the perception of the discipline of CS among learners as a result of the FACT curriculum? This chapter describes FACT’s “systems of assessments” that measured growth of algorithmic thinking skills, transfer of these skills, as well as noncognitive aspects such as beliefs and perceptions of computing. It also briefly describes the results of empirical research involving their use.

FACT’s Systems of Assessments

FACT’s multifaceted assessments included the following types of measures. They are described in more detail below:

1. Open-ended and directed programming assignments with attendant rubrics that built on the concepts taught
2. Innovative programming exercises inspired by Parson’s puzzles (Parsons and Haden, 2006)
3. Low-stakes high-frequency quizzes² with open-ended, multiple-choice items that were interspersed throughout the course for formative feedback to the learner through feedback and explanations, and to the teacher. These were aimed at assessing understanding of individual concepts and constructs just taught.
4. A summative assessment with multiple-choice items (most of which were reused from the 2012 Israel National Exam)
5. Final project of students’ choosing (to be done in pairs)
6. Final project presentation to the whole class along with individual written student reflections and a “studio” of students’ final projects on Scratch website
7. Student artifact-based interviews around their final projects
8. Block-to-text-based programming PFL assessment
9. Open-ended responses to questions such as “What do computer scientists do?” and “Computing is _____.”
10. Pre-post surveys on student interest and attitudes toward CS and programming
11. A pretest to allow for assessing pre-to-post-FACT learning gains on questions based on Scratch code (inspired by Ericson & McKlin, 2012)

In keeping with its learner-centered philosophy, FACT placed a heavy emphasis on “learning by doing” in the Scratch programming environment. In addition to open-ended time to dabble with programming following example videos (that modeled algorithmic thinking and computational problem-solving in Scratch through worked examples), there were specific assignments with attendant rubrics that built

²FACT’s quizzes and summative assessment have been shared on the assessment platform, Edfinity. <http://edfinity.com/join/9EQE9DT8>.

Table 2 FACT's structured and open-ended Scratch programming assignments

Programming assignments (Scratch/pseudocode)	Algorithmic/CT concepts/constructs
Share a recipe	Serial execution; repetition; selection
(Scratch) Make a life cycle of choice	Serial execution
(Scratch) Draw a spirograph with any polygon	Simple nested loop + creative computing
(Scratch) Create a simple animation	Forever loop
(Scratch) Generic polygon maker	Variables; user input
Look inside Scratch code and explain the text version of code	Algorithms in different forms (analogous representations for deeper learning)
(Scratch) Draw a "Squirrel"	Loops, variables, creative computing
Open-ended project (in pairs): Create a game using "repeat until"	Loops ending with Boolean condition
(Scratch) Maze game	Conditionals; event handlers
(Scratch) Guess my number game	Loops, variables, conditionals, Boolean logic
(Scratch) Final, open-ended project of choice	All CT topics taught in FACT

on the concepts taught in the videos (Table 2). *Rubrics included items for creativity to encourage student self-expression through programming.*

Some programming exercises inspired by Parson's puzzles (Denny, Luxton-Reilly, & Simon, 2008; Parsons & Haden, 2006) involved presenting all the Scratch blocks (in jumbled sequence) required for a program that students needed to snap in correct order.

Low-stakes, high-frequency auto-graded quizzes throughout FACT tested students' understanding of specific CS concepts and constructs, and included explanations, to give learners immediate feedback on their response (examples shown in Figs. 1 and 2). Following these quizzes, some learners would re-watch the video lecture, or try things out in Scratch, thus taking more control of their learning. Many quiz questions were based on small snippets of Scratch or pseudo-code; these were designed to help learners develop familiarity with code tracing—the ability to read/understand code has been found to be positively correlated with code writing (Bornat, 1987; Lopez et al., 2008).

Summative assessments included a posttest conducted online that included multiple-choice and open-ended response items. These items were aimed at assessing learners' CT ability through questions that required code tracing and/or debugging (Figs. 3 and 4). It included six out of nine questions from the 2012 Israel National Exam that are described in Zur Bargury et al. (2013). A final project of learners' choosing to be done with a partner or individually was also part of the summative assessments. Aligning with the social and participatory aspects of learner-centered environments, each student pair also presented the final projects to the whole class on a special "Expo" day and showcased their projects in an online Scratch studio of games, so peers could play (and test) the games and provide peer feedback. Students also individually wrote reflections on their final projects using a



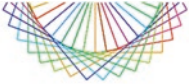
<p>Fill in the blanks below:</p>  <p>What is the value of x at the end of the script: <input type="text" value="12"/> ✓</p> <p>What is the value of y at the end of the script: <input type="text" value="-1"/> ✓</p>	<p>x and y are variables. Consider the following code snippet:</p> <p>x = 5; y = 7;</p> <p>IF (x > 4) AND (y < 6)</p> <p>{</p> <p><some action></p> <p>}</p> <p>Will <some action> within the IF condition above be executed?</p> <p><input type="radio"/> Yes</p> <p><input type="radio"/> No</p>	<p>In the code below, how many times will the sound 'La' be played?</p>  <p><input type="radio"/> 1</p> <p><input type="radio"/> 2</p> <p><input type="radio"/> 3</p> <p><input type="radio"/> 4</p>
--	--	---

Fig. 1 Sample quiz questions used in formative assessments

Simple Loops & Simple Nested Loops
Class Assignment

More Nested Loops
Class Assignment


- Week 3: (More Loops &) Variables
- Week 4: Advanced Looping & Iteration
- Week 5: Boolean Logic
- Week 6: Conditionals
- Week 7: Final Project
- Material from Class2Go course



Staff Debug Info

(W2-D3-Q3A) CHECK ALL THAT APPLY (2 points possible)

Which blocks in the code above would have to be changed in order to generate the (same but smaller) version of the Spirograph seen below?



☐ repeat 36

☐ repeat 4

☒ move 100 steps

☐ turn 10 degrees

EXPLANATION

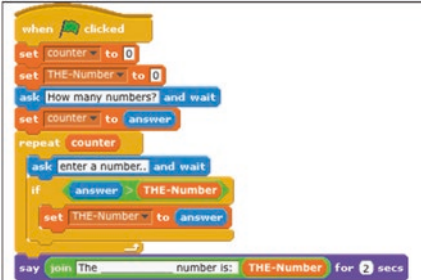
The size of the individual squares in the spirograph is half the size from what it was before, so since the side of the square is created with "move 100", that will change to something lesser (looks like "move 50"). However, the number of squares in the spirograph is the same as before, which means that the same 10 degrees is being turned. And since the same 10 degrees is being turned, the "repeat 36" will also be the same, since the total degrees turned must equal to 360.

Save Final Submit Hide Answers You have used 0 of 1 submissions

Staff Debug Info

Fig. 2 Auto-graded quiz question with explanation on OpenEdX

document adapted from Scott (2013). The final project thus afforded learners the opportunity to problem-solve, code, debug, collaborate, plan, communicate, present, and reflect on their work. Inspired by past research (Barron et al., 2002), “artifact-based interviews” around the final Scratch projects were also conducted individually with each student.



When the code above is executed, what is value of 'THE-Number' at the end of the script for the following inputs after 'counter' is set to 3-

15, 24, 3

This code below does not work. Can you figure out why?
[Note: This program is executed on a stage which has red bricks]

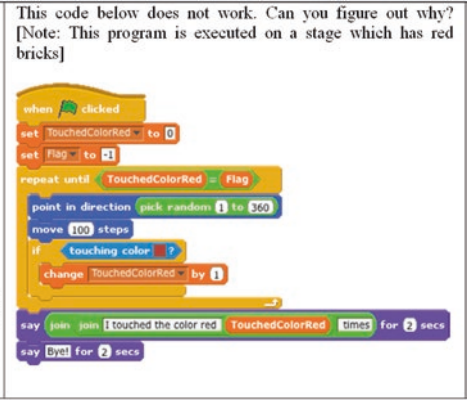


Fig. 3 Sample questions in the CT summative test

Kayla is saving money to buy a new bicycle. On Kayla's birthday, her six uncles gave her money to help her buy the bicycle. Here is a script that includes the cost of the bicycle and the amount of money Kayla received from each of her uncles. The script sums up the amount of money she has received, and the sprite informs her whether she has enough money to buy the bicycle. The script lacks 3 instructions.

(1) Choose one of the following instructions, which must be put in the place where number 1 is marked.

(2) Choose one of the following instructions, which must be put in the place where number 2 is marked.

(2) Choose one of the following instructions, which must be put in the place where number 3 is marked.

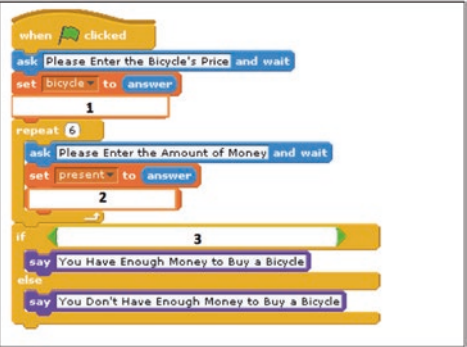


Fig. 4 Sample question from the 2012 Israel National Exam in the CT summative test

FACT’s unique PFL test was designed and administered after the end of the course to specifically assess how well students were able to transfer their computational understanding built in FACT in the context of the block-based Scratch programming environment (and through extensive use of pseudo-code throughout FACT) to the context of a text-based (Java-like) programming language. PFL assessments evaluate how well students learn with new resources or scaffolding that are included as part of the assessment (Schwartz & Martin, 2004; Schwartz et al., 2005). The PFL assessment items were preceded by “new learning” in the form of syntax in a text-based language for constructs that students had encountered in the context of Scratch in FACT. To help learners make connections back to the past learning context (Pea, 1987) and see “the old in new” (Schwartz, Chase, & Bransford, 2012), references are made to the equivalent constructs in Scratch, for example, *PRINT displays things specified in parenthesis to the computer screen one line at a time (like SAY in Scratch)* (Fig. 5). Two different types of syntax were explained, one of a Pascal-like, and the other of a Java-like, language. These were followed by

'<-' (left arrow) is used to assign values to variables. *For example:* `n <- 5` assigns the value 5 to the variable `n`.
 If there are blocks of compound statements (or steps), then the **BEGIN..END** construct is used to delimit (or hold together) those statement blocks (like the yellow blocks for REPEAT and IF blocks in Scratch).
FOR and **WHILE** are loop constructs like REPEAT & REPEAT UNTIL in Scratch
WHILE (*some condition is true*)
 BEGIN
 ... (Execute some commands)
 END
PRINT displays things specified in parenthesis to the computer screen one line at a time (like SAY in Scratch).
Commas are used inside the PRINT command like JOIN in Scratch to combine a text message with a variable

Question #1: *When the code below is executed, what is displayed on the computer screen?*
`PRINT("before loop starts");`
`num <- 0;`
`WHILE (num < 6) DO`
`BEGIN`
`num <- num + 1;`
`PRINT("Loop counter number ", num);`
`END`
`PRINT("after loop ends");`

Fig. 5 Sample PFL question following new syntax specification

questions involving simple code snippets that used the text-based syntax in order to measure students' ability to read and understand the text-based code snippets.

Lastly, since perspectives and practices of the discipline are key ingredients of modern STEM learning, affective aspects such as students' growth in their beliefs and understanding of computing as a discipline and changes in their attendant attitudes toward CS were also assessed through pre-post attitude surveys and responses to the free-response question—"What, in your view, do computer scientists do?"

Participants, Procedures, and Data Measures

In its second iteration, FACT was taught over a seven week period in a public middle school classroom in Northern California. The student sample comprised 28 children from 7th and 8th grade (20 boys and 8 girls; mean age, ~12.3 years) enrolled in a semester-long "Computers" elective class. The class met for 55 min, four times per week. The classroom teacher and researcher were present in the classroom at all times. IRB permission was sought from parents and students prior to the start of the study. An independent researcher assisted with grading. The following data measures were used in the research:

- *Prior experience survey:* This gathered information about students' prior experiences in computational activities (adapted from Barron, 2004).
- *Pre-post CS perceptions survey:* This included the free-response question "What do computer scientists do?"
- *Pre-post CT assessments:* This measured pre- and post-FACT CT skills.

- *Formative quizzes*: These captured student progress and targets of difficulty throughout the course.
- *Scratch projects*: About ten directed and open-ended programming projects through the course (along with rubrics).
- *Final Scratch projects, presentations, and artifact-based student interviews on the final project*.
- “*Preparation of future learning*” (PFL) assessment designed to assess transfer of learning from block-based to a text-based programming language.

Since the current focus is on assessments rather than a description of the curriculum and study procedures (described in detail in Grover et al., 2015), the remainder of the chapter includes results and a brief discussion.

Results

Due to space constraints, qualitative grading of Scratch projects is not discussed here. This section chapter focuses on quantitative scoring of students’ performance on the designed summative and PFL tests, and touches briefly on qualitative analysis of artifact-based interviews after the final project.

The pretest-to-summative test effect size (Cohen’s *d*) was ~2.4, and all learners showed statistically significant learning gains (Tables 3 and 4). Students found ‘serial execution’ or the concept of sequence to be the easiest, followed by conditionals; and loops were the hardest for students to grasp. This was perhaps because most of the questions on loops also involved variable manipulation—a concept that novice learners often struggle with.

Details in Zur Bargury et al. (2013) on scoring and student performance in the student sample of ~4000 middle school students in Israel allowed us to conduct a comparative analysis between our results and those reported in Israel. This analysis revealed comparable performances by our students on the six questions we used

Table 3 Within-student comparison of pretest and summative test scores (Cohen’s *d* = 2.4)

N	Pretest score		Summative score		t-stat	p-value
	Mean	SD	Mean	SD		
28	28.1	21.2	81.6	21.0	-15.5	<0.001

The t-stat and its following p-value come from a t-test to test whether the pre- and post-means are the same

Table 4 Summative test scores breakdown by CS topics

	Mean	Std. dev.
Overall score	81.6	21.2
By CS topic		
Serial execution	91.1	20.7
Conditionals	84.9	20.5
Loops	77.2	26.3
Vocabulary	77.4	22.2

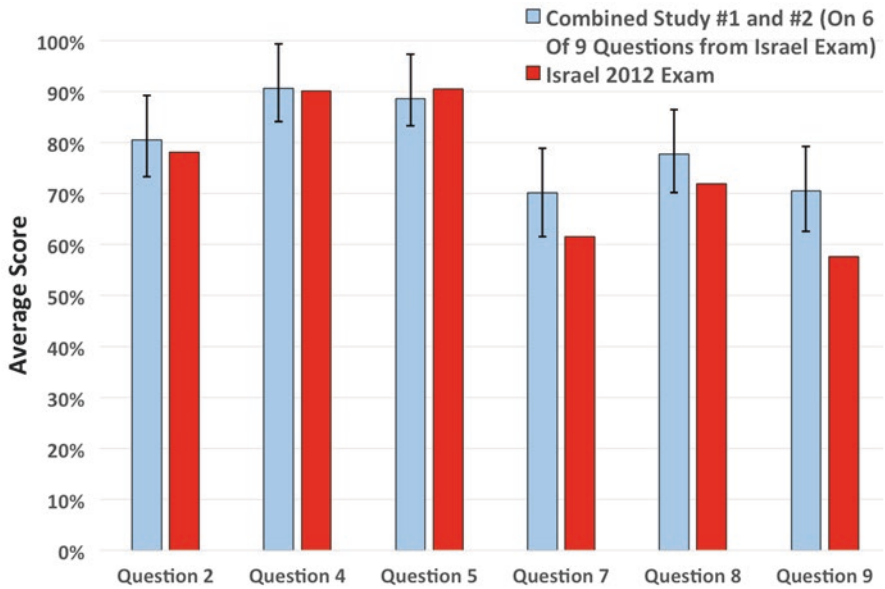


Fig. 6 Student performance post-FACT vs. 2012 Israel National Exam results (N = 4082). **Note:**Error bars represent the margin of error (half-width of a 95% confidence interval)

from the Israel exam (Fig. 6). Interviews around a difficult summative test question (from the Israel National Exam) suggested that those items needed to be refined to improve their validity.

On the PFL test, there was evidence of understanding of algorithmic flow of control in code written in a text-based programming language. Grover, Pea, and Cooper (2014a) provide more details on the PFL questions and their scoring. Regression analyses revealed that ELL students had trouble with the text-heavy PFL test (Grover, Pea, & Cooper, 2016a, b), which may explain the mean score of 65%.

Students’ pre-post responses to the question “What do computer scientists do?” revealed a significant shift from naïve “computer-centric” beliefs of computer scientists as people who engage mostly in fixing, making, studying, or “experimenting” with computers to embrace a view of CS as a creative problem-solving discipline with diverse real-world applications (Fig. 7). More details on this aspect of the research are detailed in Grover, Pea, & Cooper, 2014b.

Final projects were graded based on a rubric inspired by Martin, Walter, and Barron (2009) who used it for grading game projects in AgentSheets. The difference in programming contexts necessitated modifications to make it work for Scratch projects. As in the original rubric, grading criteria were separated into “game design” and “programming” items. It should be noted that the rubric was designed less as a tool for giving the student a grade than to understand students’ application of CT concepts learned. Additionally, individual “artifact-based interviews” were conducted around students’ final projects and guided by questions such as:

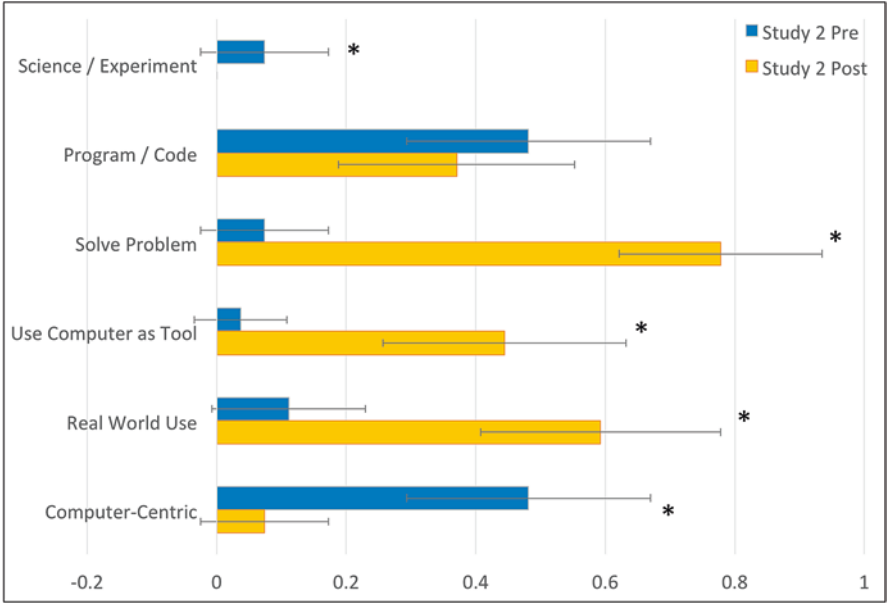


Fig. 7 Pre-post FACT responses to “What do computer scientists do?”

- How did you decide on your project?
- What does it do? How did you do it? (*Student controls the mouse to points things on the screen while talking*)
- What was it like working on this project? [*Fun/challenging/difficult/creative/boring?*]

The qualitative rubric-based grading of the final projects (Fig. 8) and student interviews both revealed that even the students who had low English proficiency and had performed poorly in the summative CT and PFL tests were able to demonstrate high levels of engagement in the final project as well as a reasonable understanding of algorithmic constructs. Two such students were Kevin and Isaac (pseudonyms) who worked together to build a Halloween-themed three-level *Scary Maze* game. The game was not complex, although three levels of increasing difficulty were cleverly incorporated with increasingly narrow passageways. The challenge was to navigate the maze without touching the walls (that narrowed in width from level 1 to level 3). The game elements and visual execution were bare bones, but the students had added interesting effects such as background music during navigation, and a scary scream accompanying a scary face to end the game. Though simple, the game implemented a coherent theme and had an intuitive game play which was very engaging. The programming was not very complex; no variables were used (although they mentioned in the interview that they wanted to add a timer). They implemented different levels by having the scene change when the sprite touched a red door. The game had a bug related to where to position the sprite

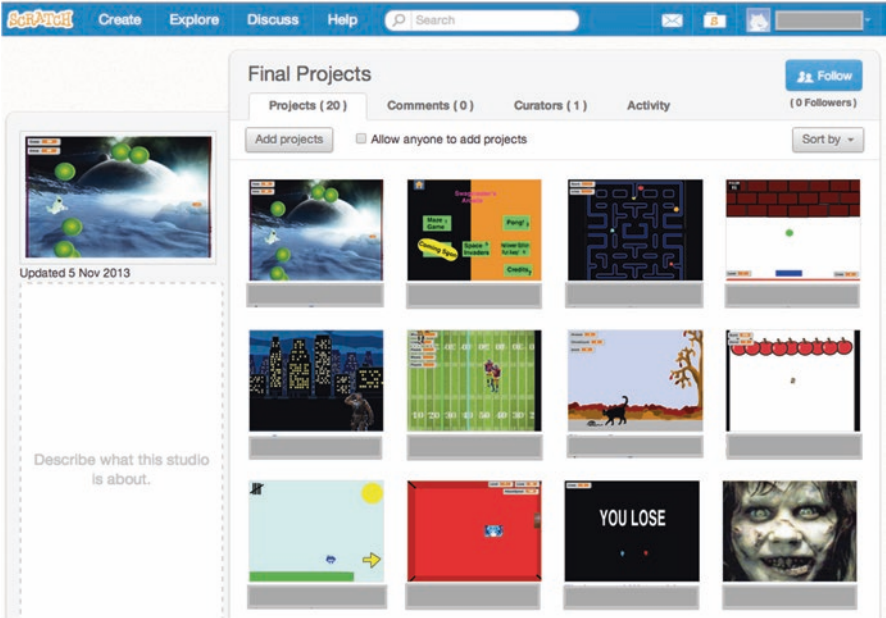


Fig. 8 Studio of students’ final projects on Scratch website

Table 5 Highlights of “artifact-based” interview with Kevin (pseudonym)

Observation/interpretation	Student quote(s) or exchanges with interviewer
Connected “scary” final project idea with event in personal life	“we watched <i>Insidious</i> the day before... and then we got this project assigned and we just decided to do something scary. coz we were traumatized coz of the movie”
Decision to do Maze game, (but with conscious effort to make enhancements)	“We decided to do a maze because, uhm, well, we had an assignment where we were supposed to do a maze... before... We also added like uhm, like levels and stuff like that”
Able to describe what the code did, referred to specific code elements while talking	e.g. “we have in forever our up and arrow keys...” “And each level has a different color red; so when you touch it it’ll take you to the next level”
Was aware of a bug in the code and showed it	Here you see when you touch the black, it’ll start you down here, instead of up there... and w-we don’t know why. We tried to fix it but... we couldn’t fix it, I don’t know what was going on...

In talking through problem causing the bug, he managed to figure that an IF-ELSE would be needed to know which level they were on and reset the position accordingly

at the start of each level. The artifact-based interview with Kevin (excerpts in Table 5) revealed that he was aware of the bug. In the process of talking through the problem during the interview, he was also able to figure how to fix the bug (with the use of a ‘level’ variable and a conditional to test which level the user was on).

Discussion & Implications for Future Work

The various types of measurement designed and used in FACT addressed different facets of understanding of student learning and provided students with opportunities to demonstrate their learning in multiple ways. The formative quizzes were a unique feature of FACT not routinely seen in middle school curricula. They provided useful feedback to students on their learning and understanding as well as to the researchers on aspects of curriculum design based on student performance. Given their formative nature, they were used to inform midcourse curriculum adjustments. For example, students' performance on the 'loops and variables' quiz prompted more time to be dedicated to that topic. In post-course student surveys, while some students indicated that there were "too many" online quizzes, other students found them to be useful for their learning. Some students specifically acknowledged the usefulness of the feedback and explanations they received in response to incorrect answers in quizzes.

The summative assessment test with multiple-choice and open-ended Scratch-based questions as well as the PFL assessment pointed to difficulties that students had with loops with changing variable values. While those topics are traditionally difficult for novices to grasp (Cooper, 2010; du Boulay, 1986; Ebrahimi, 1994; Robins, Rountree, & Rountree, 2003; Spohrer & Soloway, 1986), the results also suggested that other pedagogies may be more productive in introducing children to those CT concepts.³ Results on the PFL test also suggest that they would benefit from redesign. For a balanced set of questions with robust construct validity, the PFL test should assess learners on transfer of individual concepts taught—serial execution, variables, conditionals, and loops—in addition to some questions that involve multiple concepts. Additionally, assessments need to be designed such that they do not disadvantage learners who have difficulty with the English language. The multiple-choice questions in the formative quizzes demonstrated face validity but would benefit from additional reliability studies.

It was apparent that learners benefited from being given opportunities to demonstrate their understanding in multiple ways. This was true for all students regardless of their performance on the formative quizzes and summative CT test. In their artifact-based interviews, each student expressed how much they enjoyed the final project and indicated a personal connection to the final project. In contrast to the decontextualized summative test with questions that involved making sense of Scratch code and answering questions based on them, the final project was a more meaningful form of performance assessment for which Barron and Darrington-Hammond (2008) argue. It embodied learner agency and instilled a sense of personal accomplishment in every student as they presented their projects to their peers. Most importantly, it worked well for all students, even those who performed poorly on the summative test. That said, the value of tests that can be graded easily and objectively to assess understanding of specific CT constructs, however, should not be underemphasized. This is especially pertinent considering the challenges of grading open-ended projects.

³This question is at the heart of my ongoing NSF-funded research project (#1543062) at SRI International being conducted in partnership with San Francisco Unified School District (<https://www.sri.com/work/projects/middle-school-computer-science>).

No single type of assessment alone would have captured the cognitive, affective, and transfer aspects of deeper learning described earlier in this chapter. Each type of assessment served a purpose, and together, they provided a comprehensive view of student learning. Additionally, as this research experience has shown, not all assessments work equally well for all students to demonstrate their learning. This appears to be especially true for those students who are otherwise disadvantaged and/or have poor academic preparation and/or out-of-school experiences that are not intellectually enriching. It is important to keep this in mind as "CS For All" aims to broaden participation and level the playing field in CS for all, with special attention to those groups that have historically been marginalized. This research therefore argues for "systems of assessments" for algorithmic thinking aspects of CT and also presents empirically tested examples for an introductory computing course.

This article addresses a critical gap in the prevalent thinking about the design of introductory computing experiences in formal K-12 CS education. It presents results from the second iteration of a DBR effort. Follow-up work currently underway includes validating an assessment of introductory programming and algorithmic concepts informed by this research. Results from its pilot use in diverse middle school CS classrooms in Northern California have revealed new insights into student misconceptions about loops, variables and expressions (Grover & Basu, 2017). It is hoped that educators and curriculum designers will also be able use the ideas presented in this chapter to inform the design of introductory CS learning experiences at various levels of K-12 education, especially aspects related to assessment of deeper CS learning.

Acknowledgments The research described in this chapter was part of my Ph.D. dissertation at Stanford University. This effort benefited immensely from the support and guidance from my advisors and members of my doctoral committee. I am very grateful for the intellectual contributions of Dr. Roy Pea, Dr. Daniel Schwartz, and Dr. Brigid Barron at the Stanford Graduate School of Education, and Dr. Stephen Cooper at the Department of Computer Science, Stanford University. I would also like to acknowledge the support of the school district, principal, classroom teacher, and students who participated in this research. This project was funded by a grant from the National Science Foundation (#1343227).

References

- Astrachan, O., Barnes, T., Garcia, D. D., Paul, J., Simon, B., & Snyder, L. (2011). CS principles: piloting a new course at national scale. In *Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 397–398). ACM.
- Barron, B. (2004). Learning ecologies for technological fluency: Gender and experience differences. *Journal of Educational Computing Research*, 31(1), 1–36.
- Barron, B., & Daring-Hammond, L. (2008). How can we teach for meaningful learning? In L. Daring-Hammond, B. Barron, P. D. Pearson, A. H. Schoenfeld, E. K. Stage, T. D. Zimmerman, G. N. Cervetti, & J. L. Tilson (Eds.), *Powerful learning: What we know about teaching for understanding*. San Francisco, CA: Jossey-Bass.
- Barron, B., Martin, C., Roberts, E., Osipovich, A., & Ross, M. (2002). Assisting and assessing the development of technological fluencies: Insights from a project-based approach to teaching computer science. In *Proceedings of the conference on computer support for collaborative learning: Foundations for a CSCL community* (pp. 668–669). International Society of the Learning Sciences.

- Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A First Look* (SRI Technical Report) Menlo Park, CA: SRI International. Retrieved from <http://pact.sri.com/resources.html>
- Black, P., & Wiliam, D. (1998). *Inside the black box: Raising standards through classroom assessment*. Granada Learning.
- Bornat, R. (1987). *Programming from first principles*. Englewood Cliffs, NJ: Prentice Hall International.
- Bransford, J. D., & Schwartz, D. L. (1999). Rethinking transfer: A simple proposal with multiple implications. In A. Iran-Nejad & P. D. Pearson (Eds.), *Review of research in education* (Vol. 24, pp. 61–101). Washington, DC: American Educational Research Association.
- Bransford, J. D., Brown, A., & Cocking, R. (2000). *How people learn: Mind, brain, experience and school* (Expanded ed.). Washington, DC: National Academy.
- 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.
- Chin, D. B., Dohmen, I. M., Cheng, B. H., Oppezzo, M. A., Chase, C. C., & Schwartz, D. L. (2010). Preparing students for future learning with Teachable Agents. *Educational Technology Research and Development*, 58(6), 649–669.
- College Board. (2014). *AP computer science principles: Performance assessment*. Retrieved from <https://advancesinap.collegeboard.org/stem/computer-science-principles/course-details>.
- Conley, D. T., & Darling-Hammond, L. (2013). *Creating systems of assessment for deeper learning*. Stanford, CA: Stanford Center for Opportunity Policy in Education.
- Cooper, S. (2010). The design of Alice. *ACM Transactions on Computing Education (TOCE)*, 10(4), 15.
- Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). A future for computing education research. *Communications of the ACM*, 57(11), 34–36.
- Dede, C. (2009). Immersive interfaces for engagement and learning. *Science*, 323(5910), 66–69.
- Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research* (pp. 113–124). ACM.
- du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57–73.
- Engle, R. A., Lam, D. P., Meyer, X. S., & Nix, S. E. (2012). How does expansive framing promote transfer? Several proposed explanations and a research agenda for investigating them. *Educational Psychologist*, 47(3), 215–231.
- Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. *International Journal of Human-Computer Studies*, 41(4), 457–480.
- Ericson, B., & McKlin, T. (2012). Effective and sustainable computing summer camps. In *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 289–294). ACM.
- Fields, D. A., Quirke, L., Amely, J., & Maughan, J. (2016). Combining big data and thick data analyses for understanding youth learning trajectories in a summer coding camp. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 150–155). ACM.
- 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.
- Fletcher, G. H., & Lu, J. J. (2009). Human computing skills: Rethinking the K-12 experience. *Communications of the ACM*, 52(2), 23–25.
- Gentner, D., Loewenstein, J., & Thompson, L. (2003). Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, 95(2), 393–408.
- Glass, A. L., & Sinha, N. (2013). Providing the answers does not improve performance on a college final exam. *Educational Psychology*, 33(1), 87–118.
- Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, 3(2), 47–53.

- Grover, S. (2011). *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.
- Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43.
- Grover, S. & Pea, R. (2016). Designing for deeper learning in a blended computer science course for Middle School: A design-based research approach. In *Proceedings of the 12th international conference of the learning sciences*, Singapore.
- Grover, S. & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In: *Proceedings of the 48th ACM Technical Symposium on Computer Science Education (SIGCSE '17)*. Seattle, WA. ACM.
- Grover, S., Bienkowski, M., Basu, S., Eagle, M., Diana, N. & Stamper, J. (2017). A framework for hypothesis-driven approaches to support data-driven learning analytics In measuring computational thinking in block-based programming. In: *Proceedings of the 7th International Learning Analytics & Knowledge Conference (2017)*. Vancouver, CA. ACM.
- Grover, S. Basu, S., & Bienkowski, M. (2017). Designing programming tasks for measuring computational thinking. In: *Proceedings of the Annual Meeting of the American Educational Research Association*. San Antonio, TX.
- Grover, S., Pea, R., & Cooper, S. (2014b). Remediating misperceptions of computer science among Middle School students. In *Proceedings of the 45th ACM technical symposium on computer science education* (pp. 343–348). ACM.
- Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for Middle School students. *Computer Science Education*, 25(2), 199–237.
- Grover, S., Pea, R., & Cooper, S. (2016a). Factors influencing computer science learning in Middle School. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 552–557). ACM.
- Koh, K. H., Nickerson, H., Basawapatna, A., & Repenning, A. (2014). Early validation of computational thinking pattern analysis. In *Proceedings of the 2014 conference on innovation and technology in computer science education* (pp. 213–218). ACM.
- Kurland, D. M., & Pea, R. D. (1985). Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research*, 1(2), 235–243.
- Lee, M. J., Ko, A. J., & Kwan, I. (2013). In-game assessments increase novice programmers' engagement and level completion speed. In *Proceedings of the ninth annual international ACM conference on international computing education research* (pp. 153–160). ACM.
- Lewis, C. M., et al. (2013). *Online curriculum*. Retrieved from <http://colleenmlewis.com/scratch>.
- Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In: *Proceedings of the Fourth international Workshop on Computing Education Research* (pp. 101–112). ACM.
- Martin, C. K., Walter, S., & Barron, B. (2009). Looking at learning through student designed computer games: A rubric approach with novice programming projects. *Unpublished paper, Stanford University*.
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010). Learning computer science concepts with scratch. In *Proceedings of the sixth international workshop on computing education research (ICER '10)* (pp. 69–76). New York, NY: ACM.
- Mislevy, R. J., Steinberg, L. S., & Almond, R. G. (2003). Focus article: On the structure of educational assessments. *Measurement: Interdisciplinary research and perspectives*, 1(1), 3–62.
- Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *Revista de Educación a Distancia*, 46. doi:10.6018/red/4.
- Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 21–29). ACM.

- Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, 36(1), 75–79.
- Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian conference on computing education* (Vol. 52, pp. 157–163). Australian Computer Society.
- Pea, R. D. (1987). Socializing the knowledge transfer problem. *International Journal of Educational Research*, 11(6), 639–663.
- Pellegrino, J. W., & Hilton, M. L. (Eds.). (2013). *Education for life and work: Developing transferable knowledge and skills in the 21st century*. Washington, DC: National Academies.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172.
- Schwartz, D. L., & Arena, D. (2013). *Measuring what matters most: Choice-based assessments for the digital age*. Cambridge, MA: MIT.
- Schwartz, D. L., & Martin, T. (2004). Inventing to prepare for future learning: The hidden efficiency of encouraging original student production in statistics instruction. *Cognition and Instruction*, 22(2), 129–184.
- Schwartz, D. L., Bransford, J. D., & Sears, D. (2005). Efficiency and innovation in transfer. In J. Mestre (Ed.), *Transfer of learning from a modern multidisciplinary perspective* (pp. 1–51). Greenwich, CT: Information Age.
- Schwartz, D. L., Chase, C. C., & Bransford, J. D. (2012). Resisting overzealous transfer: Coordinating previously successful routines with needs for new learning. *Educational Psychologist*, 47(3), 204–214.
- Scott, J. (2013). The royal society of Edinburgh/British computer society computer science exemption project. In *Proceedings of ITiCSE'13* (pp. 313–315).
- Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, 29(7), 624–632.
- SRI International (2013). Exploring CS curricular mapping. Retrieved from <http://pact.sri.com>.
- Weintrop, D., Beheshti, E., Horn, M. S., Orton, K., Trouille, L., Jona, K., & Wilensky, U. (2014). Interactive assessment tools for computational thinking in High School STEM classrooms. In *Intelligent Technologies for Interactive Entertainment* (pp. 22–25). Springer International Publishing.
- Werner, L., Denner, J., & Campe, S. (2015). Children programming games: a strategy for measuring computational learning. *ACM Transactions on Computing Education (TOCE)*, 14(4), 24.
- 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)* (pp. 215–220). New York, NY: ACM.
- Werner, L., McDowell, C., & Denner, J. (2013). A first step in learning analytics: Pre-processing low-level Alice logging data of Middle School students. *JEDM-Journal of Educational Data Mining*, 5(2), 11–37.
- Whitehouse.gov (2016). Computer science for all. Retrieved from <https://www.whitehouse.gov/the-press-office/2016/01/30/weekly-address-giving-every-student-opportunity-learn-through-computer>.
- Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–36.
- Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., & Clayborn, L. (2015). *Sowing the seeds: A landscape study on assessment in secondary computer science education*. New York, NY: Computer Science Teacher Association.
- Zur Bargury, I. (2012). A new curriculum for junior-high in computer science. In *Proceedings of the 17th ACM annual conference on innovation and technology in computer science education* (pp. 204–208). ACM.
- Zur Bargury, I., Pâr, B., & Lanzberg, D. (2013). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of ITiCSE'13* (pp. 267–272). Canterbury, England, UK.