Computer Science Education, 2015
Vol. 25, No. 2, 199–237, http://dx.doi.org/10.1080/08993408.2015.1033142

Designing for deeper learning in a blended computer science course for middle school students

Shuchi Grover[a,1]*, Roy Pea[b] and Stephen Cooper[c]

[a]*Graduate School of Education, Stanford University, Stanford, CA, USA;* [b]*Graduate School of Education/H-STAR, Stanford University, Stanford, CA, USA;* [c]*Computer Science Department, Stanford University, Stanford, CA, USA*

(*Received 15 November 2014; accepted 16 January 2015*)

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

The focus of this research was to create and test an introductory computer science course for middle school. Titled "Foundations for Advancing Computational Thinking" (FACT), the course aims to prepare and motivate middle school learners for future engagement with algorithmic problem solving. FACT was also piloted as a seven-week course on Stanford's OpenEdX MOOC platform for blended in-class learning. Unique aspects of FACT include balanced pedagogical designs that address the cognitive, interpersonal, and intrapersonal aspects of "deeper learning"; a focus on pedagogical strategies for mediating and assessing for transfer from block-based to text-based programming; curricular materials for remedying misperceptions of computing; and "systems of assessments" (including formative and summative quizzes and tests, directed as well as open-ended programming assignments, and a transfer test) to get a comprehensive picture of students' deeper computational learning. Empirical investigations, accomplished over two iterations of a design-based research effort with students (aged 11–14 years) in a public school, sought to examine student understanding of algorithmic constructs, and how well students transferred this learning from Scratch to text-based languages. Changes in student perceptions of computing as a discipline were measured. Results and mixed-method analyses revealed that students in both studies (1) achieved substantial learning gains in algorithmic thinking skills, (2) were able to transfer their learning from Scratch to a text-based programming context, and (3) achieved significant growth toward a more mature understanding of computing as a discipline. Factor analyses of prior computing experience, multivariate regression analyses, and qualitative analyses of student projects and artifact-based interviews were conducted to better understand the factors affecting learning outcomes. Prior computing experiences (as measured by a pretest) and math ability were found to be strong predictors of learning outcomes.

*Corresponding author. Email: shuchi.grover@sri.com
[1]Present Affiliation: Center for Technology in Learning, SRI International, Menlo Park, CA 94025, USA.

© 2015 Taylor & Francis

Keywords: computational thinking; computer science education; K-12 CS curriculum; middle school computer science; curriculum design; pedagogical content knowledge; design-based research; deeper learning; blended learning; transfer; preparation for future learning; perceptions of computing

1. Introduction

Computational thinking (CT) skills are seen as key for all, not only computer scientists (Wing, 2006, 2011). Although the needs of high school students across the US are being prioritized through pilot courses such as Exploring Computer Science (ECS) and AP CS Principles, there is growing belief that experience with computing must start at an earlier age. Middle school experiences are formative for cognitive and social development in the K-12 schooling journey especially for future engagement with STEM fields (Tai, Liu, Maltese, & Fan, 2006), and should make students amenable to diverse future opportunities as part of their "possible selves" (Markus & Nurius, 1986).

Nations have begun to act on the imperative for training young minds in computing. The UK recently formulated a computing curriculum for all elementary and secondary school students following an ambitious policy charter of the Royal Society (2012) calling for the expansion of computer science (CS) education initiatives. There are also initiatives underway in countries such as New Zealand (Bell, Andreae, & Robins, 2012) and Denmark (Caspersen & Nowack, 2013). Israel is introducing computing in earlier grades through a nationwide middle school curriculum to build the workforce for the Israeli high-tech industry (Zur Bargury, 2012).

Introductory computing courses usually include programming, which is a complex problem-solving activity novices find difficult (for example, du Boulay, 1986; Pea & Kurland, 1984; Robins, Rountree, & Rountree, 2003). Arguably, learners' success in future engagement with computing will depend on how well introductory curricula prepare them in both the cognitive and affective dimensions of computational learning. On the cognitive dimension, students' ability to transfer learning to future computing contexts depends on how deeply they learn foundational disciplinary concepts and constructs. On the affective dimension, learners' attitudes toward computing draws on the interest and awareness such curricula foster among learners regarding this discipline.

If the goal is to reach every student, computing education curricula must enter the classroom in public schools across the US. Unfortunately, most of the activity around computationally rich curricula in middle school has thus far been disparate, mostly relegated to after school and informal settings (Phillips, Cooper, & Stephenson, 2012). US middle schools also face a critical shortage of teachers to teach introductory computing. Making computing

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

courses available online may help accelerate scaling to wider audiences of students and teachers. Although recent months have seen growth of CS curricula at online venues like Khan Academy (http://www.khanacademy.org/) and Code.org, their success for development of deeper, transferable CT skills is yet to be empirically validated, and so far they lack rigorous assessments. The recent explosion of massive open online course – or MOOC – platforms for higher education courses could serve this crucial need in K-12. However, an online course for middle school students would have to be consciously designed for engagement and active learning.

The broad goal of this research was to address the cognitive and affective aspects of learning CS through a structured curriculum for middle school with a well-defined sequence of instruction and activities, combined with formative and summative assessments that also assess transfer of learning from visual, block-based to text-based programming contexts. The curriculum leveraged pedagogical ideas from the learning sciences and computing education research about how children develop conceptual understanding in general, and how they can best develop CT skills, specifically, foundational ideas of algorithmic problem solving.

This paper describes the "design-based research" (Barab & Squire, 2004) effort with data and results over two iterations of the use of a seven-week course titled "Foundations for Advancing Computational Thinking" (FACT) in a blended middle school classroom setting. The first iteration was largely set in a face-to-face classroom context; the second took place in the context of blended in-class learning involving an online version of FACT designed and deployed on Stanford University's OpenEdX MOOC platform. 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 the curriculum 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?

The paper focuses on the design of the learning experience to introduce students to computing and algorithmic thinking and has four parts. A survey of relevant related work presents a backdrop to the research framework and the rationale of the FACT curriculum design. A description of the research methodology follows, including empirical studies, results, and related discussions of findings in the two studies. The Methods section also provides details of the curriculum and assessment pedagogy and design, including the design of the online curriculum and blended classroom experience. The

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

paper ends with a synthesis of the main findings of this research as well as its implications and anticipated future directions.

2. Related work

For ease of organization, the literature review is presented under appropriate subheadings. Although not included in this literature review due to space constraints, this research benefits from a thorough background study of CT and associated ideas of "computational literacy" (DiSessa, 2001) conducted by the authors in Grover and Pea (2013).

2.1. *Misperceptions of computing among K-12 students*

It is particularly alarming that most young teens are completely unaware of CS as a discipline and career choice. Furthermore, this lack of awareness is often coupled with negative attitudes toward CS among students (Carter, 2006; Greening, 1998; Hewner & Guzdial, 2008; Hewner, 2013; Martin, 2004; Mitchell, Purchase, & Hamer, 2009; Yardi & Bruckman, 2007). Martin (2004) claimed "CS has a fundamental image problem" and asserted that students finishing high school have a difficult time seeing themselves as computer scientists since they do not have a clear understanding of what CS is and what a computer scientist does. Carter (2006) recommended fixing "computer science's image" by educating students on how computing is really used in the real world – "for example, for special effects in movies, to improve the quality of life for people with missing limbs, and for allowing communication for people with speech impediments".

Students with an incomplete view of CS are clearly ill equipped to make informed educational choices, and it is not surprising that few students opt to study CS in high school or beyond. Taub, Ben-Ari, and Armoni (2009) surveyed and interviewed middle school students who had completed CS Unplugged activities (Bell, Alexander, Freeman, & Grimley, 2009) over the course of a semester. Their research questions examined students' perceptions and attitudes toward CS and their intent in studying and working in this field. The results were mixed and showed only some growth in students' perceptions of CS. Clearly, K-12 CS curricula need to consciously address misperceptions of computing and make students aware of exemplary societal uses of CS as a creative and engaging discipline with an impact in almost all other domains.

2.2. *Developing CT skills in middle school*

Most contemporary efforts to introduce children to CT through programming in K-12 are centered on tools and environments that boast a "low floor, high ceiling" for ease of use by young programmers. Over the last

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

two decades, visual block-based programming environments such as Scratch, Alice, Game Maker, Kodu, and Greenfoot have gained popularity, as have Web-based simulation authoring tools such as Agentsheets and Agentcubes, and tangible tools such as Arduino and robotics kits. Graphical programming environments are relatively easy to use and allow early experiences to focus on designing and creating, avoiding issues of programming syntax.

Repenning, Webb, and Ioannidou (2010) report on high engagement among middle school children with the scalable game design curriculum that teaches the use of "CT patterns" in the design of video games using Agentsheets.

Denner, Werner, and Ortiz (2012) employed a three-stage "Use-Modify-Create" progression (Lee et al., 2011) to help students engage with CT. Students completed tutorials and worked through a series of self-paced instructional exercises in Alice built to provide scaffolding and "challenges" (Campe, Denner, & Werner, 2013). Students then designed and developed their own games in Alice. Student learning was assessed through a *Fairy Assessment* created in Alice that tested students on algorithmic thinking, and how effectively they made use of abstraction and modeling (Werner, Denner, Campe, & Kawamoto, 2012). This assessment was Alice-based, and grading was subjective and time-consuming – a perennial challenge when assessing student projects. The authors summarize their experience with game design projects in Alice over several research projects by acknowledging that while computer game programming (CGP) in environments such as Alice has proven to be

> a good strategy to attract underrepresented students to computing in middle school and to engage them in programming concepts and systems thinking … CGP does not automatically result in learning, and without some intention on the part of the teacher, it will not result in them learning specific programming concepts. (Campe et al., 2013)

Other research with middle school age children has been conducted using Scratch (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). "Scratchers" are encouraged to become members of the thriving million-strong Scratch community and incentivized to "remix" projects. The success of the open-ended nature of Scratch as a motivating programming environment for both girls and boys is noteworthy (Monroy-Hernández & Resnick, 2008), and the idea of "computational participation" is compelling as a social motivator for learning computing, especially for younger learners. Empirical research, however, suggests that levels of participation are not always suggestive of depth of engagement with advanced computing concepts (Fields, Giang, & Kafai, 2014), and tinkering and remixing as approaches to engaging with CT have resulted in mixed success in formal assessments of CT development. Brennan and Resnick (2012) suggested that although student projects

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

pointed to apparent fluency as evidenced by the existence of computational concepts in the code, probing deeper through interviews revealed that students' descriptions sometimes demonstrated significant conceptual gaps, as they could not explain how their code worked. These studies echo research conducted three decades ago involving children learning LOGO programming. Kurland and Pea (1985) reported that students aged 11 and 12 years who had logged more than 50 h of LOGO programming experience under pure discovery conditions were able to write and interpret short, simple programs but had difficulty on programs involving fundamental programming concepts. In interviews, students revealed many incorrect conceptions of how programs work.

Lewis (2013) provides a structured introduction to Scratch for middle school; however, few curricula encompass a broader introduction to CS as a discipline, and studies rarely include investigations of deeper learning or how well CT learning transfers beyond introductory programming settings to future computing contexts.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

3. Research framework

"Deeper learning" (Pellegrino & Hilton, 2013) resonates with our understanding of learning environments as described in the National Academy of Sciences consensus report, *How People Learn* (Bransford, Brown, & Cocking, 2000). Deeper learning is increasingly seen as an imperative for helping students develop robust, transferable knowledge and skills for the twenty-first century. It demands that students master core academic content, engage in problem solving, work collaboratively, communicate effectively, and learn how to learn (e.g. self-directed learning). Pellegrino and Hilton (2013) outline three broad domains of competence for deeper learning:

- *Cognitive domain*, which includes thinking, reasoning, and disciplinary skills;
- *Intrapersonal domain*, which involves self-management, including the ability to regulate one's behavior and emotions to reach goals; and
- *Interpersonal domain*, which involves expressing information to others, and interpreting information from others.

This approach underscores the need for learners to be able to transfer learning to future learning contexts and describes "deeper learning" as 'the process through which a person becomes capable of taking what was learned in one situation and applying it to new situations – in other words, learning for "transfer"'.

The framework guiding this research recognizes the interconnectedness of pedagogy, disciplinary content, and normative outcomes of deeper learning.

3.1. *Teaching programming requires a pedagogical approach*

Learning to program and construct computational solutions is hard (du Boulay, 1986; Guzdial, 2004; Myers, Ko, & Burnett, 2006; Pea & Kurland, 1984; Reynolds & Caperton, 2011; Robins et al., 2003). By extension, teaching programming is not easy either (Vihavainen, Paksula, & Luukkainen, 2011), as it requires teachers to develop what Shulman (1986) influentially calls "pedagogical content knowledge," a second kind of content knowledge (beyond domain knowledge itself), the

> pedagogical knowledge, which goes beyond knowledge of subject matter per se to the dimension of subject matter knowledge for teaching. I still speak of content knowledge here, but of the particular form of content knowledge that embodies the aspects of content most germane to its teachability. Within the category of pedagogical content knowledge I include, for the most regularly taught topics in one's subject area, the most useful forms of representation of those ideas, the most powerful analogies, illustrations, examples, explanations, and demonstrations – in a word, the ways of representing and formulating the subject that make it comprehensible to others. (p. 9)

Research suggests that learners often struggle with algorithmic concepts, especially if they are left to tinker in programming environments, or if they are not taught these concepts using appropriately supportive pedagogies. Seymour Papert's pioneering efforts in the 1980s concerning children, programming and the development of procedural thinking skills through LOGO programming (Papert, 1980) inspired a large body of such research studies. This previous literature on children and programming (du Boulay, 1986; Pea & Kurland, 1984; Pea, Soloway, & Spohrer, 1987; Perkins & Simmons, 1988; among others) revealed the types of problems children experience on their way to understanding computing and called for a need to study the pedagogy of programming to help children build better cognitive models of foundational concepts of CS. NRC (2011) argued for an application of research in the sciences of learning to design grade- and age-appropriate curricula for CT to maximize its impact on and significance for K-12 students. A curriculum that scaffolds learning experiences with basic algorithmic notions of flow of control in computational problem solving at a younger age is needed to provide that strong foundation for better success in future programming experiences.

There is as yet little consensus what these scaffolded learning experiences look like. Specifically, how can we help *all* learners acquire skills and knowledge of the basics of computational constructs and how these constructs come together in creating a computational solution? How should students be taught to program in a way that they learn the processes of problem-solving inherent to programming? Generally speaking, evidence in past research (e.g. Brown & Campione, 1994; Mayer, 2004; Schwartz & Bransford, 1998) supports the belief that some combination of guided

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

discovery and instruction would be more successful in fostering deep learning for transfer in such an environment than pure discovery and "tinkering". The pedagogical strategies adopted in this research are described in the Section 4.1.2 (*FACT pedagogy*).

3.2. Rigor in assessment of CT skills

Despite the many efforts aimed at CT assessment (e.g. Basawapatna, Koh, Repenning, Webb, & Marshall, 2011; Fields, Searle, Kafai, & Min, 2012; Meerbaum-Salant, Armoni, & Ben-Ari, 2010; Werner et al., 2012), assessing the learning of computational concepts and constructs in these programming environments remains a challenge. To our knowledge, no studies at the K-12 level have explicitly investigated the transfer of CT skills to future learning contexts. New approaches to fostering and assessing transfer such as PFL (Schwartz, Bransford, & Sears, 2005) show promise in the context of science and mathematics learning at the secondary level (Dede, 2009; Schwartz & Martin, 2004). Interventions in CS education could similarly benefit from these approaches.

Although open-ended projects afford students choices in their learning, assessing such projects is subjective and time-consuming, especially with a large student population, and could also provide an inaccurate sense of students' computational competencies (Brennan & Resnick, 2012). "Artifact-based interviews" can help provide a more accurate picture of student understanding of their programming projects (Barron, Martin, Roberts, Osipovich, & Ross, 2002), although these, too, are time-consuming. There is thus a need for more objective formative assessment instruments that can illuminate student understanding of specific computing concepts. Moskal, Lurie, and Cooper (2004) developed a multiple-choice instrument for measuring learning of Alice programming concepts, but it has not been used to measure student learning in K-12 settings.

SRI International (2013) is creating systematic frameworks for assessing CT focusing on assessing CS concepts, inquiry skills, communication and collaboration skills as key elements of CT practices. Their principled assessments of CT are being piloted in high school ECS classrooms (Rutstein, Snow, & Bienkowski, 2014).

Recently developed introductory computing curricula at the elementary and middle school levels outside the US (Lewis & Shah, 2012; Scott, 2013; Zur Bargury, Pârv, & Lanzberg, 2013) provide useful ideas for multiple-choice assessments that make it easier to measure learning, especially in large-scale settings.

Barron and Darling-Hammond (2008) contend that robust assessments for meaningful learning must include intellectually ambitious performance assessments that require application of desired concepts and skills in disciplined ways; rubrics that define what constitutes good work; and frequent

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

formative assessments to guide feedback to students and teachers. Conley and Darling-Hammond (2013) assert that assessments for *deeper learning* must measure higher order cognitive skills, and more importantly, skills that support transferable learning, and abilities such as collaboration, complex problem solving, planning, reflection, and communication of these ideas through use of appropriate vocabulary of the domain in addition to presentation of projects to a broader audience. These assessments are in addition to those that measure key subject matter concepts specifically. This assertion implies the need for multiple measures or "systems of assessments" that are complementary, encourage and reflect deeper learning, and contribute to a comprehensive picture of student learning. No prior efforts, to our knowledge, include such comprehensive assessments.

4. Research methodology

This design-based research effort involved two iterations that studied the use of a seven-week introductory CS course titled FACT in a public middle school classroom setting. The first iteration was in a traditional face-to-face classroom setting, whereas the second iteration involved investigations on a blended model of learning using an online version of FACT designed and created on the OpenEdX MOOC platform. The research encompassed three endeavors: (a) iterative design of curriculum and assessments; (b) engineering a blended learning experience using an online version of FACT; and (c) empirical investigations around FACT use in a classroom to answer the research questions outlined earlier.

4.1. Curriculum design: content, pedagogy and assessments

4.1.1. FACT content

The seven-week FACT curriculum (Table 1) was inspired by the 36-week long ECS high school curriculum (Goode et al., 2013) and included topics that were considered by the authors as foundational and engaging for a shorter course for middle school. While CT encompasses several elements (Grover & Pea, 2013), FACT focused largely on algorithmic problem solving in addition to broader notions of computing as a discipline. The

Table 1. FACT curriculum unit-level breakdown.

Unit 1	Computing is everywhere!/what is CS?
Unit 2	What are algorithms and programs? Computational solution as a precise sequence of instructions
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)

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

curriculum design effort was guided by goals for deeper learning, attending to the development of cognitive abilities through mastery of disciplinary learning for transfer, in addition to interpersonal and intrapersonal abilities.

FACT expressly engaged students' narrow perceptions of CS to help them see computing in a new light. This was done with a view to "expansively frame" (Engle, Lam, Meyer, & Nix, 2012) the curriculum so learners would see a broader relevance of their learning. To give students a sense for exemplary societal uses of CS, Unit 1 "Computing is Everywhere!" was designed so students would see examples of computing being used for varied purposes and in diverse fields. Publicly available videos were created/found. These included videos such as "M.I.T. Computer Program Reveals Invisible Motion in Video" and "Untangling the hairy physics of Rapunzel" that exemplified engaging innovations in computing and demonstrated the use of computing in contexts novel for most middle school students. Sebastian Thrun's TED talk on Google's driverless car was an example of a project to tackle the problem of fatalities in automobile accidents. The videos for this unit are publicly available (http://bit.ly/CS-rocks; http://stanford.edu/~shuchig/Computing-vignettes.html).

FACT also devoted attention to the following additional curricular goals:

- Gaining a familiarity with CS vocabulary to communicate computational ideas more effectively.
- Learning core *practices* of programming in the context of the introductory content described above, including
  - Task decomposition and construction of a programmatic solution.
  - Using abstraction.
  - Iterative development of programs.
  - Debugging and testing of programs.
- Interpersonal and intrapersonal aspects of computational learning:
  - Working with others.
  - Learning from and supporting one another's learning.
  - Showcasing and sharing one's work with the classroom community.
  - Reflection on, and revision of, projects.

Table 2 lists the FACT curricular topics by "main ideas". These ideas were used to outline more detailed learning goals, in accordance with the Understanding by Design framework (Wiggins & McTighe, 2005).

4.1.2. FACT pedagogy

The pedagogical design exercise hinged on the belief that no single pedagogical approach would likely accommodate the diverse goals of deeper learning. A minimally guided discovery approach often makes for high learner engagement and agency but misses out on helping students develop

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table 2. FACT content goals by main ideas and learning goals.

Main idea	Learning goals
Main idea #1: computing is everywhere!	• Students will appreciate the pervasiveness of computing in our lives today • Students will see computing as a discipline with wide applicability in many subjects and fields of human endeavor
Main idea #2: what is computer science?	• Students will learn that computer science is a problem-solving discipline with applications in the real world, and computer scientists are creative problem solvers who work on problems to make our lives better and easier using computation
Main idea(s) #3: computational solutions: algorithms; programs; pseudo-code; deeper computing concepts vs. superficial syntactical details	• Students will learn what an algorithm is, and be able to write out a detailed sequence of instructions to perform a task • Students will learn that algorithms usually contain three essential elements: serially executed instructions, repeated actions, and decision or selection of alternate courses of action (based on some condition) • Students will learn the difference between a program and an algorithm • Students will learn different ways to represent an algorithmic solution • Students will describe an algorithm as pseudo-code in "semi-English" as an in-between representation between algorithms and programs • Students will understand that the same algorithm or pseudo-code can subsequently be coded in different programming languages • Students will understand that though syntactical details of programming languages differ, the deeper concepts of computation and algorithmic flow of control is the same
Main idea(s) #4: algorithmic flow of control (sequence, serial execution, loops, conditions)	• Students will learn how programs are executed sequentially • Students will learn how simple loops work • Students will learn algorithmic flow of control – how commands are executed in sequence even when there are loops, except that the commands within a loop are repeated (in sequence) and then the program proceeds to the commands after the loop • Students will learn how to create different pathways in programs using conditional statements
Main idea #5: variables	• Students will learn what data are, and how it is used in a program • Students will learn how variables are an abstraction or representation of data in the program • Students will learn how to make code work for a certain set of valid inputs given by the user • Students will learn how variables are created, used, assigned values, and updated • Students will learn how variable values change within loops. • Student will learn what initialization is and why it is important

(*Continued*)

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table 2. (*Continued*).

Main idea #6: boolean logic	• Students will learn the idea of controlling loops and conditionals using Boolean tests • Students will learn how AND, OR, and NOT operators work in computing contexts
Main idea #7: problem decomposition	• Students will learn the importance of planning before programming • Students will learn the need for breaking down problems into smaller manageable tasks • Students will learn the usefulness of modular code and how abstraction helps with modularity of code
Main idea #8: abstraction	• Students will learn computational solutions are abstractions; and that these abstractions can be represented in different ways • Students will learn that variables used in programs are abstractions of data in the real world • Students will be given a brief overview of functional abstraction and its use in making computational solutions modular and manageable

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

robust mental models of concepts (Mayer, 2004), with the consequence that deeper, transferable understanding of core disciplinary ideas may suffer. Similarly, a pure "instructionist" (Papert, 1991) approach that does not engage students' prior knowledge or experiences does not lend weight to learner agency, thus failing on many dimensions valued in learner-centered environments. It appears then that instead of adopting a monocultural approach to pedagogy, a course that aims for a balanced set of curricular goals for deeper learning would benefit from espousing a balanced approach to curriculum, pedagogy, and assessment design, too. The goal for FACT curriculum design and research, therefore, was to tackle this effort as a design balance problem and *engineer a balanced curriculum with a balanced pedagogy and assessments* drawn from diverse influences to guide this effort.

FACT pedagogy borrowed from inquiry-based approaches of ECS – Engage, Explore, Explain, Elaborate and Evaluate, pedagogical ideas of scaffolding (Pea, 2004) and cognitive apprenticeship (Brown, Collins, & Newman, 1989). It involved working (and thinking aloud) through examples to model solutions to computational problems in a manner that revealed the underlying structure of the problem, and the process of composing the solution in pseudo-code or in Scratch, the programming environment used in FACT. Code reading and tracing were modeled throughout. Often students were expected to think about and discuss programming scenarios or problems *before* the solution was modeled. Academic language and computing vocabulary were used during this scaffolding process.

Learners were encouraged to use pseudo-code as a step before coding. This paraphrasing of problems and solutions before programming has been

found to be beneficial for algorithmic thinking in general (Bornat, 1987; Fidge & Teague, 2009; Fincher, 1999; Mayer, 1989). Experiences with pseudo-code are instrumental in building a deeper understanding of computing constructs. Seeing algorithmic solutions expressed in various formats and using synonymous/analogical terms for the same concept (for example, REPEAT, WHILE, and FOR, to express the idea of loops and repetition), helps learners go beyond surface features of the syntax and programming language to see the deeper structure of how sequences, loops, and conditionals, for example, characterize most solutions expressed in any imperative language. This design draws on prior cognitive science studies using analogous representations in learning for understanding deeper structures in problem-solving situations (Gentner, Loewenstein, & Thompson, 2003; Schwartz, Chase, Oppezzo, & Chin, 2011). It is contended that guiding students to draw analogies between different formalisms can foster deep and abstract fundamental concepts of a domain. Dann, Cosgrove, Slater, Culyba, and Cooper (2012) and Touretzky, Marghitu, Ludi, Bernstein, and Ni (2013) have adopted this approach for introductory computing contexts. Building deeper understandings is part of mediating transfer and "PFL" – one of the central learning goals of FACT. Finally, FACT emphasized "learning by doing" through a mix of directed assignments and open-ended projects (Barron & Darling-Hammond, 2008) in Scratch, which also served as assessments.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

4.1.3. FACT's systems of assessments

Well-designed multiple-choice assessments can be used to further learners' understanding (Glass & Sinha, 2013) and to provide learners with feedback and explanations rather than simply testing (Black & William, 1998). Low-stakes, high-frequency quizzes throughout FACT tested students' understanding of specific CS concepts and constructs, and provided learners with immediate feedback on their understanding. Inspired by the multiple-choice assessments mentioned earlier, quizzes often involved small snippets of Scratch or pseudo-code on which questions were based (Table 3). These assessments were designed to help learners develop familiarity with code tracing – the ability to read/understand code (Bornat, 1987; Lister, Fidge, & Teague, 2009; Lopez, Whalley, Robbins, & Lister, 2008). Inspired by Parson's puzzles (Denny, Luxton-Reilly, & Simon, 2008; Parsons & Haden, 2006), some quiz questions also involved presenting jumbled blocks in Scratch required for a program and having students snap them in correct order.

FACT placed a heavy emphasis on "learning by doing" involving programming in Scratch. In addition to open-ended time to dabble in Scratch, there were specific assignments with attendant rubrics that built on the concepts taught (Table 4). Rubrics included items for creativity, encouraging students to add their own distinctive elements.

Table 3. Sample quiz questions used in formative assessments.

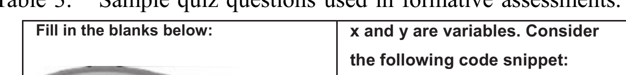
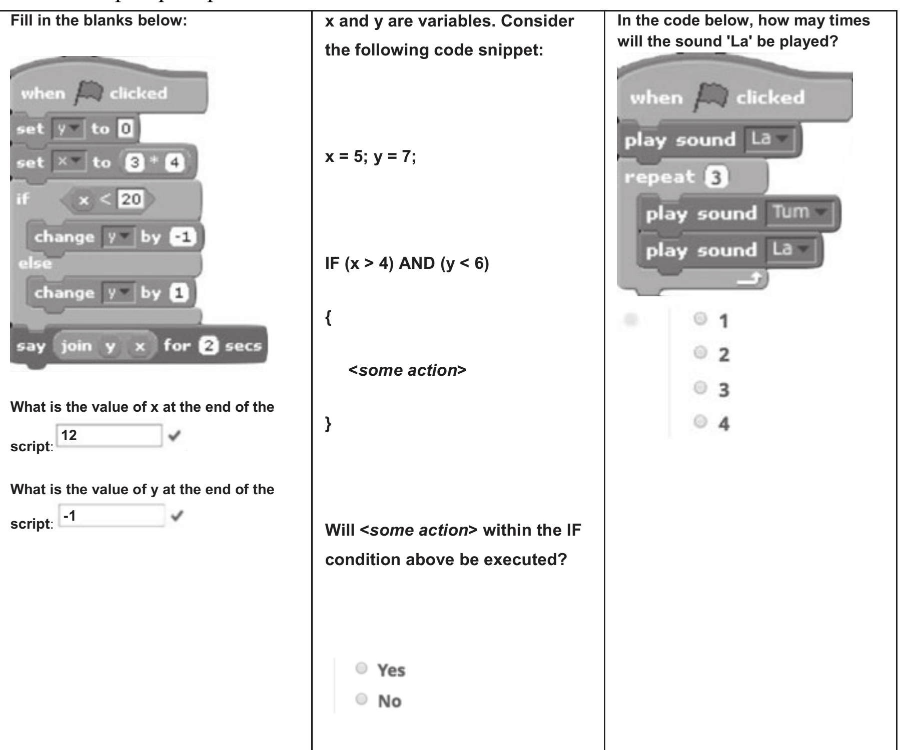

Table 4. Sample structured and open-ended Scratch programming assignments in FACT.

Programming assignments (Scratch/pseudo-code)	Algorithmic/CT concepts/constructs
Share a recipe	Sequence of instructions – serial execution; repetition; selection
(Scratch) make a life cycle of choice to use in a 6th grade science class	Serial execution
(Scratch) draw a spirograph from a polygon of choice	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 “Squiral”	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*

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table 5. Samples questions created for the CT summative test.

when [flag] clicked
set counter to 0
set THE-Number to 0
ask How many numbers? and wait
set counter to answer
repeat counter
  ask enter a number.. and wait
  if answer > THE-Number
    set THE-Number to answer
say join The number is: THE-Number for 2 secs
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]
	when [flag] clicked
	set TouchedColorRed to 0
	set Flag to -1
	repeat until TouchedColorRed = Flag
	  point in direction pick random 1 to 360
	  move 100 steps
	  if touching color ?
	    change TouchedColorRed by 1
	say join join I touched the color red TouchedColorRed times for 2 secs
	say Bye! for 2 secs

Summative assessments involved a final posttest online that included multiple-choice and open-ended questions to assess learners' CT ability through questions that required code tracing and/or debugging (Table 5). They also included six questions from the 2012 Israel National Exam (Zur Bargury et al., 2013). A final project of learner's choosing to be done with a partner was also included as part of summative assessments. Aligned with the desired social and participatory aspects of learning environments, the final projects were presented during a whole-class "Expo", and showcased in an online studio of games on the Scratch website so classmates could play with, and provide feedback on, their peers' games. The different tasks involved creating the project, testing it, demonstrating it to the entire class, documenting and writing reflections on it, and finally, playing with each others' games in the online studio. These activities afforded learners the opportunity to problem solve, collaborate, plan, communicate, present, and reflect on their work in a document adapted from the *Starting from Scratch* curriculum (Scott, 2013). Inspired by past research (Barron et al., 2002), "artifact-based interviews" explaining their Scratch projects were conducted.

Another unique aspect of FACT was the transfer or PFL test that was specially designed and administered after the end of the course to assess how well students were able to transfer their computational understanding built in the block-based Scratch programming environment to questions in Pascal/Java-like code borrowed from past AP exams.

Lastly, since perspectives and practices of the discipline are considered essential ingredients of a CS curriculum, affective aspects such as students' growth in their understanding of computing as a discipline and changes in their attendant attitudes toward CS were also assessed through pre- and post-responses to the free-response question – "What do computer scientists do?"

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

4.2. Participants and procedures

Empirical studies were conducted in a public middle school classroom in Northern California. Like most middle schools in the US, it caters to students (aged ~11–14) in grades 6, 7, and 8. Two iterations (henceforth referred to as "Study1" and "Study2") were conducted with two different cohorts in the "Computers" elective mixed-age class (with 7th and 8th grade students) that met four days a week for 55 min periods (Table 6).

In Study1, the course was taught face-to-face by the lead researcher and author. One unit (out of the six units) was piloted on the online MOOC platform with face-to-face lessons replaced by video ones and interspersed with automated quizzes and other activities to be done individually or collaboratively (usually in pairs). Extensive feedback was sought from learners on their experiences with the online unit as a precursor to creating online materials for the entire course for Study2.

Study2 was conducted in the same classroom with a new cohort and used an online version of FACT on OpenEdX with roughly 60 videos (1–5 min in length), ten quizzes that were interspersed through the course, and learning sequences for blended learning comprising a mix of activities to be done individually or collaboratively. Several refinements were also made to the curriculum based on experiences, and teacher and student feedback in Study1. The improvements to Study1 incorporated in the blended version of FACT as part of attempts to refine the curriculum are detailed in Table A1 in the Appendix 1. The classroom teacher, without a background in CS or programming, was present in the classroom at all times assisting with classroom management and "learning right alongside the students" during both studies.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

4.2.1. *Designing the blended classroom experience in Study2*

The affordances of the OpenEdX platform were leveraged to design a "learning sequence" for each day. Each week's materials were divided into four parts that roughly mapped to the four days' worth of work (Figure A1). To leverage the social affordances of the classroom in this blended setting, and to encourage active learning, the learning sequence for each day (Figure 1) was planned to include a balanced mix of individual learning, activities that may/may not involve a partner, active learning by doing, working on quizzes (Figure A2 shows one such example), responding to

Table 6. Student samples in Study1 and Study2.

		Count by Gender		Count by Grade		Count in Sp. Programs	
Study	Mean age	Male	Female	Grade 7	Grade 8	ELL	Special Ed.
1	12.9	21	5	15	11	4	2
2	12.3	20	8	16	12	3	1

Day 5	Day 6	Day 7	Day 8
Comps-Dumb+powerful (3m24s)	Characteristics of Algorithms (3m 43s)	Vid #20 Intro to loops - 4m	Initialization - 1min
Think & Write-What can computers not do (5 mins)	Quiz 3 questions (3 mins)	Quiz 7 questions (5 mins)	Quiz - 4 ques (5 min)
types of instructions (1m53s)	Sequence; Repetition; Selection (7 mins)	Make a polygon in Scratch (5 mins)	Scratch Assignment - 4 part problem
Patterns in human instructions (1m55s)	Programs & programming Languages (4m 09s)	Intro Spirograph activity (5 mins)	Optional Scratch Assignment - olympic rings
Algorithms (3 m 16s)	Quiz (Algorithms & programs) 5 ques; 5 min	Nested Loops vid - 2m51s	Kids helping other kids catch-up
Share a Recipe	First Program Example (4m 38s)	Quiz - 5 mins	
	Scratch Assignment: Science Life Cycle	Rings Demo video - 4m02s	
		Scratch Assignment:Make a Spirograph	
		Optional Quiz (2 questions)	

Figure 1. Learning sequences planned for the four days of Week 2 on OpenEdX FACT.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

thought questions on the discussion board, and working on programming projects with others or simply helping others on theirs.

The modular nature of the OpenEdX platform design allowed various types of “elements” to be added to a course page. This platform feature aided the use of contextual discussion prompts below instructional videos, or a Scratch window “iframe”[1] (http://scratch.mit.edu) right below the video so students could try out the ideas discussed in the video. Qualtrics surveys were similarly “iframe”-d to obtain student feedback and responses to open-ended “thought questions” which students completed at the end of a video lecture. In Figure 1, different shades of gray depict the various types of learning activity. From lightest to darkest are videos, videos with activities below them, thought questions, Scratch assignments, quizzes, and lastly, extra (optional) assignments.

4.3. Data measures

The main and additional data measures (Table 7) included

- Pre- and posttests of “computational knowledge” (using code snippets with questions).

Table 7. Main instruments for capturing relevant data measures.

Instrument	Pre-intervention	Post-intervention	Source(s)
Computational knowledge test	✓	✓	Several questions from Ericson and McKlin (2012); Meerbaum-Salant et al. (2010); Zur Bargury et al. (2013)
PFL test		✓	Designed using AP CS questions (Inspired by Schwartz and Martin (2004))
Prior experience survey (programming experience and technology fluency)	✓		Adapted from Barron (2004)
CS perceptions survey	✓	✓	Ericson and McKlin (2012)

- Prior computing experience pre-course survey; pre- and post-surveys to measure CS interest and attitudes, and demographic information including age, gender, and academic placement.
- PFL test to assess transfer to text-based programming.
- CS Perceptions survey including pre-and post- response to: "What do computer scientists do?" Comparative data were also gathered from college students.

Additional data measures in Study2 included (1) Automated quizzes on OpenEdX and (2) Final projects and individual artifact-based interviews about the project.

5. Analysis and results

In order to answer the three research questions, data were analyzed separately for each study; however, factor analyses on prior experience variables and multivariate regressions to determine which variables predicted outcome measures of interest were conducted on the combined sample from the two studies. The "perceptions of computing" responses were analyzed using a mix of qualitative coding and quantitative methods.

A comparative analysis was also conducted using the six questions of the posttest that were employed in the National CS Exam administered to about 4000 middle school students in Israel (Zur Bargury et al., 2013).

5.1. Research question #1: learning of algorithmic constructs

The effect size (Cohen's *d*) as measured by pretest and posttest of computational learning was roughly 2.4 in both studies, and all learners in both studies, regardless of pretest performance, showed significant gains from pre- to posttest as measured by matched pairs *t*-tests. The average learning gain for students in Study2 was statistically higher (as measured by *t*-tests) than for those in Study1 (Table 8).

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table 8. Comparison of CT test scores (out of 100) between Study1 and Study2.

	Study1		Study2					
	N	Mean (SD)	*N*	Mean (SD)	*t*	$p < \|t\|$	*z*	$p < \|z\|$
Pretest	24	36.33 (18.19)	28	28.06 (21.18)	1.5	0.14	1.76	0.08
Posttest	26	78.58 (17.08)	28	81.60 (21.24)	−0.58	0.56	−1.26	0.21
Learning gain	24	43.08 (12.17)	28	53.07 (18.34)	−2.34	0.02*	−2.46	0.01*

Notes: Shapiro–Wilk tests for normality revealed that although the pretest score had a normal distribution, the posttest score did not. Hence, the non-parametric Mann–Whitney (Wilcoxon) rank-sum test was also conducted to test the difference between the pretest and posttest.
*$p < 0.05$.

On CT constructs taught, students found serial execution easiest followed by conditionals; loops were the hardest for students to grasp (Table 9). This was perhaps because most of the questions on loops also involved variable manipulation – a concept which students found difficult to learn in both studies.

5.2. *Research question #2: PFL transfer test*

On the PFL test, although students scored an average of about 65% (Table 10), there was evidence that students were generally able to understand algorithmic flow of control in snippets of code written in text-based programming languages. The syntax for the (Pascal and Java) programming languages was explained and provided to the students before they attempted the questions. The two researchers – the lead author and a doctoral student in education with knowledge of programming, scored the PFL test. Inter-rater reliability (Cohen's Kappa) for our independent grading was 82.1%. Table A2 provides details on the PFL test and scoring.

Table 9. Posttest scores by CT topics for Study1 and Study2.

Variable	Study1 Mean (SD)	Study2 Mean (SD)	*t*-Stat	*p*	*z*-Score	*p*
Overall	78.6 (17.1)	81.6 (21.2)	−0.6	0.56	−1.3	0.21
By CS topic						
Serial execution	97.4 (13.1)	91.1 (20.7)	1.4	0.18	1.6	0.12
Conditionals	84.5 (19.0)	84.9 (20.5)	−0.1	0.94	−0.4	0.72
Loops	74.1 (21.9)	77.2 (26.3)	−0.5	0.64	−1.1	0.29

Table 10. Comparison of PFL test scores (out of 100) for Study1 and Study2.

	Study1		Study2					
	N	Mean (SD)	*N*	Mean (SD)	*t*	$p < \|t\|$	*z*	$p < \|z\|$
PFL test	25	63.37 (28.86)	27	65.07 (26.47)	−0.22	0.82	−0.08	0.93

Note: *p*-values for the PFL test come from a *t*-test of equality of means across samples with unequal variance.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Some errors were the result of badly worded questions. For example, in the question, "How many numbers will be processed by the program below?" (Question #4, Table A2), some students counted the number of variables that the code was using rather than the count of numbers processed by the loop. Several were common "off-by-1" loop errors. Students struggled on the question with the FOR loop. Prior research suggests that FOR loops are problematic for older novice programmers too (Robins et al., 2003) due to the complex "behind-the-scenes" action involved with incrementing the loop variable. It is therefore not surprising that fewer than 50% of the students tackled that question correctly. It is also noteworthy that most of the questions on the PFL test involved loops and variables – topics students had the most difficulty with in the computational learning posttest. Prior literature contends that skills mastery in the original context is essential for transfer (Kurland, Pea, Clement, & Mawby, 1986).

5.3. Research question #3: perceptions of computing

Responses to the "perceptions of computing" question were analyzed through quantitative analysis of qualitative data coded by two human coders (Cohen's Kappa measure of inter-rater reliability was 79.1%).[2] The frequencies of occurrences of the main coding categories (as a percentage of total occurrences across all codes) in pre- and post-responses are presented in Figure 2. A comparative data-set was gathered by asking the same question

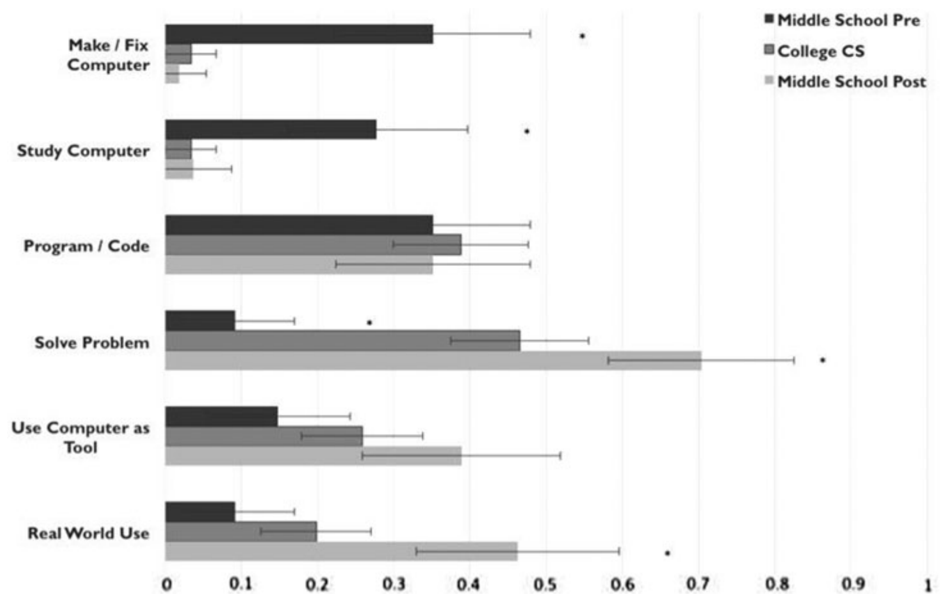

Figure 2. "What do computer scientists do?"– College students' responses ($N = 168$) vs. pre- and post-responses from Study1 and Study2 (combined).

of 168 undergraduate students in a major university in the same geographical area who had recently completed an introductory computer science (CS1) course that introduced students to Java and software engineering principles including object-oriented design, decomposition, encapsulation, abstraction, and testing. Figure 2 (which also shows this comparison data with college students) reveals a significant shift from naïve "computer-centric" notions of computer scientists ("make/fix/study computer") to a more sophisticated understanding of CS as a problem-solving discipline that uses the computer as a tool to solve real-world problems in many diverse fields.

5.4. Other results

Additional findings revealed in data analyses that cannot presently be further discussed:

- The mean quiz score across the 10 automated quizzes for the 28 students in Study2 was 73.14 (SD = 12.68). These scores were significantly positively correlated with posttest performance. Data on automated quizzes from Study2 aggregated by the OpenEdX platform revealed that students found variables hard to grasp.
- The comparative analysis with the results from the Israel national exam (Zur Bargury et al., 2013) revealed that the only significant difference was on one question in which our students achieved a higher score (Figure A3).
- Some gender differences were observed with girls performing better than boys, and on average, logging into the online course after school a significantly greater number of times than boys (as measured by *t*-tests); however, the small number of females in the sample precluded the drawing of deeper conclusions.
- Regression analyses revealed math performance was a positively correlated predictor for performance in the pretest as well as the posttest (even when controlling for the pretest). This may also have been due to the fact that some questions required knowledge and use of math.
- The curriculum helped all students irrespective of prior experience as measured by the self-report survey. Prior programming experience as measured by the pretest was found to positively predict performance on both the posttest as well as the PFL test. Regressing posttest performance on prior experience factors (that resulted from factor analyses on prior experience survey data) revealed that among students who did not have prior programming experience, those with experience in media creation generally did better than those that did not, and those that engaged only in online gaming and video watching (to the exclusion of programming or media creation activities), did worse.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

5.5. *Qualitative analysis*

Although the research design relied mainly on quantitative pre- and post-analyses for measuring growth of student learning, regression analyses suggested that the text-heavy nature of the posttest and PFL test appeared to disadvantage English Language Learners (ELL). Consequently, a qualitative analysis of final projects and interviews was conducted for four students who scored in the lowest quartile in the posttest.

5.5.1 Student interviews

About a month after the final projects had ended, all students in Study2 were interviewed individually. These interviews lasted between 12 and 17 min. The main goal was to have each student explain his or her final project and its inner workings, and attempt Question 9 employed from Israeli nationwide exam (Figure 3) by talking aloud through the solution. Most of the lower-performing students had trouble with some subparts of that question, and it was our aim to understand through the interview process what aspect(s) of the question caused them difficulty. This part of the interview was an attempt to better understand the broader goals and designs for assessments with a view to future improvements in assessment design.

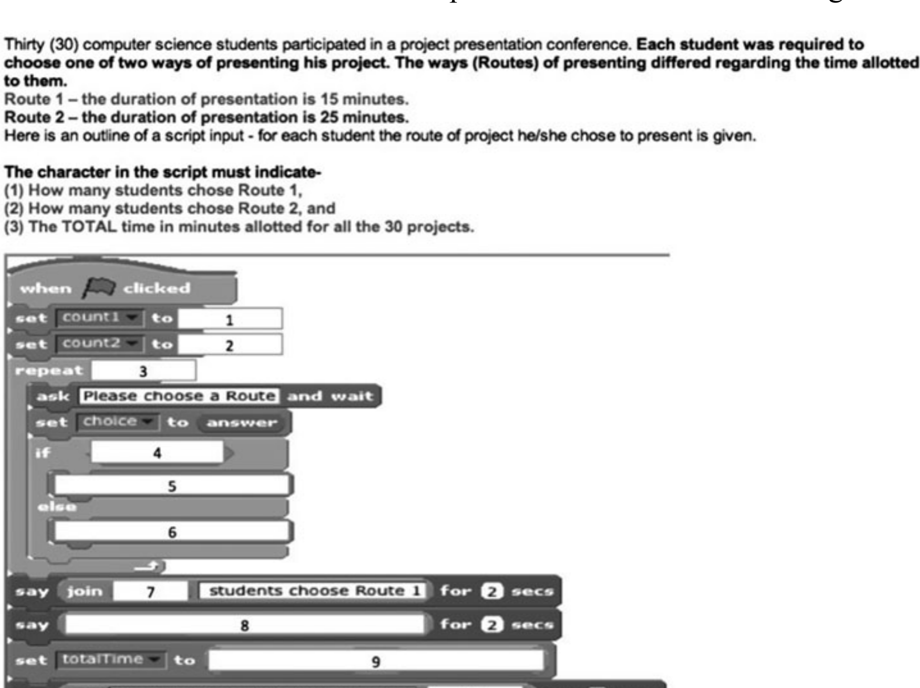

Figure 3. Question 9 from 2012 Israeli nationwide used in the posttest and interviews.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

The interviews were guided by the following broad questions that were posed to each student (although not always in the exact same sequence):

- How did you decide on your project?
- What does it do? How did you do it? (*Student controls the mouse while talking*).
- What was it like working on this project?
- Overall how did you feel about this course?
- How does it feel to now know more about CS?
- Thinking back to on the course, what was the hardest part?
- Lastly, bring up the posttest question (Israel Exam Question 9 as presented in Figure 3) and have the student work through the problem on the screen.

The interviews were conducted in a small video recording anteroom that was attached to the computer laboratory in which the *Computers* class met. The interview was recorded using Camtasia software, which allows for screen-capture in addition to video capture via webcam, and audio through the built-in laptop microphone. Although all the students in the class were interviewed (barring those for whom parental consent to capture audio/video recordings was not secured), interviews were transcribed and analyzed only for the four students featured in this qualitative analysis.

The interviews were transcribed by another doctoral education researcher, who with the lead author independently annotated the interviews with the four students, Kevin, Isaac, Alex, and Lucas (pseudonyms) for evidence of understanding of the project, areas the learner found challenging, scaffolding which the learner needed to figure things out in their project code, and description of project experience. These analyses revealed that students' final projects evinced high levels of engagement and showed evidence of understanding of program construction and algorithmic constructs that the posttest was unable to capture. Excerpts from one such sample interview are provided in Table A4.

6. Discussion

6.1. Conceptual learning of algorithmic constructs

Based on the quantitative analyses of the results for the two studies, and the qualitative analysis of final projects and interviews for a subset of learners reported above, it appears that the FACT course helped all learners build a substantial understanding of basic algorithmic flow of control in computational solutions. As these results also reveal, Study2 in a blended in-classroom learning setting using the online version of FACT worked just as well, if not better, than Study1, an intervention that used the face-to-face version of the FACT curriculum, and hence is therefore deemed a "successful"

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

online intervention per Means, Toyama, Murphy, Bakia, and Jones (2010). This difference could be reasonably attributed to refinements inspired by design-based research.

The qualitative analysis of the final projects and interviews revealed that students showed higher levels of engagement and more evidence of understanding of program construction and algorithmic constructs than was indicated by their posttest performance.

6.2. Transfer of learning to a text-based programming context

The answer to the second research question, "Does the curriculum promote an understanding of algorithmic concepts that goes deeper than tool-related syntax details as measured by PFL assessments?" is a cautiously optimistic "yes." The results of the PFL test were promising, and in answer to the research questions that guided this study, the students were found to be able to transfer many ideas from the seven-week curriculum using Scratch to a text-based programming context. They were broadly able to interpret programs in the new context of text-based languages, although the mechanics of some constructs (e.g. "For" loops) were difficult to grasp. The PFL test was, in all likelihood, too difficult compared to the posttest. Not only was it testing learners on new learning and programming in a context completely alien to most students, it also focused mainly on loops and variables, which were the hardest concepts for learners to grasp as evidenced by the breakdown of Scratch posttest performance by constructs. For a balanced set of questions with robust construct validity, the PFL test should test learners on the individual concepts taught – serial execution, variables, conditionals, and loops – in addition to some advanced snippets of code that incorporate all concepts. Additionally, the PFL test should be redesigned such that it does not disadvantage learners who have difficulty with the English language.

6.3. Affective aspects of computing

In terms of interests and attitudes toward computing, neither study made a significant difference to learners' incoming motivational markers. Such a ceiling effect has been observed in similar contexts where learners self-select into the intervention, and come in with high levels of interest and motivation. That said, student reactions to the videos in the "Computing is Everywhere!" unit in both studies evidenced a growing awareness of what computer scientists do, and the meaning of computer science as a discipline and its application in our world. Affective indicators also suggested that this curriculum helped them see CS in a positive, new light, and sparked curiosity to learn more. Student perceptions shifted significantly from narrow and naïve views of CS as a science that involves experiments or a study of computers in order to build, fix, or improve them to a sophisticated understanding of CS as a

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

problem-solving discipline where computers are used as tools to make people's lives easier, and in creative and engaging ways, too.

6.4. The role of the classroom teacher in online FACT

In these studies, the regular classroom teacher in charge of teaching the *Computers* elective did not have a CS background, and as such did not have a deep understanding of computational concepts and programming. She was also unfamiliar with Scratch. The researcher therefore undertook some of the responsibilities that would normally fall on the teacher, especially for grading Scratch assignments and projects, and providing feedback to students, as well as administrative assistance on tracking assignment submissions and aspects of online course navigation, such as explaining to students what to do next in the course sequence, or describing the mechanics of uploading assignments. The classroom teacher provided assistance in ensuring students stayed on task and followed the required course sequence. Sometimes she would work through the materials on her own in parallel with the class in an effort to build her own understanding of computing and skills in programming. Other classroom management matters that needed attention often interrupted her sessions in online FACT. She reported using the course on her own with her next cohort of students. In general, however, the role of the teacher with online FACT curriculum could vary depending on the comfort level the teacher enjoys with introductory CS content.

6.5. Methodological limitations

Statistical power was an issue due to the relatively small sample sizes in the two studies, although this was mitigated to some extent by combining the datasets for the correlational and regression analyses as well as the factor analysis (as measured by the KMO scores for sampling adequacy). In general, the study would have benefited from a better experimental setup with a control condition where student learning in a different but comparable setting could be tested using the same set of assessments. This was accomplished to a small extent through the preliminary exploratory studies where students learning Scratch in a similar school in the same district were given questions on the PFL test in addition to questions on vocabulary and computational learning. However, this was not a control group in a strict sense, and the surveys and questions were administered with the purpose of testing the instruments and validating the preliminary hypotheses that few middle school computing experiences pay attention to deeper understanding, transfer, perceptions of computing, or vocabulary of the discipline. Although a strict control group is difficult to implement in a real-world situation, students working with other middle school CS curricula (such as those recently released by Khan Academy and Code.org) could potentially constitute such control groups in future studies.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

The surveys before and after the course as well as some of the questions on the quizzes, and many of the questions on the posttest and PFL test required students to read large amounts of text. This was a challenge for ELL students and learners not very fluent with English. Such students sometimes require the help of aides in core subject classrooms – help that this classroom did not have because it was an elective course. Some of the questions that had appeared in the nationwide exam in Israel (Question 9, for example, as described above) were found to be problematic. This finding has prompted a redesign in Israel as well (I. Zur Bargury, personal communication, 6 December 2013), and some PFL test questions were open to multiple interpretations. Clearly, the assessments need to go through more rigorous processes of validation, which involves testing the assessments in different settings.

There were threats to external validity as well. The findings cannot be broadly generalized since the studies were conducted in only one public school situated in a generally upper-middle class community in Northern California. The sample sizes had a gender imbalance. Lastly, the studies were conducted in an elective class, indicating a self-selection bias among learners who came into the course with generally high interest and motivation.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

7. Conclusion and scholarly implications

This research makes several unique contributions. It is perhaps the first online introductory middle school curriculum that has been empirically shown to result in learning gains (albeit only in the context in which it was studied) to foster deeper transferable CT skills (focused on algorithmic thinking) in middle school students. It serves as an example of MOOC course design for K-12 learners that employs design-based research effectively to refine a curriculum aiming to foster active learning by balancing the individual and social, and online and offline, in a blended classroom setting. The explicit attention devoted to teaching and assessing for PFL and transfer, and to remedying misperceptions of computing as a discipline through an engaging corpus of publicly available videos are also unique aspects of this research, which could be gainfully employed in other pedagogies for CT. The idea of using "systems of assessments" and the creation of a corpus of questions for assessing PFL and computational learning (in Scratch and pseudo-code) are innovations of this research that contributes to the broader computing education research enterprise.

Perhaps the most salient implications of this research concern the foundational premise of this effort – that students as young as 12 and 13 can and should be taught CT in addition to being made aware of computer science and its applications in other disciplines. The middle school years are crucial for identity building as well as cognitive development for the analytical reasoning required of STEM disciplines. Learners must be taught with a view to deeper learning so that they master the core disciplinary ideas in ways

that attend to transfer of that learning to productive future efforts. Consciously attending to creating bridges to future learning, and assessing for transfer through specially designed PFL assessments are unique contributions that FACT makes to the space of disciplinary-based research devoted to computing education.

The findings from these empirical studies also provide some crucial pointers for K-8 introductory computing curricula. For example, which specific topics need extra attention based on novices' targets of difficulty as revealed in this research? Curricula and pedagogies need to pay special attention when introducing learners to variables and loops, for example. The strong correlation between computational learning and Math ability (as measured by the California state test score in this research) is a critical finding, as well. As schools across the US enact the "computing for all" mantra, children should develop strong mathematical skills for success in computing. Given the synergies between these domains of thinking and problem solving, perhaps there is also a case for teaching Math through computing and vice versa.

Nonetheless, this research has demographic and research design limitations as described above. Future directions involve overcoming these limitations to create a robust curriculum shown to work in diverse settings and with broader audiences of middle school students and teachers across the US. Other work in the future also involves creating a version of FACT for teacher professional development, perhaps as a MOOC for an online, geographically dispersed teacher audience.

As von Hippel and Tyre (1995) and Engeström (2009) rightly point out about the essential nature of design-based research, any curricular innovation is a continuous process, and any particular version of it is simply a point along the journey. This is true of FACT as one such design-based research enterprise. Although FACT currently embodies a learning theory-informed, pedagogically robust design of a curriculum for computing in middle school, it still has room for improvement. This research represents only the first two iterations of what should be seen as an ongoing systematic design-based research effort in diverse settings and with broader audiences of middle school students and teachers.

Acknowledgments

This paper is based on a PhD dissertation completed by Grover under the direction of Pea and Cooper and draws on Grover's prior doctoral work under the supervision of Pea. The work benefitted from the guidance of the members of the dissertation committee: Profs Daniel Schwartz, Brigid Barron, and Mehran Sahami. The author would like to acknowledge Stanford's Office of the Vice Provost for Online Learning and members of the Stanford OpenEdX team for their support in creating and running the online course on OpenEdX. The author is grateful for suggestions from Prof. Mark Guzdial from Georgia Institute of Technology's College of Computing. Lastly, the author would like to acknowledge the support of the school district, principal, classroom teacher, and students who participated (but remain anonymous) in this dissertation research.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by funding from the National Science Foundation [NSF-0835854, NSF-1343227].

Notes

1. An IFrame (Inline Frame) is an HTML document embedded inside another HTML document on a website. The IFrame HTML element is often used to insert content from another source, into a web page. In this instance, it was used to embed a Scratch window below an instructional video.
2. The coding and inter-rater reliability calculations were typically completed together for datasets from both Study1 and Study2 to ensure consistency in the coding process across the two studies. All the responses were initially open-coded. The coding categories were arrived at in discussion between the 2 coders. A single response could be coded for the presence of more than one category. One set of responses was coded independently first (specifically, the pre-course responses in Study2), and then the coders met to discuss differences and interpretations of the codes. Finally, the next three sets (pre- and post-responses of Study1, post-responses of Study2) were coded. There was some confusion concerning the "Use of a computer as a tool" category, in which one coder interpreted any mention of creation of technology products as evidence of the learner talking about a computer scientist using the "computer as a tool", as opposed to the interpretation that the learner's response more explicitly reflected the notion that the computer scientist *used* the computer to make people's lives easier or convenient or better (to distinguish from the naïve "computer-centric" view of the computer only being "studied" or "fixed" or "improved"). It is our belief that this confusion was never completely resolved well, and resulted in significant differences for that coding category.

References

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *Journal of the Learning Sciences, 13*(1), 1–14.

Barron, B. (2004). Learning ecologies for technological fluency: Gender and experience differences. *Journal of Educational Computing Research, 31*(1), 1–36.

Barron, B., & Darling-Hammond, L. (2008). How can we teach for meaningful learning? In L. Darling-Hammond, B. Barron, P. D. Pearson, A. H. Schoenfeld, E. K. Stage, T. D. Zimmerman, … J. L. Tilson (Eds.), *Powerful learning: What we know about teaching for understanding* (pp. 11–70). 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). Boulder, CO: International Society of the Learning Sciences.

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

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology, 13*, 20–29.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Bell, T., Andreae, P., & Robins, A. (2012). Computer science in NZ high schools: The first year of the new standards. In *Proceedings of the 43rd ACM technical symposium on computer science education*. Raleigh, NC.

Black, P., & Wiliam, D. (1998). Assessment and classroom learning. *Assessment in Education, 5*, 7–74.

Bornat, R. (1987). *Programming from first principles*. Upper Saddle River, NJ: Prentice Hall International.

Bransford, J. D., Brown, A., & Cocking, R. (Eds.). (2000). *How people learn: Mind, brain, experience and school, expanded edition*. Washington, DC: National Academy Press.

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.

Brown, A. L., & Campione, J. C. (1994). Guided discovery in a community of learners. In K. McGilly (Ed.), *Classroom lessons: Integrating cognitive theory and classroom practice* (pp. 229–272). Cambridge: MIT Press.

Brown, J. S., Collins, A., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In *Knowing, learning, and instruction: Essays in honor of Robert Glaser* (p. 487). Hillsdale, NJ: Lawrence Erlbaum Associates.

Campe, S., Denner, J., & Werner, L. (2013). Intentional computing: Getting the results you want from game programming classes. *Journal of Computing Teachers*.

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of SIGCSE* (pp. 27–31). ACM, Houston, TX, USA.

Caspersen, M. E., & Nowack, P. (2013). Computational thinking and practice: A generic approach to computing in Danish high schools. In *Proceedings of the fifteenth Australasian computing education conference* (Vol. 136, pp. 137–143). Sydney: Australian Computer Society.

Conley, D. T., & Darling-Hammond, L. (2013). *Creating systems of assessment for deeper learning*. Retrieved from http://scee.groupsite.com/uploads/files/x/000/09e/76f/creating-systems-assessment-deeper-learning.pdf

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 141–146). New York, NY: ACM.

Dede, C. (2009). Immersive interfaces for engagement and learning. *Science, 323*, 66–69.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*, 240–249.

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). New York, NY: ACM.

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

du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research, 2*, 57–73.

Engeström, Y. (2009). The future of activity theory: A rough draft. In A. Sannino, H. Daniels, & K. D. Gutierrez (Eds.), *Learning and expanding with activity theory* (pp. 303–328). New York, NY: Cambridge University Press.

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*, 215–231.

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). New York, NY: ACM.

Fidge, C., & Teague, D. (2009). Losing their marbles: Syntax-free programming for assessing problem-solving skills. In *Proceedings of the eleventh Australasian conference on computing education-volume 95* (pp. 75–82). Sydney: Australian Computer Society.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Fields, D. A., Giang, M., & Kafai, Y. (2014). Programming in the wild: Trends in youth computational participation in the online Scratch community. In *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 2–11). New York, NY: ACM.
Fields, D. A., Searle, K. A., Kafai, Y. B., & Min, H. S. (2012). *Debuggems to assess student learning in e-textiles*. Proceedings of the 43rd SIGCSE technical symposium on computer science education. ACM Press, New York, NY.
Fincher, S. (1999). What are we doing when we teach programming? In *Frontiers in Education Conference, 1999* (Vol. 1, pp. 12A4-1–12A4-5). New York, NY: IEEE.
Gentner, D., Loewenstein, J., & Thompson, L. (2003). Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology, 95*, 393–408.
Glass, A. L., & Sinha, N. (2013). Providing the answers does not improve performance on a college final exam. *Educational Psychology, 33*, 87–118.
Goode, J., Chapman, G., Margolis, J., Landa, J., Ullah, T., Watkins, D., & Stephenson, C. (2013). *Exploring computer science*. Retrieved from http://www.exploringcs.org/curriculum
Greening, T. (1998). Computer science: Through the eyes of potential students. In *Proceedings of the 3rd Australasian conference on computer science education* (pp. 145–154). ACM, The University of Queensland, Australia.
Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*, 38–43.
Guzdial, M. (2004). Programming environments for novices. *Computer Science Education Research, 2004*, 127–154.
Hewner, M. (2013). Undergraduate conceptions of the field of computer science. In *Proceedings of the ninth annual international ACM conference on international computing education research* (pp. 107–114). New York, NY: ACM.
Hewner, M., & Guzdial, M. (2008). Attitudes about computing in postsecondary graduates. In *Proceeding of ICER 2008* (pp. 71–78). ACM, Sydney, Australia.
Kurland, D. M., & Pea, R. D. (1985). Children's mental models of recursive logo programs. *Journal of Educational Computing Research, 1*, 235–243.
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.
Lewis, C. M., & Shah, N. (2012). Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 57–62). New York, NY: ACM.
Lewis, C. M. (2013). *Online curriculum*. Retrieved from http://colleenmlewis.com/Scratch
Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In *ACM SIGCSE Bulletin* (Vol. 41(3), pp. 161–165). New York, NY: ACM.
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). New York, NY: ACM.
Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). *Programming by choice: Urban youth learning programming with Scratch*. Proceedings of SIGCSE '08. ACM Press, New York, NY.
Markus, H., & Nurius, P. (1986). Possible selves. *American Psychologist, 41*, 954–969.
Martin, C. D. (2004). *Draw a computer scientist*. Working group reports on innovation and technology in computer science education (ITiCSE-WGR'04) (pp. 11–12). New York, NY: ACM.
Mayer, R. E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 129–159). Hillsdale, NJ: Lawrence Erlbaum.
Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? *American Psychologist, 59*, 14–19.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Means, B., Toyama, Y., Murphy, R., Bakia, M., & Jones, K. (2010). *Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies*. Washington, DC: US Department of Education.
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.
Mitchell, A., Purchase, H. C., & Hamer, J. (2009). Computing science: What do pupils think? In *Proceedings of the 14th annual conference on innovation and technology in computer science education (ITiCSE'09)* (p. 353). New York, NY: ACM.
Monroy-Hernández, A., & Resnick, M. (2008). Feature: Empowering kids to create and share programmable media. *Interactions, 15*, 50–53.
Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin, 36*, 75–79.
Myers, B. A., Ko, A. J., & Burnett, M. M. (2006). Invited research overview: End-user programming. In *CHI'06 extended abstracts on Human factors in computing systems* (pp. 75–80). New York, NY: ACM.
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.
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.
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). Sydney: Australian Computer Society.
Pea, R. D. (2004). The social and technological dimensions of scaffolding and related theoretical concepts for learning, education, and human activity. *Journal of the Learning Sciences, 13*, 423–451.
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. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics, 9*, 5–30.
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 Press.
Perkins, D. N., & Simmons, R. (1988). Patterns of misunderstanding: An integrative model for science, math, and programming. *Review of Educational Research, 58*, 303–326.
Phillips, P., Cooper, S., & Stephenson, C. (2012). *Computer science K-8: Building a strong foundation*. CSTA voice special issue, 13–14.
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)* (pp. 265–269). New York, NY: ACM Press.
Reynolds, R., & Caperton, I. H. (2011). Contrasts in student engagement, meaning-making, dislikes, and challenges in a discovery-based program of game design learning. *Educational Technology Research and Development, 59*, 267–289.
Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*, 137–172.
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/
Rutstein, D., Snow, E., & Bienkowski, M. (2014). *Computational thinking practices: Analyzing and modeling a critical domain in computer science education*. Paper presented at the 2014 annual meeting of the American Educational Research Association (AERA), Philadelphia, PA.
Schwartz, D. L., & Bransford, J. D. (1998). A time for telling. *Cognition and Instruction, 16*, 475–5223.

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., Oppezzo, M. A., & Chin, D. B. (2011). Practicing versus inventing with contrasting cases: The effects of telling first on learning and transfer. *Journal of Educational Psychology, 103*, 759.
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*, 129–184.
Scott, J. (2013). The royal society of Edinburgh/British computer society computer science exemplification project. In *Proceedings of ITiCSE'13* (pp. 313–315). New York, NY: ACM.
Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher, 15*, 4–14.
SRI International. (2013). *Exploring CS curricular mapping*. Retrieved from http://pact.sri.com/?page_id=1380
Tai, R., Liu, C. Q., Maltese, A. V., & Fan, X. (2006). Career choice: Enhanced: Planning early for careers in science. *Science, 312*, 1143–1144.
Taub, R., Ben-Ari, M., & Armoni, M. (2009). The effect of CS unplugged on middle-school students' views of CS. In *SIGCSE Bulletin* (Vol. 41(3), pp. 99–103). New York, NY: ACM.
Touretzky, D. S., Marghitu, D., Ludi, S., Bernstein, D., & Ni, L. (2013). Accelerating K-12 computational thinking using scaffolding, staging, and abstraction. In *Proceeding of the 44th ACM technical symposium on computer science education* (pp. 609–614). New York, NY: ACM.
Vihavainen, A., Paksula, M., & Luukkainen, M. (2011). Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 93–98). New York, NY: ACM.
von Hippel, E., & Tyre, M. J. (1995). How learning by doing is done: Problem identification in novel process equipment. *Research Policy, 24*(1), 1–12.
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.
Wiggins, G. P., & McTighe, J. (2005). *Understanding by design. Association for supervision and curriculum development*. Alexandria, VA: Association for Supervision and Curriculum Development.
Wing, J. (2006). Computational thinking. *Communications of the ACM, 49*, 33–36.
Wing, J. (2011). Research notebook: Computational thinking – What and why? *The Link Magazine*, Spring. Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://link.cs.cmu.edu/article.php?a=600
Yardi, S., & Bruckman, A. (2007). What is computing? Bridging the gap between teenagers' perceptions and graduate students' experiences. In *Proceedings of ICER 2007* (pp. 39–50). New York, NY: ACM.
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). New York, NY: ACM.
Zur-Bargury, I., Pârv, B., & Lanzberg, D. (2013). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (pp. 267–272). New York, NY: ACM.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Appendix 1. Additional Figures and Tables

Figure A1. Course organized into four learning sequences for each day of the week.

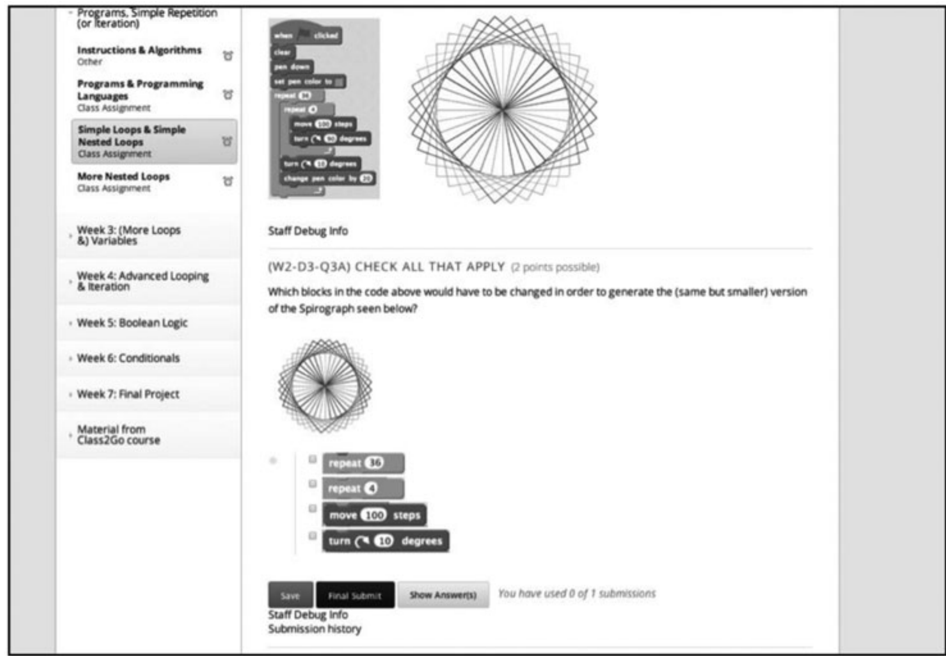

Figure A2. Quiz question requiring students to understand code and answer questions.

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

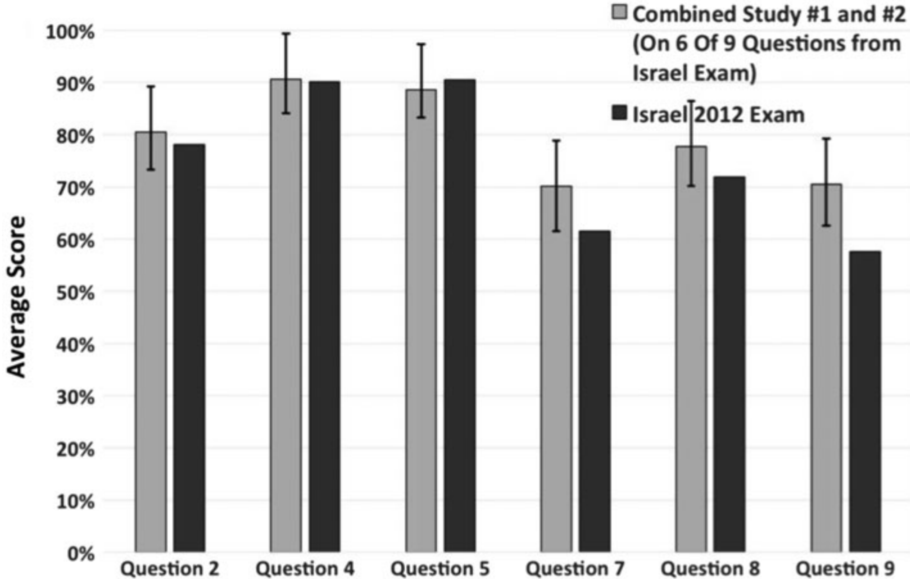

Figure A3. Comparison of student performance in Study1 and Study2 vs. 2012 Israeli students' results (on the six questions from the Israel national exam used in the posttest).

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table A1. Course design revisions in second iteration of design-based research.

	Study1	Study2
Medium of instruction	Face-to-face with one online unit	All units online on OpenEdx used in blended in-classroom setting
Duration	6 weeks	6 weeks + 1 week for final project
Student interactions	Restricted to open-ended pair projects and helping others fix Scratch assignments	Additional student interactions in answering thought questions; in addition to mid-course and culminating project in pairs
"Computing is Everywhere!" video corpus	Included videos that were voted down (e.g. "iPhone as a hearing aid," computation and big data for cancer detection)	More engaging corpus of "Computing is Everywhere" videos, including Google's driverless car; Boston dynamics' Big Dog robot; IBM Watson on Jeopardy!
"Computing is Everywhere!" video use	Restricted to Week 1 only	In addition to the first week (during Unit 1), one such video was added to the materials for Week 2 to 6, to continue giving learners more examples of cool applications of CS (e.g. Mars Rover singing Happy Birthday to itself; Scratch and Xbox Kinect; and others.)
Videos created	In pilot online Unit 8 videos between 4 and 12 min (average ~8 min) in length	~60 videos, between 1 and 5 min in length
Quizzes	Created on school LMS, auto-graded	Created on Stanford OpenEdX, auto-graded, performance data captured in OpenEdX dashboards
FAQ for online course	None	Created a detailed FAQ section
Assessments	Pretest; posttest; PFL test	Ongoing quiz scores; Pretest; Posttest; PFL test; Final projects and project presentations; artifact-based student interviews
Sequencing of content	Pseudo-code and problem-solving paradigm introduced in Week 2	Pseudo-code and problem-solving paradigm introduced in Week 4
Loops and variables	9 days dedicated to these topics	12 days dedicated to these topics
Boolean logic	Introduced with conditionals	Introduced with repeat until construct (before conditionals)
Scratch examples and assignments		Used more games, more creative artifacts, more engaging assignment projects (e.g. dropped assign a grade, simple generic polygon maker; added 4-Quadrant Art, Pong)
Vocabulary development	No special effort beyond providing definition while introducing concept	Created a "word wall" course tab where terms and definitions were added. The teacher also printed this out and pasted on wall just outside classroom door

(*Continued*)

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table A1. (*Continued*).

	Study1	Study2
Final project	Not a requirement (due to constraints imposed by end-of-school year)	Elaborate final project requiring students to also document the experience from pre-project planning to post-project reflections
Final projects demo	Small informal demos with students who chose to share	Final Projects "Expo Day" with each group demonstrating their project to the class
Student interviews	None	In order to assess students' understanding of their programs and thinking behind the code, students were interviewed on the workings of their final project (especially since most students did their projects in pairs). It was also done to compare and contrast students' CT growth as assessed by the posttest

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table A2. PFL test: new syntax specification, questions, their goals, and results describing student responses.

New syntax specification for Questions 1, 2-

'←' (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**

PFL Assessment Question	Goal	Results ***(column to the right for Ques 3,4& 5 indicates number***	

#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");
	To test whether students were able to transfer in ideas of (a) sequence and what comes before and after the (b) looping (using a WHILE loop here), (c) how variable values change with each iteration of the loop and (d) understanding the terminating condition of the loop.	32% of the respondents misunderstood the question (and explained the code instead).
		71% of the 68% who followed the required format of the response (i.e roughly 50% of the total) got it correct while 19% were off by 1 (*num* was 5 rather than 6).
		71% of the total number of students paid attention to the Print commands before and after the loop.	

#2: *What is the value of num at the end of the following code?*
 	
num ← 0;
 	
FOR i ←1 **to** 5 **DO**
BEGIN
    num ← num + i;
END
PRINT(num);
	To test whether students could make sense of (a) a FOR loop and (b) how the variables *num* and *i* changed with each loop iteration.	Just under 50% of the students gave the correct answer of 15. Around 36% of the students wrote 5 or 6 suggesting that they were answering the ending value of variable *i* and not the variable *num*.	

***#3**: What are FirstCounter & SecondCounter keeping track of?*
 	
int TotalCount = 0; **int** FirstCounter = 0;
 	
int SecondCounter = 0;
 	
while (TotalCount < 100) {
    int num;
    num = **InputFromUser()**;
    if (num == 0)
   {
        FirstCounter++;
   }
   else
   {
        SecondCounter++;
   }
   TotalCount++;
}
	To test (a) if students could transfer in ideas they have encountered of conditional checks within loops; and (b) if they could follow how variables are changed based on conditional checks.	**Correct (3 points)**
		<u>Example</u>: "FirstCounter keeps track of how many times the user has input '0' and SecondCounter keeps track of how many times the user has input a number that was not '0."
			19

		Mostly Correct (2 points) *"it keeps track of 0s for 100 responses."*	**2**
		Mostly Wrong (1 point) *<u>Example</u>: variable total set to 0 / variable First set to 0 / variable second set to 0 / repeat total till total <100 / / input variable num / if num is equal to 0 / set first counter else second"*	**1**
		Wrong (0) *<u>Example</u>:"The code is asking for numbers that are less than 100. The two numbers."*	**3**

<table>
<tr>
<td rowspan="10">#4: What are FirstCounter & SecondCounter keeping track of?
#5: How many numbers will be processed by the program below?

int TotalCount = 0;
int FirstCounter = 0;
int SecondCounter = 0;

while (TotalCount < 100)
{
   int x, num;

   num = InputFromUser();
   x = num % 2; // Note this additional command

if (x == 0)
   {
      FirstCounter++;
   }
   else
   {
      SecondCounter++;
   }
      TotalCount++;
}</td>
<td rowspan="10">To test ideas similar to #3 but with the added twist (of adding the modulus operator in between) and to also see if students could figure out in this context how many times a loop iterates.</td>
<td colspan="2">Question #4</td>
</tr>
<tr><td>Correct (3 points)
<u>Example:</u> “FirstCounter is keeping track of even numbers and the SecondCounter is keeping track of odd numbers.”</td><td>14</td></tr>
<tr><td>Mostly Correct (2 points)
<u>Example</u>: “1st is even, second is odd”</td><td>2</td></tr>
<tr><td>Mostly Wrong (1 point)
<u>Ex</u>:”first counter taking track of number / second takes track of the other number”</td><td>3</td></tr>
<tr><td>Wrong (0)</td><td>6</td></tr>
<tr><td colspan="2">Question #5</td></tr>
<tr><td>Correct (answer: 100)</td><td>12</td></tr>
<tr><td>Mostly Correct (answer: Off by 1)</td><td>3</td></tr>
<tr><td>Variables (misunderstood question)</td><td>8</td></tr>
<tr><td>Wrong</td><td>2</td></tr>
</table>

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015

Table A3. Sample "artifact-based" interview (X = Interviewer; I = Interviewee).

Interpretation/question posed	Student quote(s) or exchanges with interviewer (X)
Connected final project idea with event in personal life	"it was becoming Halloween so we thought that we should do like a scary maze … This was the only scary maze that I knew about, coz my cousin made me play it"
"What made you think of the Insidious song?"	"Ooh. it seems … it's a creepy little song. It's like … yeah …"
Able to describe what the code did, referred to the code elements while talking.	e.g. we switched it to the uh.. first maze … and then we put the sprite to the beginning … and then we did a Forever loop … and we did "If … the key up arrow is pressed … change y by 1; and if the down key is pressed, change y by negative … then if touching black uh … you re-start it to the beginning, and um if you touch red you um you um change backgrounds to the second maze"
Knew what the bug was; showed it and had some idea what could fix it without X's help; realized that a variable was needed to keep track of the level and knew where that variable would be set and updated	And as you go to the black, it just takes you down … It's coz we had to like put something in between and … it switches the backdrop. so like in between one of these things … or … if./ else I think, you put in there. X: where would you change that variable … ? I: whenever it touches the red … X: very good! and then when it touches the black … ? I: It depends on what level it is, it would go over there …
Found the project to be fun and creative	X: So in general if you had to say whether it was fun or challenging or difficult or easy or creative or boring which ones would you choose I: Fun X: anything else? I: Creative X: What was the creative part of it? I: Like putting, like, the face at the last, and the screaming …

Downloaded by [Mr Roy Pea] at 08:37 17 May 2015