

# Teaching tools, teachers' rules: exploring the impact of teaching styles on young children's programming knowledge in ScratchJr

Amanda Strawhacker<sup>1</sup>  · Melissa Lee<sup>1,2</sup> · Marina Umaschi Bers<sup>1</sup>

Accepted: 15 February 2017  
© Springer Science+Business Media Dordrecht 2017

**Abstract** Computer programming tools for young children are being created and used in early childhood classrooms more than ever. However, little is known about the relationship between a teacher's unique instructional style and their students' ability to explore and retain programming content. In this mixed-methods study, quantitative and qualitative data were collected from N = 6 teachers and N = 222 Kindergarten through second grade students at six schools across the United States. These teachers and students participated in an investigation of the relationship between teaching styles and student learning outcomes. All participants engaged in a minimum of two lessons and a maximum of seven lessons using the ScratchJr programming environment to introduce coding. Teachers reported on their classroom structure, lesson plan, teaching style and comfort with technology. They also administered ScratchJr Solve It assessments to capture various aspects of students' programming comprehension, which were analyzed for trends in learning outcomes. Results from this descriptive, exploratory study show that all students were successful in attaining foundational ScratchJr programming comprehension. Statistically significant findings revealed higher programming achievement in students whose educators demonstrated flexibility in lesson planning, responsiveness to student needs, technological content expertise, and concern for developing students' independent thinking. Implications for research in the development of computational thinking strategies are discussed, as well as

---

✉ Amanda Strawhacker  
amanda.strawhacker@tufts.edu

Melissa Lee  
leemel@iu.edu

Marina Umaschi Bers  
marina.bers@tufts.edu

<sup>1</sup> DevTech Research Group, Eliot Pearson Department of Child Study and Human Development, Tufts University, 105 College Ave., Medford, MA 02155, USA

<sup>2</sup> Present Address: Department of Curriculum Studies, Indiana University School of Education, 201 N. Rose Avenue, Bloomington, IN 47405-1006, USA

suggestions for successfully implementing early childhood classroom interventions with ScratchJr.

**Keywords** Early childhood education · Improving classroom teaching · Media in education · Teaching/learning strategies · ScratchJr · Computer programming

## Introduction

Many researchers agree that an educator's unique classroom management style has a strong influence on what her students learn in the early childhood classroom (Jeon et al. 2014; Kunter et al. 2007; Mashburn et al. 2008; Rimm-Kaufman et al. 2009). However, considerable uncertainties remain regarding the impact of instructional methods on technological integration and learning outcomes in early childhood (Bebell et al. 2004). The issue of how to foster effective technological interventions in early childhood education settings has become even more pressing with the growing policy-level changes and the administrative support for technological literacy as an academic goal in the US (Keengwe and Onchwari 2011; Cuban et al. 2001). Although this has dramatically improved access and availability of technology in school settings, many intangible barriers to effective technology programs still exist to hinder effective learning practices, including educator beliefs, attitudes, skepticism about technological integration, issues with assessing learning outcomes, and lack of appropriate teacher training (Chen 2008; Hew and Brush 2007; Howard 2013).

This study seeks to describe and explore educational attitudes and practices of teachers who chose to implement ScratchJr in their early childhood classrooms. ScratchJr is an introductory programming language that enables young children (ages 5–7) to create their own interactive stories and games by snapping together graphical programming blocks. Developed as a research collaboration between the DevTech Research Group at Tufts, the MIT Lifelong Kindergarten Group, and the Playful Invention Company, this app facilitates the integration of technology, specifically programming, into the early childhood classroom while promoting problem solving and self-expression (Brennan and Resnick 2012; Flannery et al. 2013; Papert 1980). In this paper, student's programming outcomes and computational thinking strategies, captured by administering a ScratchJr comprehension assessment, are compared and analyzed in light of their teachers' self-reported technology instructional styles to glean patterns and correlations.

## Teaching with technology in schools

Many technology and education researchers cite computer programming instruction as a “paradigm shift” in technological education, because it allows learners to think about problems in a qualitatively different way, using different analytic and representational tools (Koschmann 1996; Papert 1980). Developing “computational thinking,” the set of skills, practices and attitudes around procedural solutions to information-processing challenges, is becoming more and more popular as an educational goal for children in the US and abroad (Brennan and Resnick 2012; Cuny et al. 2010; Portelance and Bers 2015; Pretz 2014; Wing 2006; US Department of Education 2010). Young children have become a particular

research focus, since evidence shows that even children as young as 4 years old can engage in core computational thinking skills, provided they work with a developmentally appropriate tool that supports such learning (Bers 2008, 2012; Kazakoff et al. 2013).

A recurring theme in the literature is the effect of teacher preparation and instructional style on student learning outcomes when classrooms engage with new technology (Fessakis et al. 2013; Ringstaff and Kelly 2002; Keengwe and Onchwari 2011). However, in many studies that examine children's learning outcomes in technology-rich environments, the researchers themselves often develop the lesson plan and lead instruction (Clements and Gullo 1984). While this may have the benefit of controlling for the variable nature of instructors and teaching approaches, or allowing researchers to examine children's potential learning outcomes when exposed to a highly-trained instructor, the downside of this approach is that it is difficult to draw conclusions that extend to the broad majority of educational settings. As Fessakis et al. (2013) point out, the success of programming curricula in PreK-2nd grade classrooms is not as dependent on available technology as it is on "appropriately designed learning activities and supporting material [...] easily integrated in every day school practice by well informed and prepared teachers." (p. 89). Education researchers know that there is diversity among teachers, even well-informed and prepared ones, but it is not yet known what impact that diversity may have on children's technological learning outcomes.

### **Redefining technological integration: technological thinking and learning**

Recently, research has blossomed in areas of educational technology integration, twenty-first century skills, and technological literacy (Blazer 2008; Ringstaff and Kelly 2002). Coding is one of the most powerful aspects of educational technology, and new tablet-based tools and learning initiatives for introducing programming literacy are actively being developed ([www.gethopsotch.com](http://www.gethopsotch.com), [www.hourofcode.com](http://www.hourofcode.com), [www.kodable.com](http://www.kodable.com), [www.scratchjr.org](http://www.scratchjr.org), [www.thefoos.com](http://www.thefoos.com)). Studies have shown that children as young as 4 years old can master foundational concepts of sequencing, parallel programming, looping procedures, and conditional statements (Kazakoff et al. 2013; Sullivan and Bers 2016).

It is important to understand the construct of technological literacy and how it is being approached in the scope of this study. The 2014 National Assessment of Educational Progress (NAEP) developed a landmark Technology and Engineering Literacy Framework, in which technological and engineering literacy is broadly defined as "the capacity to use, understand, and evaluate technology as well as to understand technological principles and strategies needed to develop solutions and achieve goals" (NAEP 2014). The framework puts forth goals for children entering 4th, 8th, and 12th grade, describes several areas and sub-areas within technological literacy, ranging from technological ethics in society to systems thinking and troubleshooting of engineering designs. The research team acknowledges the diversity in interpretation and application of the term technological literacy (Petrina 1998; Technology for All Americans Project, & International Technology Education Association 2000). For example, within the NAEP framework the sub-area of "Construction and Exchange of Ideas and Solutions" is applicable to the general goal of encouraging children to use technology and software to express themselves. For the duration of the paper, technological literacy will be used broadly to refer to skills related to computational thinking.

Computational thinking fosters foundational skills of sequencing, logic, and executive functioning in young learners (Bers 2012; Bers and Kazakoff 2012; Papert 1980). The LOGO

educational programming environment, based on the research of Seymour Papert in the late nineteen-seventies is one of the most influential coding tools that has informed the design and theory behind this new breed of technological tools (Fessakis et al. 2013; Papert 1980; Resnick and Silverman 2005). Papert's work with LOGO was informed by his own theoretical perspective, Constructionism, which held that children learn best when they can create and deconstruct their own physical and virtual objects (Papert 1980). Although Constructionism is a developmental learning theory rather than a pedagogical one, its tenants of child-directed construction in digital environments have influenced the development of countless educational tools and technologies available today. For example, the recent Scratch programming language developed by at MIT was build on the theoretical grounding of Constructionism, and it has in turn been influential in shaping design features of newer tools, including those mentioned at the beginning of this section (Resnick et al. 2009). ScratchJr, a direct descendent of the LOGO and Scratch research, is a programming tool designed to be developmentally appropriate for children ages 5–7 years (Flannery et al. 2013). Indeed, early research has shown that children are able to master core programming concepts in ScratchJr, including sequencing, looping procedures, and parallel programming (Resnick et al. 2009, Portelance and Bers 2015). However, much is still unknown about how teachers use these tools in their schools, and about the learning outcomes that children can achieve in naturalistic learning environments when lessons are taught by teachers, and not by researchers.

Brennan and Resnick (2012) discuss computational thinking in the context of Scratch programming as a set of concepts, practices, and perspectives that shape how students view problems in the world, and their role in seeking solutions. They define a concept as a skill that is useful in most computer programming languages and transferable to non-programming contexts, and practices as those actions that “focus on the process of thinking and learning, moving beyond *what* you are learning to *how* you are learning” (2012, p. 7, original emphasis). For the purpose of this paper, we will focus on two key constructs described by Brennan and Resnick (2012), the concept of sequencing (“A series of individual steps or instructions expressing a particular task that can be executed by the computer,” 2012, p. 3) and the practice of debugging (a “critical [practice] for designers to deal with—and anticipate—problems [in their code],” 2012, p. 6), as metrics to capture student learning in the programming context. Furthermore, two new aspects of computational thinking not mentioned by Brennan and Resnick, will be introduced and examined as they relate to foundational computer programming ideas specific to the early childhood context. These are the concept of one-to-one correlation between a command and its action, which will be called symbol recognition here, and the practice of constructing commands into a sequenced program to achieve a specific action, which will be called goal-oriented sequencing.

## The ScratchJr educational programming environment

ScratchJr is a tablet-based computer programming tool designed specifically to allow young learners (aged 5–7 years) to explore concepts of programming, debugging, and digital content creation in a developmentally appropriate learning environment (Flannery et al. 2013). ScratchJr is the result of 5 years of collaborative research by the DevTech Research Group at Tufts, the Lifelong Kindergarten Group at MIT, and the Playful Invention Company. Elements of the interface, language, and support materials were inspired by the popular Scratch programming language for children ages 8 and older, which was created by the MIT Lifelong Kindergarten Group (Resnick et al. 2009). However, special considerations and decisions were made to ensure the developmental

suitability of ScratchJr for a younger audience. For a depth description of ScratchJr design features and prototype development, see Flannery et al. (2013), and Strawhacker et al. (2015).

ScratchJr is an open-ended coding environment that allows children to create self-directed interactive projects, stories, and games (see Fig. 1). Due to the nature of the tool itself as flexible and open-ended, it is a useful mechanism for examining the relationship between structure in teaching and classroom management styles, and their children's learning of programming concepts and practices. ScratchJr is a free, public-access educational tool, and the ScratchJr research team has also made resource materials and instructional guides freely available. The present study leverages this freedom by investigating teaching styles, classroom practices, and student learning outcomes in early learning centers from across the United States, without requiring researchers to be present during data collection. Participating teachers implemented lessons using ScratchJr with over 200 of their students. Researchers did not participate in any of the lessons, but instead collected detailed quantitative and qualitative data in the form of teaching journals and self-report surveys from the teachers, as well as ScratchJr programming comprehension assessments from their students. These data were analyzed for trends in teacher attitudes and experiences, and possible relationships with student learning outcomes in ScratchJr.

The goal of this study is to examine naturalistic classroom settings in which early childhood educators use ScratchJr software in their lesson activities. Specifically, researchers wanted to know:

1. What programming learning outcomes can we observe in students aged 5–8 years old after they engage with ScratchJr in their naturalistic classroom setting?
2. How might an educator's teaching style with ScratchJr relate to their students programming learning outcomes?



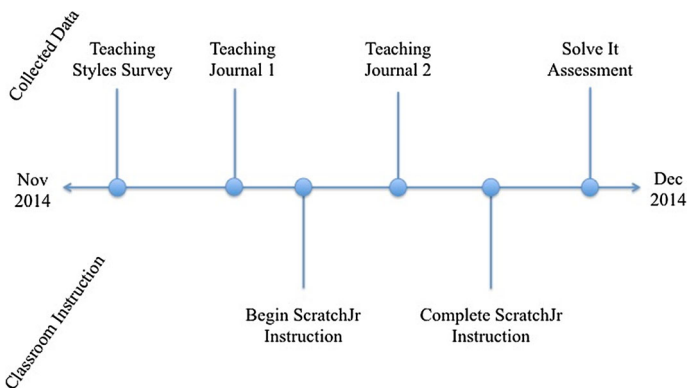
**Fig. 1** Screenshot of a child-made project using the ScratchJr programming environment ([www.scratchjr.org](http://www.scratchjr.org))

## Method

### Study design

A mix of qualitative and quantitative data were collected during this study to examine teaching styles, lesson implementation, and student learning outcomes from schools in various regions of the United States. This study is based on previous work conducted in the U.S. on early childhood programming assessments. In one study, a pilot sample of three classrooms used a beta-version of ScratchJr to facilitate teachers' daily and weekly lessons (Lee 2015). Teachers were invited to incorporate ScratchJr into lessons in any way they preferred, while researchers collected qualitative observations and quantitative assessments of students' learning outcomes related to ScratchJr. From this pilot study, several interesting themes emerged, including the relationship between a teacher's classroom management style and lesson plan on students' comprehension of ScratchJr programming content. Results showed that teachers with highly-structured classroom management styles, but largely student-driven instructional activities, typically yielded the highest performance from their students on assessments of ScratchJr programming knowledge Lee 2015. Additionally, in a study of young children's learning using the ScratchJr programming language, Flannery et al. (2013) found that older children consistently demonstrated more sophisticated programming expertise compared to younger ones.

The current study explores these same themes and outcomes on a larger scale, involving  $N = 6$  teachers and  $N = 222$  students from 6 schools around the United States. Teacher volunteers agreed to lead at least two lessons of 45 min or longer involving ScratchJr in their regularly scheduled classes over the course of one school year. In addition, teachers submitted several journal entries and surveys that were used to collect qualitative data about their teaching philosophy, relevant lesson plans, and classroom experiences using the software. Finally, teachers were asked to administer a ScratchJr programming assessment to their students and submit the results back to the research team for evaluation. See Fig. 2 for a complete list in relative chronological order of data collected from participating teachers and their students.



**Fig. 2** Data collection timeline. *Note:* This timeline describes the relative chronological order of data collected and submitted to the research team by participating teachers

## Procedure and sample

This study used a self-selected sample of teachers who volunteered to provide self-report data through the duration of their ScratchJr lesson implementations. Recruitment was done by sending an study invitation to a subscription-based e-mail list of teachers interested in volunteering to participate in education research. Teachers were asked to complete at least two lessons (either in a school or after-school program) totaling 90 min of instruction over the course of 2 months (November–December 2014). Of the initial 41 volunteer teachers from 34 schools around the US and internationally, six went on to complete and submit requested data, one of them male and five of them female teachers (see Fig. 2). Data is presented here in as much detail as is possible considering that some teachers chose not to answer some questions concerning their personal background and experience teaching young children and integrating technology in school settings. It should be noted that all six participating educators were not lead classroom teachers, but filled various education roles that can be characterized as technology integration specialist. Technology integration specialists are described by Hofer et al. (2004) as “school-based [staff] whose primary concern is empowering teachers to harness the power of technology integration for student learning” (p. 1). These educators worked with several classrooms within their schools, and were the only teachers represented from each school. To address this nuance, we will be using the words “teacher” and “educator” interchangeably throughout this report. Additionally, educators did not report to us about specific classroom structures, as their unique contexts were sometimes quite complex. For example, some teachers reported “seeing the same group of kids every other week” and others referred to “special block time” that caused them to work with children from the same classroom at different times.

Of the six participating schools, five are private institutions with tuition and academic application requirements, and one is a public magnet school focusing on STEM (Science, Technology, Engineering, and Mathematics) education. Total school enrollment ranged from 180 to 757 students, and proportions of students of color varied, with the lowest being

**Table 1** 2013–2014 participating school demographics

School ID	Total school enrollment	% Students of color	Early childhood enrollment	Grade span	Type	Locale
A	624	16%	108	Pre-K–12th	Private	Large suburb
B	306	8%	68	Pre-K–8th	Private	Small city
C	708	22%	92	Pre-K–12th	Private	Large suburb
D	277	11%	74	Pre-K–8th	Private	Large suburb
E	757	50%	297	Pre-K–6th	Public magnet	Large suburb
F	180	<i>Unreported</i>	<i>Unreported</i>	Pre-K–8th	Private	<i>Unreported</i>

Demographic information based on US Census Bureau 2015, US Department of Education 2013, National Center for Education Statistics 2015a, b

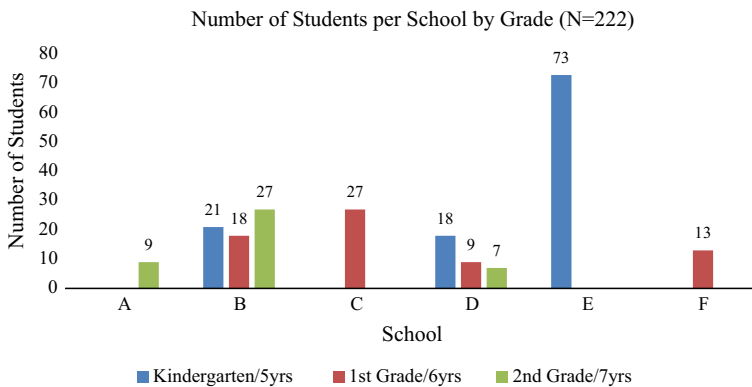
8% of the student body and the highest being 50% (see Table 1) (US Census Bureau 2015, US Department of Education 2013, National Center for Education Statistics 2015a, b).

All schools allowed student access to iPads during class time, and implemented a one-iPad-per-student program, starting in the early grades. Students' programming assessments were collected from six of these participating schools, and comprised work done by 222 5 to 7-year-old students in Kindergarten through to second-grade. See Fig. 3 for a breakdown of students according to grade/age in each school. In the U.S, where all the schools are based for this study, kindergarteners typically matriculate at 5–6 years of age, first grade students are typically 6–7 years old, and second grade students are typically seven to 8 years old. For the duration of the paper, we will be using grade to categorize the children in the sample.

## A look in the classroom

### Teacher self-reports

Before introducing ScratchJr in their classrooms, teachers were asked to complete several surveys to capture their instructional style and lesson planning process. Questions were



**Fig. 3** Students in the sample, by grade and school

**Table 2** Teaching context and lesson implementation

School ID	M/F	Student: teacher ratio	# Lessons	Length of lessons (min)	Total lesson time (h)
A	F	12:1	7	50	5–8
B	F	13:1	7	30–45	3.5–5.25
C	F	15:1	4	30–45	2–3
D	F	10:1	7	30–45	3.5–5.25
E	F	24:1	4	30–45	2–3
F	M	17:1	8	45–60	6–8

Demographic information based on US Department of Education, National Center for Education Statistics 2015a, b



answered using a 5-point Likert scale ranging from Strongly Disagree to Strongly Agree. For the duration of the study, teachers periodically (at least twice, but more often if they chose) recorded semi-structured "Teaching Journals," which asked open-ended questions about their teaching experience. Sample questions included "How have students responded to ScratchJr in your classes," "How many lessons with ScratchJr have you completed," and "What are some challenges you faced when teaching with ScratchJr, and how did you address this in class." Journal posts were uploaded directly into an online repository of responses organized by timestamp and teacher information. In these surveys and journal posts, teachers reported on information about their activity plans, number of students attending ScratchJr lessons, and frequency and duration of lessons (see Table 2).

## Teaching styles

In order to measure teachers' unique instructional styles and attitudes, the researchers developed a survey based on teaching style instruments currently used in education research (Ford and Il 2015; Grasha and Yangarber-Hicks 2000; Limongelli et al. 2013; Lin et al. 2013). The Grasha–Riechmann Teaching Styles Inventory, developed within the field of education research, categorizes educator behaviors into five styles: Expert, Formal Authority, Personal Model, Facilitator, and Delegator (Grasha and Riechmann-Hruska 1996). In Grasha and Riechmann's research, teachers were shown to exhibit these styles (often more than one) when teaching, and to influence the classroom environment in fairly predictable ways based on their approach (Grasha 1994, Grasha and Riechmann-Hruska 1996). The Grasha–Riechmann Inventory was developed in the context of higher-education, with college faculty. For this study, the survey instrument has been modified to apply directly to early childhood educators, and to focus on instances when teachers mediate lessons using technologies or tangible teaching tools. This modified survey consists of 40 items and uses the Likert scale answer system described above.

Since Grasha (1994) argues that all teachers demonstrate some level of all teaching styles, the surveys will be used here to capture the magnitude of teachers' use of each style. As such, each style will be described here as individual measures of teaching style, taken from Grasha's discussion of the five teaching styles that emerged from his research (1994). Teachers who use the Expert teaching style are focused on displaying detailed knowledge, and challenging students to enhance their competence. The Formal Authority style is characterized by a focus on setting clear, almost rigid expectations and reinforcing acceptable ways of doing things. The Personal Model style focuses on a hands-on approach, and these teachers often use themselves as models of how to engage in the content. Facilitator teachers are characterized by an emphasis on personal flexibility, a focus on students' needs and goals, and a willingness to explore options and alternative courses of action to achieve them. Finally, Delegator style teachers are mainly concerned with developing students' capacity to work autonomously. Teachers fell into High, Medium, and Low categories for each style based on their scores on the Grasha–Riechmann Teaching Styles Inventory. These categories were arrived at using Grasha's recommendations after analyzing results from 381 teachers results on the Inventory (Grasha 1994).

These styles will be analyzed separately as predictors of children's programming learning. In the conclusion, the authors will return to each teacher's unique style in the context of their self-reported experiences teaching with ScratchJr.

## Technological, pedagogical, and content knowledge

Teachers' knowledge of pedagogical approaches and content knowledge around ScratchJr integration was measured using targeted questions about their ScratchJr content mastery, as well as relevant questions from the Technologies, Pedagogies, and Content Knowledge (TPCK) instrument, modified for in-service early childhood educators (Bers et al. 2013; Schmidt et al. 2009). These questions represented items from all three domains of Technology, Pedagogy, and Content Knowledge. The 23-item Likert-style survey captured elements of ScratchJr knowledge, ability to integrate technology with traditional subject material, and attitudes about technology use in their schools and classroom.

## Teaching experience and attitudes

In addition to the surveys adapted from existing research instruments, teachers also completed a 16-item self-reflection questionnaire in order to describe their approach to lesson planning, classroom structure, and teacher-led versus student-led activities. Teachers also submitted semi-structured journal entries online at various points throughout their lesson implementation. Although researchers requested at least one journal entry before and one during or after the second ScratchJr lesson, teachers were invited to submit more as they liked. It is beyond the scope of this paper to present a comprehensive analysis of these interview-style journal entries. However, excerpts will be used to contextualize quantitative findings.

### *ScratchJr solve its: show what we know*

In order to assess the relationship between the diverse teaching styles, technological knowledge, and pedagogical approaches of the six participating teachers, an assessment measure was administered to all 222 students who took part in the ScratchJr interventions (see Fig. 4). Questions were developed based on pilot work with ScratchJr in schools (Portelance and Bers 2015), targeting children's sequencing skills and programming knowledge (Strawhacker and Bers 2015; Sullivan and Bers 2016). These assessments allow us to describe how students have understood programming concepts after engaging with teachers using various technology-mediated instructional approaches.

To complete the Solve It assessment, students are asked to view a series of videos uploaded to a dedicated YouTube channel ([bit.ly/2IDEN0Q](https://bit.ly/2IDEN0Q)) showing ScratchJr projects and several written prompts, to be read aloud by the teacher. The YouTube channel was developed for ease of use by teachers with internet access, but downloaded versions of the videos were available to teachers via email request. The videos were recorded in ScratchJr's "Presentation Mode," so that students could view a finished ScratchJr project, but not the programs that created it (see Figs. 4, 5, 6).

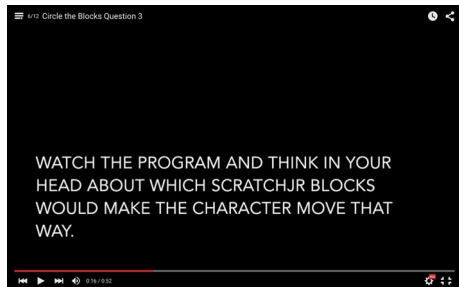
Solve It videos showed several ScratchJr projects, and asked questions about each of them, resulting in twelve unique Solve It questions or tasks. Teachers were encouraged to allow students to re-watch videos up to five times, and to break up the assessment into no more than three questions in one session or class period. The content of the videos and the ScratchJr programming blocks that they demonstrate are detailed in Tables 3 and 4.

In the following descriptions, it is important to distinguish between the words *program* and *project*, as they are not used interchangeably here. A *program* is a single sequence of programming blocks that only applies to (or controls) a single character. For example, a



**Fig. 4** Kindergarten students complete a solve it assessment during circle time in their classroom

**Fig. 5** Screenshot from Solve It Question videos on YouTube



**Fig. 6** Screenshot from Solve It Project videos on YouTube



program could be a list of ScratchJr blocks reading, Begin, Move Right, End. A *project*, on the other hand, is the term in ScratchJr for a collection of pages and characters, each with its own program(s), that come together to form a cohesive whole. A project might show several characters conversing and then taking a trip to the ocean together. It is possible to show a project without revealing any character's programs, and in fact that is what was recorded to create the Solve It videos. Children only saw several successive projects, and were instructed to surmise based on the characters' actions what their programs must be. In traditional computer science, this is akin to hiding the terminal or compiler of a program, and showing only the executed state resulting from running the program. In software

**Table 3** Solve it videos and ScratchJr concepts used

Video description	Programming commands used	Programming concepts assessed
Video 1: Gliding Cat—This project shows a hand tapping a cartoon cat, making the cat glide left and right on the screen. In some questions, the cat also performs an action between glides such as a spin, a jump, or a step up or down on the screen	Motion Blocks that make characters move around the screen. Interactive Blocks that initiate a character's program when a human user taps on the character	Sequences of actions
Video 2: Hopping Rabbit and Cat—This project shows a rabbit that hops and glides to the right, and bumps into a cat. The cat then imitates the rabbit by hopping and gliding to the right	Motion Blocks that make characters move around the screen. Interactive Blocks that initiate one character's program when another character touches it on the screen	Conditional sequences that require a concrete trigger event (i.e. one character touching another one). Timing in sequences of actions. Multiple programming solutions to accomplish one sequence or result
Video 3: Cat and Chicken Conversation—This project shows a cat approach a chicken, and emit a speech bubble that reads "Hi!" The chicken then moves towards the cat and another speech bubble appears above the chicken that reads "Hi!"	Motion Blocks that make characters move around the screen. Interactive Blocks that initiate one character's program at a specific time during the project	Conditional sequences that require an abstract trigger event (i.e. one sequence programmatically "calls" another one to action without changing the displayed characters). Timing in sequences of actions. Multiple programming solutions to accomplish one sequence or result

**Table 4** The four types of solve it questions

Question type and description	Computational thinking concept	Computational thinking practice
Fix the Program (multiple choice)—observe two videos of the same project, with one programming command changed from the first to the second viewing. Given the original program, the student must circle the block that was added or removed to alter the project	Symbol decoding	Debugging
Circle the blocks (multiple choice)—observe a video of a project. Given all the possible blocks in ScratchJr, the student must circle all blocks that were used to create the project	Symbol decoding	Goal-oriented programming
Match the program (multiple choice)—observe a video of a project. Given four possible complete programs, the student must circle one of the two correct programs that would create the project they viewed	Sequencing comprehension	Debugging
Reverse engineering (open response)—observe a video of a project. Given all possible blocks in ScratchJr, the student must recreate all sequences used to create the project	Sequencing comprehension, symbol decoding	Goal-oriented programming

development fields, the task of analyzing a system in order to identify the system's components and recreate the design is referred to as "reverse engineering" (Chikofsky and Cross 1990). Although all of the Solve It tasks rely in part on reverse engineering skills, the term "reverse engineering" is only applied to one Solve It task. These tasks are explained in more detail in the next sections.

Solve It questions were adapted from pilot versions of the Solve It assessment used in previous studies of obotics in Kindergarten through second grade classroom settings (Strawhacker and Bers 2015). Results from this prior work suggest that students from Kindergarten through second grade had developed a firm grasp of programming concepts after engaging in developmentally appropriate programming educational interventions. After 13 h of open-ended programming instruction, Kindergarteners were able to demonstrate comprehension of core programming elements such as symbol recognition in the ScratchJr language, and debugging basic sequences of actions.

### *Scoring the solve its*

Solve It tasks were designed to capture student thinking using a variety of response types, focusing on core concepts of command and sequencing comprehension, and core practices of debugging and program construction (see Table 4). These skills were targeted as evidence of computational thinking development in 5- to 8-year-old learners (Kindergarten through second grade) (Brennan and Resnick 2012). The scoring rubric was designed by the lead researchers and assessments were scored by research assistants with substantial interrater agreement ( $\kappa = 0.68$ ).

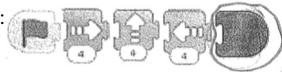

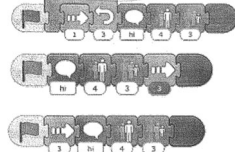

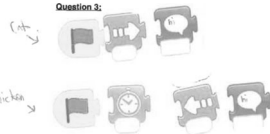
### **Fix the program tasks**

Fix the Program (Fix) tasks require children to observe two versions of the same project, and identify the programming block that is different between the two. This question type is considered multiple choice, because programming blocks are shown for the child to choose from. This task type is intended to assess a child's ability to engage in the computational thinking practice of debugging, or correcting an existing program based on observation of its execution. Since children must select from various ScratchJr blocks based on what they observed in the project, this task also reveals whether children can apply the computational concept of symbol decoding, or recognizing that the actions they observed on the screen correspond to specific programming icons on their answer sheet (see Table 5).

### **Circle the block tasks**

Similar to Fix tasks, Circle the Block (Circle) tasks require children to observe a project, and identify the programming blocks that comprise its program. However, instead of selecting a single block, this question asks children to select from a list of all 24 ScratchJr blocks and circle *all* the blocks that are needed for that project's creation, including blocks used by different characters in the project. Although this question type is technically still multiple choice ("select all that apply" style), it is much more difficult because the original program is not shown (for obvious reasons) as it was in Fix tasks. Unlike Fix tasks, which focus on debugging, Circle tasks are intended to assess a child's ability to engage in the computational thinking practice of goal-oriented programming, or programming with a

**Table 5** Student-completed example of each type of solve it question, organized by computational thinking concepts and practices

Skills assessed	Debugging	Program construction
Command comprehension	<p>Circle a</p> <p>Circle a block to remove</p> <p>block:</p> 	<p>Circle all necessary</p> <p>Question 1:</p>  <p>blocks:</p> 
Sequencing comprehension	<p>Circle a sequence:</p> <p>Question 2:</p> 	<p>Construct a sequence:</p> <p>Question 2:</p> 

specific end-product in mind. The block-identification element of the task also tests children's ability to decode symbols.

### Match the program tasks

Match the Program (Match) tasks require children to observe a project with a single character and select from four options the best match for that character's program. This question type is multiple choice, and each multiple choice option is a complete program. Like the Fix task, Match tasks are intended to assess children's ability to engage in the computational thinking practice of sequencing, or intentionally ordering blocks in a specific series. Since children can only select from several completed sequences, they must read each one to identify errors in their construction, and so the practice of debugging is also assessed.

### Reverse engineering tasks

Reverse Engineering tasks require children to observe a program, then identify (using cut-out paper icons of all 24 blocks) the programming blocks needed to create its programs, and finally construct programs for all characters shown in the project. This is the only open-response question type, and is certainly the most advanced Solve It task. Because there are many possible ways that a child can correctly (or incorrectly) respond to this task, the scoring of this task is significantly more complicated. After several trials, efforts were made to create a simple, flexible rubric to apply to multiple classifications of answer. Responses are scored in stages, first to assess whether there are distinct programs for each character in the project (unless only one character was shown), then to identify whether correct programming blocks were selected, and finally to score the relative sequence of the blocks. This task type is intended to capture children's ability to engage in the practice of

goal-oriented programming. Since responses were judged based on children's block selection as well as their relative order, computational thinking concepts of symbol decoding and sequencing comprehension were both assessed in Reverse Engineering tasks (see Table 5).

## Results

### Student learning outcomes: solve it tasks

Solve It assessments were collected from  $N = 222$  students in the  $N = 6$  participant schools. Of the 12 questions, one Fix task was found to have a typographical error making the question unsolvable, and so it was not analyzed. The eight multiple choice tasks (Fix, Circle, and Match) had an 89% or better response rate from the entire sample, with a varied proportion of responses from Kindergarten students ( $89 \leq N \leq 112$ , depending on the question), first grade students ( $N = 65$ ), and second grade students ( $N = 42$ ). The open-response tasks (Reverse Engineering) had just over a 47% response rate from the entire sample, and comprised  $N = 26$  K, 21 first, and 34 s graders (23, 32, and 79% of their grade sample, respectively).

### General results: comprehension of ScratchJr

Students could earn a possible 42 points total on the Solve It assessment, although a total score does not convey much information since each question type has its own scoring system. For this reason, raw scores were transformed into T-scores, which standardize scores around a mean of 50 in order to more precisely describe a single score in terms of the distribution of all other scores.

Correlations were computed among the eleven Solve It Tasks, and between grade and Solve It performance. The results suggest that several of the 65 correlations were statistically significant. Many tasks were strongly correlated (using two-tailed Pearson's product-moment correlations) with other tasks of the same type, for example Fix Tasks 1 and 2 ( $N = 215$ ),  $r = +0.43$ ,  $p < 0.001$ , Circle Tasks 1 and 3 ( $N = 194$ ),  $r = +0.54$ ,  $p < 0.001$ , and Reverse Engineer Tasks 2 and 3 ( $N = 78$ ),  $r = +0.35$ ,  $p < 0.05$ . Performance on Fix Task 1 was a strong predictor of performance on many other tasks, and results indicated a strong positive relationship ( $p < 0.05$ ) between a student's grade and their scores on many Solve It tasks. This suggests that these tasks probably do consistently measure ScratchJr programming comprehension, and that students in older grades demonstrate higher programming comprehension than younger grade students.

Figure 7 shows spread of scores for the  $N = 62$  students who completed all questions on the Solve It assessment. The frequency of transformed scores is normally distributed and centered around a mean of 52.51 points ( $SD = 4.05$ ), although there are minor dips and peaks throughout. This heteroskedasticity may be caused by several factors, including nested trends among children of the same grade and school.

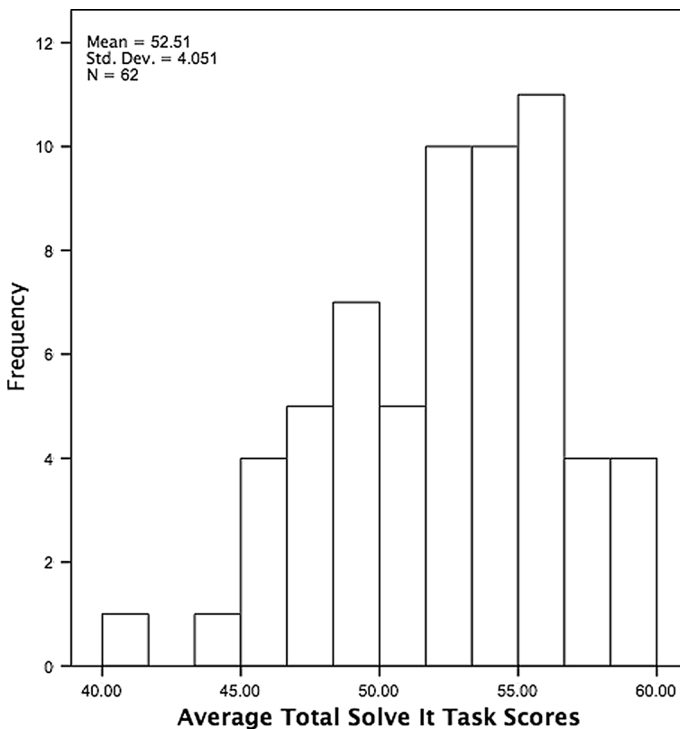
Of the  $N = 62$  students who completed all Solve It tasks,  $N = 21$  were Kindergarteners,  $N = 11$  were first graders, and  $N = 29$  were second graders. On average, each cohort performed slightly better than the last, with Kindergarteners averaging 49.9 points ( $SD = 4$ ), first graders averaging 52.2 points ( $SD = 3.1$ ), and second graders earning an average of 54.6 points ( $SD = 3.3$ ) on their total Solve It assessment. Because the data for

each grade cohort were normally distributed, a Pearson product-moment correlation coefficient was computed to reveal a positive correlation between a student's grade and their performance on the Solve It assessment [ $r = 0.514$ ,  $N = 62$ ,  $p < 0.001$ ]. Although the sample sizes make it difficult to draw larger conclusions from this finding, it may be evidence of a developmental progression in children's programming comprehension capability. Further, the overlap of standard deviations across each cohort (e.g. the first grade mean score falls within one standard deviation of the Kindergarten mean score) shows the range of children's abilities, as some students in younger grades demonstrated roughly the same programming comprehension levels as some in older grades.

The small sample size prohibited a closer look at differences among children from different schools across all tasks. To examine these trends in more detail, as well as to consider those remaining  $N = 160$  students who did not complete all eleven questions, task types are analyzed separately.

### Fix results

Fix the Program scores showed an extreme ceiling effect, with the majority of children ( $N = 215$ ) earning a perfect or near-perfect score. This trend is apparent across children from most grades and schools. The only major exception came from Kindergarteners, primarily from School D and School E. Interestingly School B ( $N = 21$ ,  $SD = 6.6$ ), the only other school with Fix scores from Kindergarten students, outperformed School E ( $N = 73$ ,  $SD = 9.6$ ) by an average of 4 points per question, and School D ( $N = 18$ ,



**Fig. 7** Distribution of average total solve it scores



$SD = 11.8$ ) by an average of 12 points. School B also had the narrowest standard deviation of any Kindergarten group, suggesting that School B Kindergarten responses were more uniform and precisely centered around the mean, with fewer outliers (and perhaps fewer guesses in responses).

For the most part, results in all grades and schools show an extreme positive skew, indicating that the Fix questions were generally easy for many students. This finding, that some tasks posed almost no challenge to most children participating in the study, forecasts a trend that will recur in later task analyses.

### Circle results

Of all the Solve-it tasks, Circle the Block was the only task type that yielded normally distributed scores ( $N = 193$ ,  $SD = 8.2$ ). Just as with the Fix tasks, the means of each grade are progressively higher, with older grades outperforming younger ones. Distributions by school are also relatively normal, although means, spreads, and sample sizes differ. A two-way MANOVA was run to determine the effect of school and grade on student's performance on Circle tasks 1, 2, and 3. Results from the MANOVA show that there was a statistically significant difference in performance on the circle tasks based on a child's school,  $F(15, 502.82) = 7.942$ ,  $p < 0.0005$ ; Wilk's  $\Lambda = 0.556$ . There was also a statistically significant difference in performance on the circle tasks based on a child's grade,  $F(6, 364) = 11.410$ ,  $p < 0.00105$ ; Wilk's  $\Lambda = 0.708$ . However, there was not a significant impact for the interaction of grade and school ( $p > 0.05$ ).

In order to look at where these differences fell, univariate ANOVAs were examined. It is evident that school had a significant impact on Circle Tasks 1 [ $F(5, 184) = 5.868$ ;  $p < 0.0005$ ], Circle Task 2 [ $F(5, 184) = 13.777$ ;  $p < 0.0005$ ], and Circle Task 3 [ $F(5, 184) = 13.309$ ;  $p < 0.0005$ ]. It is also clear that grade had a significant impact on Circle Tasks 1 [ $F(2, 184) = 4.247$ ;  $p < 0.025$ ], Circle Task 2 [ $F(2, 184) = 16.307$ ;  $p < 0.0005$ ], and Circle Task 3 [ $F(2, 184) = 16.502$ ;  $p < 0.0005$ ]. Please note that in order to make an alpha correction for the multiple ANOVAs being run, we are accepting statistical significant at  $p < 0.025$ .

### Match results

The three Match the Program tasks followed the same trends from the Fix tasks. Steep positive skewness of the data suggests that either the  $N = 194$  participants generally understood the tasks, or that some underlying variable is influencing the trend. In support of this second theory, it is interesting to note that the peaks along the left tail of the data, several standard deviations outside the mean, are comprised mainly of responses from school E, a sample of  $N = 73$  Kindergarten students. These children found challenges in Match task 1 ( $M = 44.75$ ,  $SD = 10.57$ ) and 2 ( $M = 45.17$ ,  $SD = 10.7$ ), but improved on task 3 ( $M = 48.82$ ,  $SD = 10.85$ ).

### Reverse engineering results

The assessment of Reverse Engineering yielded the most non-uniform data, perhaps because this is the most difficult concept (or rather, constellation of concepts and practices) to assess. The multiple peaks in these data are not explained by either grade or school trends, and thus imply another variable that is not captured in the Solve It assessments. It

should be noted that Reverse Engineer tasks had the lowest number of participants, a total of  $N = 78$ , or just over one-third of the total sample. This is because of challenges reported by teachers in task administration, which will be discussed in later sections.

### *Examining the classroom environment*

Results from the Teaching Styles survey revealed a variety of attitudes about technology, classroom structure, and ScratchJr content knowledge. The six teachers completed a 79-item survey, comprising three sub-scales.

The 40-item Grasha–Riechmann Teaching Styles Inventory revealed categories of teaching behaviors in our participant sample. These responses run on a scale from 5 (strongly agree) to 1 (strongly disagree) and include questions such as, “students typically work on projects alone with little supervision from me,” “my expectations for what I want students to do in this class are clearly defined,” and “activities in this class encourage students to develop their own ideas about the content.” Scores were calculated by averaging the relevant items, and their strengths (denoted by asterisks) were arrived at by comparing against the Grasha–Riechmann test norms (Grasha 1994; Grasha and Riechmann-Hruska 1996).

All of the teachers earned a High score for the Delegator teaching style (see Table 6), meaning that all of the teachers in this sample value “developing students’ capacity to function autonomously” (Grasha 1994). Generally, teachers earned mixed scores on the rest of the assessment, with most teachers earning a Mid or High score on all of the teaching styles. This is not surprising, as Grasha mentions that most teachers employ a flexible variety of teaching styles.

The 10 items relating to TPCK and 13 items about ScratchJr knowledge revealed very high marks overall, indicating that our teacher sample considers themselves very capable with educational technology in general, and ScratchJr in particular (see Fig. 8). TPCK items included questions like “my school encourages me to use new technologies and resources in my classroom,” and “it is important to use technology in my teaching.” ScratchJr knowledge items included, “I understand how to use programming blocks to make a character grow bigger and shrink in ScratchJr,” and “I understand how to make my character move by pressing the “Green Flag” in ScratchJr.”

These scores also indicate that our sample is not overly diverse in terms of Technological, Pedagogical, or ScratchJr Content knowledge. This may not be surprising, given the self-selected nature of the sample, and the fact that many of the teachers participating in this study are self-described technology integration specialists.

**Table 6** Grasha–Reichmann teaching styles inventory results by teacher

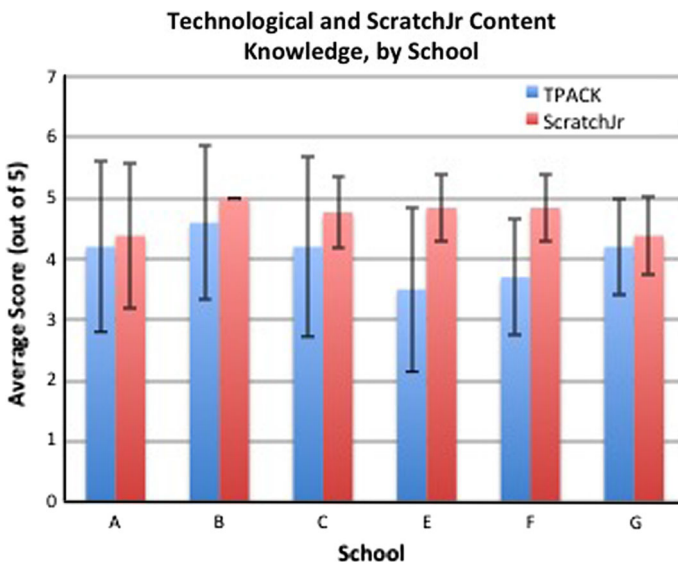
	Expert	Formal authority	Personal model	Facilitator	Delegator
Teacher A	Mid (3.25)	Mid (2.75)	Mid (3.25)	Low (2.87)	High (3.25)
Teacher B	Mid (3.37)	High (3.37)	High (3.62)	High (4.50)	High (4.12)
Teacher C	Mid (2.87)	Mid (2.62)	Mid (3.25)	High (4.87)	High (4.62)
Teacher D	Low (2.62)	High (3.12)	Mid (3.27)	High (4.12)	High (3.87)
Teacher E	Mid (2.87)	High (3.25)	Mid (3.25)	Mid (3.87)	High (3.75)
Teacher F	High (4.25)	Mid (3.00)	High (3.62)	Mid (3.5)	High (3.37)

Correlational analysis was used to examine the relationship between teaching style and perceived ScratchJr proficiency and comfort on the 79-item survey. In general, questions on each sub-scale were correlated with each other. No single question was a good predictor of questions on other sub-scales.

### *Modeling the relationship between teaching styles and programming outcomes*

A hierarchical multiple regression was conducted to generate a model of the relationship between teaching styles and Solve It programming scores, with teaching style scores used as predictor variables. A hierarchical regression was run to compare the impact of teaching styles on children's transformed mean Solve It scores. The hierarchy of predictors allowed the model to control for the impact of grade on the distribution of scores. This is because grade has been shown in prior work to be a strong predictor of Solve It scores (Sullivan and Bers 2016). Controlling for grade means that the second step of the regression isolates the relative impact of each teaching style on Solve It scores.

Multiple regression was justified in this case due to the data generally meeting the required assumptions to explore this statistical approach. Mean Solve It residuals looked normal based on the bell-shaped distribution of data and the relatively straight line of the probability–probability plot, although the distribution did reveal a very slight trend towards a negative skew. Indeed, nine cases, or 2.7% of the data fell outside the range of  $-2.5$  to  $2.5$  in casewise diagnostics. This is slightly higher than expected, but the 1.7% of outliers were determined not to be exerting undue influence on the mean. This robustness is likely due to the large sample size. Regressions of each variable against the outcome were largely uncorrelated and random, which suggests adherence to the assumption of independence of errors and homoscedasticity. In support of this finding, the Durbin-Watson statistic was 1.4 (very close to the recommended value of 2), showing the errors to be reasonably dependent. Finally, VIF values for all variables are well under the recommended limit of 10, and



**Fig. 8** Teachers' TPACK and ScratchJr knowledge scores

the tolerance values are well below 0.2. These facts both suggest that there is no multicollinearity in the data.

Based on the final model, the predictor variables show the following coefficients in the model to predict students scores on Solve It programming assessments (see Table 7). The Delegator teaching style was excluded from analysis, because teacher uniformly earned the highest score on this category, so there was no variability to correlate with student programming scores.

Of the remaining four teaching styles, Expert and Facilitator significantly predicted Solve It scores. Student grade accounts for 24.5% of Solve It scores in our sample. This value increased to 37.7% when Teaching Styles were added to the model, meaning that Teaching Style accounts for 13.2% of the variability in children's programming scores. Adjusted  $R^2$  is slightly smaller for the second model (36.2%). This shrinkage suggests if the model were derived from the population, it would account for slightly less variance in Solve It scores, closer to 11.7%. If we apply Stein's formula, we find an adjusted  $R^2$  value of 0.067, which is lower than the observed value of adjusted  $R^2$  (0.076), suggesting that the cross-validity of this model may not be very good. In conclusion, when controlling for student's age, teaching style alone accounts for approximately 12% of the variation in students' programming scores. With all four of the predicting variables included, the model accounts for over a third of the variance in Solve It programming scores within the sample.

Two strong predictors of high achievement on children's programming scores were the Expert and Facilitator teaching styles. As previously mentioned, the Expert teaching style is focused on displaying detailed knowledge and challenging students to enhance their competence, while the Facilitator style is characterized by an emphasis on personal flexibility, a focus on students' needs and goals, and a willingness to explore options and alternative courses of action to achieve them (Grasha 1994). Students of teachers who scored high on the Facilitator style scored an average of 6 points higher on Solve It than children whose teachers earned a low Facilitator score. The Expert teaching style was an even more dramatic predictor, with children of high-Expert teachers earning about 13

**Table 7** Linear model of predictors of Solve It scores, with 95% bias corrected and accelerated confidence intervals reported in brackets

	<i>b</i>	<i>SE B</i>	$\beta$	<i>p</i>
<i>Step 1</i>				
Constant	46.20 [45.15, 47.24]	0.53		$p < 0.001^*$
Student grade	4.365 [3.36, 5.37]	0.511	0.50	$p < 0.001^*$
<i>Step 2</i>				
Constant	26.94 [15.37, 38.51]	5.87		$p < 0.001^*$
Student grade	3.94 [2.70, 5.18]	0.63	0.45	$p < 0.001^*$
Expert	6.77 [3.39, 10.16]	1.72	0.45	$p < 0.001^*$
Formal authority	0.48 [-2.24, 3.19]	1.38	0.03	$p = 0.730$
Personal model	-1.04 [-4.16, 2.08]	1.58	-0.07	$p = 0.513$
Facilitator	3.07 [1.01, 5.13]	1.04	0.26	$p = 0.004^*$

Confidence intervals and standard errors based on 1000 bootstrap samples

$R^2 = 0.25$  for Step 1;  $\Delta R^2 = 0.12$  for Step 2 ( $ps < 0.001$ )

\* Statistical significance

points higher than students with low Expert-style teachers. The Formal Authority teaching style, which focuses on setting clear, rigid expectations and reinforcing acceptable ways of doing things, was not a predictor of student programming outcomes. Neither was the Personal Model style, characterized by hands-on demonstrations using oneself as a role model. The Delegator style, concerned with developing students' capacity to work autonomously, was not included in the analysis because of a lack of diversity in teacher responses on the Inventory (all teachers were high-scoring for this style), so it is impossible to determine its effect on Solve It scores.

## Discussion

In the following section, patterns in student programming assessments will be discussed, and contextualized with teachers' experiences and reflections. These data will be triangulated to arrive at a narrative explanation of the data and their relevance to the research questions. Study limitations, future work, and practical applications of the findings will be discussed.

### **ScratchJr solve its: computational thinking successes, and learning from mistakes**

Students in all schools represented in the sample performed strongly all Solve It Tasks, demonstrating computational thinking skills of symbol decoding and sequencing comprehension, and computational thinking practices of debugging and goal-oriented programming. The pattern of scores improving with students' age, and the normal distribution of responses within grades, lends internal validity to the Solve It assessment itself. Further, the statistically significant difference in mean scores on all three Circle Tasks means that these results can be extrapolated into general predictions for the larger population. We might expect to see similar trends in other early childhood classrooms that have participated in at least two ScratchJr lessons, independent of teaching style or classroom environment. Taken as a whole, these results suggest that students are able to grasp concepts with some regularity across age levels.

As with most assessments, it is interesting to examine students' "incorrect" solutions in order to glean their thinking strategies (Ginsburg 1997). For example, many students in early grades regularly interchanged blocks similar in appearance or function (i.e. "Jump" being mistaken for "Move Up"). The confusion usually led students to replace an action or sequence of actions with the first impression of the movement, suggesting that they may only be able to observe a portion of a complex movement. For example, when students replace "Jump" with "Move Up," one might think they should logically also use a "Move Down" block, and sometimes this does happen. Usually, however, the children attended to the initial upward motion of the jump action, mistaking the entire action for their initial impressions of the movement. One hypothesis to explain this pattern could be that child's working memory is simply overloaded (Baddeley 1992), and they cannot even remember the second half of the action or sequence.

Another error turned up regularly in Fix task 2, which required children to circle a "spin right" block. The majority of incorrect responses for this task were submitted by Kindergarteners, who commonly circled "spin left," "hop," and "repeat forever" blocks instead. Children at this age are just learning to differentiate between left and right

directionality, which may explain the confusion of spin left and spin right (Nachshon 1985). The hop and repeat forever block are functionally very dissimilar to spin blocks, but they do all have curved arrows on the block image. This suggests that these types of mistakes are symbol decoding errors.

The trend of students attending to the first action can also be observed in the longer sequenced programs in the Reverse Engineering challenges. Although many students submitted incorrect sequences, almost none of them began their sequence with an incorrect action. For example, in a Solve It sequence that shows “(1) Move Right, (2) Move Left, (3) Jump,” most students chose all of the correct action blocks to include in the program, and also placed the first action in the correct position. However, many students confused the order of the second and third actions. If we return to the overloaded working memory hypothesis, it seems that the child does not have much trouble remembering *which* actions occurred, but *when* they occurred in relation each other.

Although these patterns in errors were not explored in depth in these results, mistakes in assessment performance would be a rich area for future research into children’s programming learning. The finding that children seem to attend most to initial events in a sequence, may have implications for educational interventions that target general sequencing concepts outside of the programming context. For example, in learning to read a story, tie shoelaces, or count to 20, it may be that students would benefit from a learning emphasis on identifying “what comes second,” after mastering the potentially more accessible skill of naming “what comes first.” More research is needed to clarify what is happening in children’s sequencing attempts and how they characterize the order of these relative events.

### Teaching styles in the technology context

Most teachers, regardless of teaching style reported feeling that the ScratchJr intervention had been a success in their classroom because of its open-ended design. The teacher from School A (Low Facilitator, Mid-level all else), who worked with a small, high-performing group of second grade students, reported, “I think the nature of ScratchJr, the process of planning, building, testing, debugging, sharing, and ending up with a pleasing result, all lend themselves to developing the courage and the confidence to take risks, a tolerance for making mistakes, an acceptance of the processes of debugging and problem solving, and a desire to endure to accomplish a goal.” School D’s teacher (High Facilitator and Formal Authority, Mid Personal Model, Low Expert) echoed this sentiment, saying, “we use iPads in our school and I have used other programming apps in the past [...] nothing else approaches the flexibility and intuitive nature of ScratchJr. For us, ScratchJr is another way for students to express themselves creatively.” School E’s teacher (High Formal Authority, Mid all else), who worked with 73 kindergarten students, wrote “ScratchJr has reaffirmed to me why inquiry based learning is so important and impactful.” Many teachers utilized the open-ended nature of the tool in their lesson planning, and the teacher from School B (Mid Expert, High all else) cited her most successful moments as times “when the students figure out new features which we have not even discussed to date, and then begin teaching each other the new features! The energy in the room is contagious!”

There were certainly challenges that teachers faced, including frustration with children spending too much time using certain features of the app to the exclusion of programming elements. The teacher from School D (High Facilitator and Formal Authority, Mid Personal Model, Low Expert) wrote, “Definitely the paint editor was the biggest problem. Several students persistently used all their time to change the color of the [characters]

randomly and wrote no program or an exceedingly basic program.” She also mentioned challenges with Kindergarten students being “off task, needing assistance, and not following directions.” Interestingly, this is a challenge that teachers with lower Formal Authority scores did not report, which perhaps makes sense. The Formal Authority style prioritizes using tools the correct and appropriate way. This teacher’s frustration was in the children’s inclinations to spend time using the app for something other than its main purpose: programming. Even for a teacher with high Facilitator scores, she eventually came to view these paint editor explorations as indulgent, which is in line with the practices of the Formal Authority style. The teacher from School C, who earned Mid-level Formal Authority scores, voiced a similar concern but apparently for a different reason: “It can be challenging for my [Kindergarten] students to make the connection between putting blocks together for fun and actually programming Scratchy Cat to move or accomplish a goal.” Here the children are playing with the blocks, but instead of deeming this behavior off task, the teacher sees indications in their play that the children do not understand key elements of the programming language. This example serves to illustrate how the teaching styles have very subtle impact on teacher’s views of what they deem successful learning and where they see room for improvement. In spite of their differences in assessing learning, however, both of these teachers demonstrate a concern for their students’ understanding of the programming elements of ScratchJr. This suggests that teaching styles have less impact on how teachers prioritize lesson content.

Throughout the curriculum, several teachers reported similar issues with children’s focus and understanding, and many asserted that these issues were caused by the app’s design. The teacher from School B (Mid Expert, High all else) wrote, “students in all grade levels have found the recording feature in ScratchJr and can get easily sidetracked experimenting a little too much with it.” Another teacher (School A, Low Facilitator, Mid all else) mentioned that the children are so engaged with app that they have trouble “breaking away from their device when it is time to stop working.” Approaches to these issues varied. For example, the two teachers from School A and School B developed direct, authoritarian rules, such as defining blocks or features of the app that were off-limits during lessons, or requiring students to physically turn over their iPads when a teacher is talking. This approaches exemplifies the Formal Authority teaching style. The teacher from School F (High Personal Model and Expert, Mid all else) reported that he addressed device-related issues by having children observe a projection of his own screen while he worked on ScratchJr. This solution is a classic demonstration of the Personal Model style. School C’s teacher (High Facilitator, Mid all else) employed a similar approach by altering class structure to incorporate more time projecting students’ projects, but added a conversational component by having children discuss with a classmate what they viewed. Regarding challenges that occur while children use the app, she reported, “I find that if I start by giving each student 5 min of free exploration time [with ScratchJr] before starting a structured activity, they are better able to focus and participate.” These solutions, which address students’ behavioral challenges by acknowledging children’s urges and providing an outlet for them, is a clear tactic of the Facilitator style.

When teachers wrote about their students’ challenges when learning programming, their unique pedagogical approaches became clear. School A’s teacher (Low Facilitator, Mid all else) described the challenge of co-teachers and students “wanting to know *the answer*” to programming challenges (her emphasis). She explained her philosophy on approaching that issue: “Scratch and ScratchJr are very much about exploring, taking risks, testing, debugging, discussing, being persistent. Sometimes there are not quick answers. And most of the time I will not give the students (or the teachers) an answer. The whole point, to me

at least, is to go through the process of trying to figure it out, much like solving a puzzle.” This educator’s concern about developing students’ capacity to work autonomously is a strong hallmark of the Delegator style (recall that all teachers scored High marks for the Delegator style). Based on his teaching style scores, one might expect the teacher from School F (High Expert and Personal Model, Mid all else) to mention similar challenges to the teacher from School A. However, he explained that he wanted to use ScratchJr because “I needed my students to show their math knowledge and to express in a deeper way. Just drawing it out only shows one learned algorithm while using ScratchJr forced them to depict sets and sequencing which shows true understanding.” He also described his biggest challenge as not being able to share projects. He wanted to share his own work that he made to show children examples, and also to be able to share their projects with others outside the class. This focus on modeling, demonstration, and teaching by example aligns well with his High Personal Model score, but not as strongly with his Delegator score. This example demonstrate that the teaching styles are not categorically separated, and that each teacher employs a mix of styles in his or her own teaching practice.

These qualitative findings contextualize the statistically significant relationship between Expert and Facilitator styles in teachers and high programming performance in their students. It also provides context into the non-significant relationship between Formal Authority and Personal Model teaching styles and student programming achievement, as well as the untestable influence of the Delegator style. As discussed, each teacher made choices in their teaching method to address similar challenges presented by the classroom environment, the ScratchJr learning tool, and the limitations and behaviors of their students. The fact that all teachers in our study were able to devote time to this experimental curriculum is evidence of the administrative support and resources at their disposal, and each teacher portrayed exemplary thoughtfulness and intention in designing activities and reflecting on their experiences teaching with ScratchJr.

### **Maximizing programming achievement in early childhood: teaching styles for success**

The results of this study reveal that computer programming is one learning domain where children have demonstrated high achievement when the teacher employs Facilitation and Expert modeling to strengthen student learning. At this point, it is worth taking a moment to unpack further the characteristics of these two teaching styles, as described by Grasha (1994).

Facilitation differs from other styles in that it emphasizes the personal nature of the teacher-student relationship (1994). Facilitators do not readily offer correct answers, but instead guide students toward discovery in projects by asking questions, proposing and exploring a diversity of approaches, and encouraging students to “develop criteria to make informed choices” (1994). Teachers position themselves a bit like consultants or co-designers on a project, employing the scientific method in an exploratory, information gathering way, and eventually encouraging students toward independent choices and responsibility, based on their investigations. Additionally, this style is defined by a warm interpersonal relationship, with teachers viewing themselves as a source of support, encouragement, and motivation for children as they work. The benefits to this style seem readily apparent: the educators are flexible, allowing time in the curriculum to focus on student’s needs and goals, and because they are always willing to explore options and linger on questions, students know they have teacher’s support and implicit permission to investigate questions about which they are naturally curious. However, all teaching styles



are tempered by outside constraints. Challenges with this style often result from the time consuming nature of the approach. Some children may be ready to move on to different topics while others are still exploring a diversity of solutions to the first project. Additionally, children may be uncomfortable with this style "if it is not used in a positive and affirming manner," for example if children's chosen solutions are passed over in favor of more investigation to reach the "correct" answer. Finally, the application to lesson content should be considered, as a more direct approach may occasionally be more effective and time-efficient.

Expert modeling is another style that has demonstrable effects in the computer programming domain. Experts, according to Grasha (1994), are so-called because they possess and disseminate the knowledge and experience that students need to complete projects. Experts strive to maintain their status as the most knowledgeable among the students by displaying detailed information, sometimes unsolicited by students. Although this may not immediately sound like it would foster independent students, the goal of using this style is not to command respect, but to model what an expert in the field looks like, and to challenge students to cultivate their own expertise and competence. Experts are as concerned with transmitting correct knowledge as they are with ensuring that students are correctly understanding the content and are well prepared to reiterate and apply knowledge themselves. The obvious benefit of this teaching style is that the educator is extremely well-informed about the subject matter. In an educator, very little can replace an explanation from first-hand knowledge. However, this style may be overused, in which case the display of knowledge can intimidate and silence students who are still building their confidence. Additionally, this style is much more effective when educators show the underlying thought processes that they used to produce their solutions, rather than simply producing a perfect solution with no clear explanation of how it was derived.

Together, these styles are powerful tools for a programming instructor who wishes to foster the best learning outcomes in young children. The approach of being personable, flexible, and responsive to student needs, in combination with a firm grounding of the content matter and an eagerness to explain it, has been shown to result in higher learning outcomes for young children exploring computer programming.

## Limitations

In the course of a study this large-scale and comprehensive, there are certainly methodological and analytic challenges that shape the research process. It was an explicit goal of the research team to allow teachers the freedom to do exactly as they pleased in their normal school routine. Using naturalistic learning settings necessarily results in more complications in any study design, but it was an intentional choice. The research team hoped to take a representative look at the kind of results and variability that district leaders and policy-makers might expect in a large-scale ScratchJr intervention.

The most salient limitation in terms of the teaching style data is lack of diversity among the sample. This is partly due to the method of recruitment (volunteer based on email alerts), which resulted in a self-selected sample of educators already curious about and experienced with technology. Most teachers reported that they had already brought other computer programming and robotics tools into their early childhood classrooms, indicating that their students also enjoy a level of exposure not shared by the majority of their peers at the national level. Additionally, almost none of the volunteer teachers were from public

schools, and instead worked at private or chartered schools, some of them overtly emphasizing science, technology, engineering, and mathematics (STEM) education. These similarities were evident in the data. At times, the teaching style results were convergent to the point that comparisons could not be made (e.g. all of the teachers earned the same High scores for the Delegator teaching style). However, the research team was surprised to find statistically significant differences even among this fairly homogenous school and teacher sample. Teacher's instructional style alone accounts for about 12% of the variance in student programming scores in this sample. The research team hypothesizes that with a more diverse sample of teachers, it would be easier to draw generalizable conclusions about the effects of teaching style on student programming outcomes, and that these effects would be more pronounced.

A second limitation was in the collection process for the children's programming scores data. Different schools and teachers administered the assessment in slightly different ways, for example, by changing the time allowed for responses, allowing more or fewer replays of the videos, and most importantly, omitting some questions from the assessment. Some teachers deemed certain questions too advanced for their students and simply did not administer portions of the Solve Its. For example, the teacher from School E (High Formal Authority, Mid all else) did not administer Solve Its beyond the most basic Fix tasks with her youngest students because she reported feeling that ScratchJr was "pushing the limits" of her Kindergarteners. Indeed, she may have been correct, since they showed the lowest performance while most children across grade and school earned very high scores on the Fix tasks. The challenge of missing data in student assessments was addressed by examining task- and grade-specific trends rather than sub-sample trends. In cases of missing data, student responses were graded out of the possible points that a child could earn on only those sections that were administered by their teacher, and not out of the possible points of the entire assessment.

### **Summarizing the findings**

The results presented in this paper offer a promising top-down look at the implementation of ScratchJr with over 200 young children in early childhood classrooms. Teachers were enthusiastic about introducing the tool, and were able to adapt its use in a variety of contexts. All teachers reported that their students were enthusiastic and engaged in ScratchJr lessons, and its open-ended, flexible design allowed many children to continue to explore coding with their own personal goals. Finally, the Solve It programming assessment results suggest that students in Kindergarten through second grade are able to grasp concepts of sequencing and symbol decoding, as well as understand processes of debugging and goal-oriented programming.

It is the opinion of the research team that these concepts and processes demonstrated by students on the Solve It assessment reflect early computational thinking strategies for children from ages 5 to 8 years old. The findings presented here can be summarized into the following key points, which may be used to guide researchers and educators hoping to implement programming interventions in early childhood settings.

*In our sample, teaching habits associated with Expert and Facilitator styles was correlated with higher programming achievement in students.* Results from this study revealed that teaching habits of Experts and Facilitators were strong predictors of children's understanding and performance regarding the ScratchJr programming language. Children benefited from teachers who showed flexibility in their teaching styles by prioritizing student's goals and needs over their own urge to follow a pre-arranged lesson plan.

Additionally, modeling quality content knowledge and pushing children to cultivate their own coding expertise was associated with the strongest programming learning outcomes in young children.

*Regardless of teaching style, children in our sample were able to grasp concepts of symbol decoding and sequencing, and to demonstrate practices of goal-oriented programming and debugging.* In eight of the eleven tasks, a clear ceiling effect was noted in all grades and in most schools. Kindergarten through second grade students in our sample were able to associate certain programming blocks with specific actions, and they could comprehend the strategy of isolating a missing or extraneous action in a coded sequence.

*Children show developmental differences in programming comprehension, with younger children achieving less comprehension than children even 1 or 2 years older.* The strongest pattern that emerged from our Solve Its data set was the trend for scores to improve from Kindergarten to first grade, and from first grade to second grade. This could be caused by a number of cognitive, social, or personal developmental factors, but the trend was significant across our sample and is consistent with previous work involving the Solve Its assessment (Strawhacker and Bers 2015; Sullivan and Bers 2016).

An area for future work may be to define and examine early or pre-computational thinking strategies, as the evolving literature on this subject currently applies mainly to older students, and does not capture aspects of developing programming skills that may be evident in early childhood settings. The research team looks forward to pursuing work with teachers and students in naturalistic settings in order to discover and expand on existing computational thinking frameworks, and to more deeply understand children's development of computational thinking and learning skills in a variety of environments.

## Conclusion

This study has focused on children's programming comprehension using ScratchJr in their normal classroom settings. The methodological choice of this study to examine the learning outcomes of naturalistic programming lessons, yielded a larger than average student sample ( $N = 222$ ) for this kind of pilot, in-classroom research. Results showed that children in all schools and in all grades showed mid-to-high levels of ScratchJr programming comprehension on Solve It Tasks that assessed programming concepts and skills of symbol decoding, sequencing, debugging, and goal-oriented programming (Brennan and Resnick 2012). Statistically significant differences were observed among children across the sample in different grades, with older grades outperforming younger ones. Significant differences were also observed across all  $N = 6$  schools. Although our sample was fairly homogenous, the data still reveal a statistically significant relationship between certain teaching styles and high programming achievement. Specifically, Facilitation and Expert modeling by teachers, which emphasized content mastery, student-directed learning, and open exploration within the ScratchJr environment, was predictive of high achievement in programming learning outcomes in students. The research team feels strongly that the results presented here should not be taken as a condemnation of the teaching approaches that were not associated with high programming achievement in students. Indeed, it would be a fallacy to do so, since each teacher demonstrated all of the teaching styles to varying degrees. Although the Formal Authority and Personal Model styles were not predictive of programming learning, it is beyond the scope of this paper to say whether emphasizing these styles is predictive of high achievement in other domains. However, in the context of

computer programming, the results show that a rigid teaching style where the teacher demonstrates before children experiment is not as effective for student learning as when teachers present themselves to students as flexible, supportive resources of knowledge, who are there to facilitate children's own programming explorations and expertise development.

Although it will come as no surprise to developmental specialists, it is important to conclude from this study that even diverse domains of foundational computer science, early numeracy and literacy, and storytelling, children can benefit from a teaching approach that prizes self-discovery and collaborative learning models (Vygotsky 1978; Edwards et al. 1993). This finding is very much in line with Seymour Papert's theory of Constructionism, which emphasized the importance of children's self-directed creation with technology as pivotal to the development of their learning—learning about themselves, learning about powerful ideas bigger than simple technology (such as democracy, or fractions), and learning about the “language” of digital media itself (Papert 1980, 2000). Early childhood educators, who have an arsenal of teaching tactics at their disposal built through their own hands-on experience working with children, can take these results as a guidepost in helping them choose the most effective “tool from their educational toolkit” when exploring programming with young children. Designers and developers of educational tools should consider the extreme learning value in creating tools that are open-ended enough to foster child-directed, creative play experiences, and are also robust to diverse learning settings and varied pedagogical contexts.

## References

- Baddeley, A. (1992). Working memory. *Science*, 255(5044), 556–559. doi:[10.1126/science.1736359](https://doi.org/10.1126/science.1736359).
- Bebell, D., Russell, M., & O'Dwyer, L. (2004). Measuring teachers' technology uses: Why multiple-measures are more revealing. *Journal of Research on Technology in Education*, 37(1), 45–63.
- Bers, M. (2008). *Blocks to robots: Learning with technology in the early childhood classroom*. New York, NY: Teachers College Press.
- Bers, M. U. (2012). *Designing digital experiences for positive youth development: From playpen to playground*. Oxford: Cary, NC.
- Bers, M. & Kazakoff, E. (2012). Chapter 26. Developmental technologies: Technology and human development. In R. M. Lerner, M. A. Easterbrooks, J. Mistry, & I. B. Weiner (Eds.), *Handbook of psychology, developmental psychology*. Thousand Oaks, CA.
- Bers, M. U., Seddighin, S., & Sullivan, A. (2013). Ready for robotics: Bringing together the T and E of STEM in early childhood teacher education. *Journal of Technology and Teacher Education*, 21(3), 355–377.
- Blazer, C. (2008). *Literature review: Educational technology*. Research services, Miami-Dade County Public Schools. Miami-Dade: Research Services Office of Assessment, Research, and Data Analysis Miami-Dade County Public Schools.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings from AERA '12: The 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada.
- Chen, C. (2008). Why do teachers not practice what they believe regarding technology integration? *The Journal of Educational Research*, 102(1), 65–75. doi:[10.3200/JOER.102.1.65-75](https://doi.org/10.3200/JOER.102.1.65-75).
- Chikofsky, E., & Cross, J. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1), 13–17.
- Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology*, 76(6), 1051–1058. doi:[10.1037/0022-0663.76.6.1051](https://doi.org/10.1037/0022-0663.76.6.1051).
- Cuban, L., Kirkpatrick, H., & Peck, C. (2001). High access and low use of technologies in high school classrooms: Explaining an apparent paradox. *American Educational Research Journal*, 38(4), 813–834. doi:[10.3102/00028312038004813](https://doi.org/10.3102/00028312038004813).

- Cuny, J., Snyder, L., & Wing, J. M. (2010). *Demystifying computational thinking for noncomputer scientists*. Unpublished manuscript in progress, referenced in <http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>.
- Edwards, C., Gandini, L., & Forman, G. (Eds.). (1993). *The hundred languages of children*. Norwood, NJ: Ablex.
- Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97.
- Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th international conference on interaction design and children (IDC '13)* (pp. 1–10). New York, NY: ACM. doi: [10.1145/2485760.2485785](https://doi.org/10.1145/2485760.2485785).
- Ford, J. H., & Il, J. R. (2015). Teaching and learning styles in quality improvement: Identification and impact on process outcomes. *Addiction Science & Clinical Practice*, *10*(Suppl 1), A12.
- Ginsburg, H. P. (1997). *Entering the child's mind: the clinical interview in psychological research and practice*. New York: Cambridge University Press.
- Grasha, A. F. (1994). A matter of style: The teacher as expert, formal authority, personal model, facilitator, and delegator. *College Teaching*, *42*(4), 142–149.
- Grasha, A. F., & Riechmann-Hruska, S. (1996). *Teaching style survey*. Retrieved from <http://longleaf.net/teachingstyle.html>.
- Grasha, A. F., & Yangarber-Hicks, N. (2000). Integrating teaching styles and learning styles with instructional technology. *College Teaching*, *48*(1), 2–10. doi:[10.1080/87567550009596080](https://doi.org/10.1080/87567550009596080).
- Hew, K. F., & Brush, T. (2007). Integrating technology into K-12 teaching and learning: Current knowledge gaps and recommendations for future research. *Educational Technology Research and Development*, *55*(3), 223–252. doi:[10.1007/s11423-006-9022-5](https://doi.org/10.1007/s11423-006-9022-5).
- Hofer, M., Chamberlin, B., & Scot, T. (2004). Fulfilling the need for a technology integration specialist. *The Journal*, *32*(3), 34.
- Howard, S. K. (2013). Risk-aversion: Understanding teachers' resistance to technology integration. *Technology, Pedagogy and Education*, *22*(3), 357–372. doi:[10.1080/1475939X.2013.802995](https://doi.org/10.1080/1475939X.2013.802995).
- Jeon, L., Buettner, C. K., & Hur, E. (2014). Examining pre-school classroom quality in a state-wide quality rating and improvement system. *Child & Youth Care Forum*, *43*(4), 469–487.
- Kazakoff, E., Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, *41*(4), 245–255. doi:[10.1007/s10643-012-0554-5](https://doi.org/10.1007/s10643-012-0554-5).
- Keengwe, J., & Onchwari, G. (2011). Fostering meaningful student learning through constructivist pedagogy and technology integration. *International Journal of Information and Communication Technology Education*, *7*(4), 1–10.
- Koschmann, T. (1996). *CSCL, theory and practice of an emerging paradigm*. Mahwah, NJ: L. Erlbaum Associates.
- Kunter, M., Baumert, J., & Koller, O. (2007). Effective classroom management and the development of subject-related interest. *Learning and Instruction*, *17*(5), 494–509.
- Lee, M. S. C. (2015). *Teaching Tools, Teachers' Rules: ScratchJr in the Classroom*. (Unpublished master's thesis). Medford, MA: Tufts University.
- Limongelli, C., Lombardi, M., Marani, A., & Sciarrone, F. (2013). A teacher model to speed up the process of building courses. In *Human-computer interaction. applications and services* (pp. 434–443). Berlin: Springer.
- Lin, P. C., Lu, H. K., & Liu, C. H. I. A. (2013). Towards an education behavioral intention model for e-learning systems: An extension of UTAUT. *Journal of Theoretical and Applied Information Technology*, *47*(3), 1120–1127.
- Mashburn, A. J., Pianta, R. C., Hamre, B. K., Downer, J. T., Barbarin, O. A., Bryant, D., et al. (2008). Measures of classroom quality in prekindergarten and children's development of academic, language, and social skills. *Child Development*, *79*(3), 732–749. doi:[10.1111/j.1467-8624.2008.01154.x](https://doi.org/10.1111/j.1467-8624.2008.01154.x).
- Nachshon, I. (1985). Directional preferences in perception of visual stimuli. *International Journal of Neuroscience*, *25*(3–4), 161–174.
- National Assessment of Educational Progress. (2014). *Technology and engineering literacy framework for the 2014 NAEP*. National Assessment Governing Board.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
- Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal*, *39*(3/4), 720.
- Petrina, S. (1998). The politics of research in technology education: A critical content and discourse analysis of the *Journal of Technology Education*, *10*(1–8). doi:[10.21061/jte.v10i1.a.3](https://doi.org/10.21061/jte.v10i1.a.3).

- Portelance, D. J., & Bers, M. U. (2015). Code and tell: Assessing young children's learning of computational thinking using peer video interviews with ScratchJr. In *Proceedings of IDC '15: The 14th international conference on interaction design and children*. Boston, MA: ACM.
- Pretz, K. (November 21, 2014). Computer science classes for kids becoming mandatory. *The institute*. <http://theinstitute.ieee.org/career-and-education/preuniversity-education/computer-science-classes-for-kids-becoming-mandatory>.
- Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60–67. doi:10.1145/1592761.1592779.
- Resnick, M., & Silverman, B. (2005). Some reflections on designing construction kits for kids. In *Proceedings of IDC '05: The 4th international conference on interaction design and children*. New York, NY: ACM.
- Rimm-Kaufman, S., Curby, T. W., Grimm, K. J., Nathanson, L., & Brock, L. L. (2009). The contribution of children's self-regulation and classroom quality to children's adaptive behaviors in the kindergarten classroom. *Developmental Psychology*, 45(4), 958–972. doi:10.1037/a0015861.
- Ringstaff, C., & Kelly, L. (2002). *The learning return on our educational technology investment: A review of findings from research*. San Francisco, CA: WestEd RTEC.
- Schmidt, D. A., Baran, E., Thompson, A. D., Mishra, P., Koehler, M. J., & Shin, T. S. (2009). Technological pedagogical content knowledge (TPCK): The development and validation of an assessment instrument for preservice teachers. *Journal of Research on Technology in Education*, 42(2), 27.
- Strawhacker, A. L., & Bers, M. U. (2015). "I want my robot to look for food": Comparing children's programming comprehension using tangible, graphical, and hybrid user interfaces. *International Journal of Technology and Design Education*, 25(3), 293–319.
- Strawhacker, A., Lee, M., Caine, C., & Bers, M. U. (2015). ScratchJr Demo: A coding language for Kindergarten. In *Proceedings of the 14th international conference on interaction design and children (IDC '15)*. Boston, MA: ACM.
- Sullivan, A., & Bers, M. U. (2016). Robotics in the early childhood classroom: Learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade. *International Journal of Technology and Design Education*, 26(1), 3–20.
- Technology for All Americans Project, & International Technology Education Association. (2000). *Standards for technological literacy: Content for the study of technology*. International Technology Education Association.
- US Census Bureau. (2015, September 24). *State & county QuickFacts*. <http://quickfacts.census.gov/qfd/states/00000.html>.
- US Department of Education. (2010, November 1). *Transforming American education: Learning powered by technology*. <https://www.ed.gov/sites/default/files/netp2010.pdf>.
- US Department of Education. (2013, June). *Private elementary and secondary enrollment, number of schools, and average tuition, by school level, orientation, and tuition: Selected years, 1999–2000 through 2011–2012*. [https://nces.ed.gov/programs/digest/d13/tables/dt13\\_205.50.asp](https://nces.ed.gov/programs/digest/d13/tables/dt13_205.50.asp).
- US Department of Education, National Center for Education Statistics (2015a). *2013–2014 Private school universe survey* [Data File]. <https://catalog.data.gov/dataset/201314-private-school-universe-survey>
- US Department of Education, National Center for Education Statistics. (2015b). *Common core of data* [Data File]. <https://nces.ed.gov/ccd/ccddata.asp>.
- Vygotsky, L. S. (1978). Interaction between learning and development (M. LopezMorillas, Trans.). In M. Cole, V. John-Steiner, S. Scribner, & E. Souberman (Eds.), *Mind in society: The development of higher psychological processes* (pp. 79–91). Cambridge, MA: Harvard University Press.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf>.