Journal of Childhood, Education & Society
Volume 1, Issue 1, 2020, pp. 63-77
DOI: 10.37291/2717638X.20201132

ISSN: 2717-638X
Research Article

Early childhood preservice teachers' debugging block-based programs: An eye tracking study

Lucas Vasconcelos[*], **Ismahan Arslan-Ari**[*], **Fatih Ari**[***]

Abstract: Learning computational skills such as programming and debugging is very important for K-12 students given the increasing need of workforce proficient in computing technologies. Programming is an intricate cognitive task that entails iteratively creating and revising programs to create an artifact. Central to programming is debugging, which consists of systematically identifying and fixing program errors. Given its central role, debugging should be explicitly taught to early childhood preservice teachers so they can support their future students' learning to program and debug errors. In this study, we propose using eye-tracking data and cued retrospective reporting to assess preservice teachers' cognitive strategies while debugging. Several eye-tracking studies have investigated learners' debugging strategies though the literature lacks studies (a) conducted with early childhood preservice teachers and (b) that focus on block-based programming languages, such as Scratch. The present study addresses this gap in the literature. This study used mixed methods to triangulate quantitative findings from eye movement analysis and qualitative findings about employed debugging strategies into the creation of descriptive themes. Results showed that participants developed strategies such as simultaneous review of output and code, use of beacons to narrow down the area to be debugged, and eye fixation on output to form hypotheses. But most often, debugging was not informed by a hypothesis, which led to trial and error. Study limitations and directions for future research are discussed.

Article History
Received: 22 December 2019
Accepted: 27 January 2020

Keywords
Debugging;
Block-based programs;
Eye tracking;
Cognitive strategies;
Early childhood preservice teachers

Introduction

Learning computational skills such as programming is important for K-12 students given the increasing need for a workforce proficient in coding and computing technologies (Burke, 2012; K-12 Computer Science Framework Steering Committee, 2016; Obama, 2016). Various resources have been created to inspire programming instruction in K-12 classrooms such as low threshold block-based programming languages (Bau, 2015; Cooper, Dann, and Pausch, 2000; Resnick et al., 2009) that are easy and appealing to youth, open access courses for teachers on computer science (Code.org, n.d.; Google, n.d.), exemplary instructional materials for integrating programming into subject areas (Computer Science Teachers Association & International Society for Technology in Education, 2011; Project Growing Up Thinking Scientifically (GUTS), n.d.), among others. However, little has been done as professional learning on programming in early childhood preservice teacher preparation programs (Kim, Yuan, Vasconcelos, Shin, and Hill, 2018).

Training preservice teachers to program involves not only teaching how to successfully apply computer science concepts and commands, but also fostering important computational thinking practices

[*]University of South Carolina, College of Education, Department of Educational Studies, Columbia, USA, email: limadel@mailbox.sc.edu, ORCID: https://orcid.org/0000-0001-9074-203X

[*]University of South Carolina, College of Education, Department of Educational Studies, Columbia, USA , email: arslanai@mailbox.sc.edu, ORCID: https://orcid.org/0000-0003-3497-7058

[***]University of South Carolina, College of Education, Department of Educational Studies, Columbia, USA, email: arifatih@mailbox.sc.edu,ORCID: https://orcid.org/0000-0001-9512-1312

such as debugging (Basu, 2016; Brennan and Resnick, 2012; Vasconcelos and Kim, 2019). Debugging is a fundamental part of programming that entails addressing program errors or an unintended output on the screen (McCauley et al., 2008; Yen, Wu, and Lin, 2012). Training on programming and debugging is essential for early childhood preservice teachers to properly support their future students' learning to program and debug errors (Kim et al., 2018).

Debugging

Programming is an intricate cognitive task (Vihavainen, Airaksinen and Watson, 2014) in which one combines units that encapsulate specific concepts and commands from a computer science language to create an output. Given the complexity of this task, it is unlikely that one will create a program that does not need revisions in one attempt. Therefore, it is critical to explicitly teach error debugging skills during programming instruction.

Debugging, also known as troubleshooting, is defined as the process of identifying error(s) in a program and using problem-solving strategies to fix it (McCauley et al., 2008; Proctor, 2019; Yen et al., 2012). Debugging is an inherent part of programming, and programmers spend a significant amount of time doing it (Alqadi and Maletic, 2017; Beller, Spruit, Spinellis, and Zaidman, 2018). Different from random trial and error, debugging is a systematic and thoughtful process in which one tests hypotheses and applies strategies to locate and overcome the cause of a program error (Kim et al., 2018; Shute, Sun, and Asbell-Clarke, 2017). Despite its importance, explicit instruction about debugging strategies is rarely featured in computer science instruction (Proctor, 2019).

Debugging for early childhood preservice teachers. Teaching K-12 students to program has emerged as a crucial instructional goal for school teachers (Kalelioğlu, 2015; Kazimoglu, Kiernan, Bacon, and MacKinnon, 2012; Lye and Koh, 2014). And yet, preservice and in-service teacher education programs are still in need of a nationwide curriculum that supports integration of programming and other computational skills such as debugging at the K-12 level (Paul, 2016). Learning to program and debug program errors can be a daunting task for preservice teachers, especially if they are novice learners with limited to no background in computing. This is because novice learners tend to overestimate the complexity of programming tasks, encounter a higher number of program errors, and consequently experience decreased motivation towards learning to program (Isong, 2014; Sun and Hsu, 2019; Yukselturk and Altiok, 2017).

Integrating programming and debugging into K-12 teaching does not entail adding another component to the curriculum. It can help young learners develop crosscutting concepts and skills such as abstraction, conditional logic, and pattern identification (Grover and Pea, 2013; Sengupta, Kinnebrew, Basu, Biswas, and Clark, 2013). Empirical studies on debugging strategies used by early childhood preservice teachers are limited. Recent studies found that preservice teachers who are learning to program struggle at systematically forming and testing hypotheses to guide their debugging as well as explain the cause of an error even when a problem is fixed (Kim et al., 2018). In another study, preservice teachers ended up simplifying a program by removing a problematic area to avoid problem solving (Kim et al., 2016). Further empirical studies are needed.

Professional learning on programming and debugging for preservice teachers may have a crucial impact on their future students, such as providing underprivileged populations (e.g., females, students with special needs, people of color) with access to STEM learning experiences, and influencing students' dispositions to advance their education and pursue jobs in STEM fields (Leonard et al., 2016; National Research Council, 2011).

Use of eye movements in programming education. Research that uses eye-tracking devices to assess learning in computer science has increased in recent years. A survey on the use of eye-tracking in programming instruction research revealed that program comprehension and debugging are two mostly studied areas (Obaidellah, Al Haek, and Cheng, 2018). Among these studies, several have used eye movement data to assess "learners' problem-solving processes objectively" (Sun and Hsu, 2019, p. 67) as learners attempt to understand and/or debug programs. For instance, Lin et al. (2015) examined university students' cognitive processes during debugging tasks. The study found that novice learners in programming followed a linear, line-by-line approach as they debugged computer programs whereas students with prior programming experience followed a more logical, strategic approach. In a similar study, Bednarik (2012) used eye movement data to investigate visual attention patterns as a function of expertise during debugging. Findings showed novices' eye transitions between code and output areas early in the debugging task but later on they focused on the program itself. Alternatively, experts demonstrated systematic eye transitions between the code and output with focused attention on the output area throughout the debugging task. Further, Papavlasopoulou, Sharma, Giannakos, and Jaccheri (2017) used eye-tracking data to examine children's learning processes of coding during block-based programming activities. They grouped children into two groups: (1) ages of 8-12 and (2) ages of 13-17, and compared visual attention patterns, time spent on Areas of Interests (AOIs), and transitions between AOIs. Results indicated that younger children focused mostly on sprites, the visual aspects of the programming tasks, whereas older children focused mostly on script, output, and command areas. Papavlasopoulou et al. (2017) also asserted that a higher number of transitions between these areas indicate two types of processes, active debugging and hypothesis testing.

Eye tracking devices have been used not only as a data collection method, but also as part of an intervention to support programming education. Sun and Hsu (2019) implemented an eye-tracking scaffolding system that instantly gauged learners' attention by providing just-in-time hints as learners worked on programming tasks. The system tracked participants' eye movements (i.e., fixation positions and durations) to evaluate the level of attention. If, for instance, a participant did not fixate on the area that contained key information, then the system would highlight the area to direct the participant's attention. Compared to peer scaffolding and mixed scaffolding, learners using the eye tracking scaffolding demonstrated higher programming self-efficacy. However, no difference was found in terms of learning performance between experimental conditions. In another study, Bednarik, Schulte, Budde, Heinemann, and Vrzakova (2018) explored the effect of eye movement modeling examples on program comprehension and program reading. Researchers recorded eye movements of an expert programmer when s/he was working on programming tasks and used the video as a model to support novice learners' program comprehension and program reading. Findings revealed significant improvements in novice learners' program comprehension.

These studies demonstrate the potential of using eye trackers to understand learners' cognitive processes during complex programming tasks. However, most research has been conducted with higher education individuals from the field of computer science, and involved text-based programming languages (Obaidellah et al., 2018). To our knowledge, the present study is the first attempt to use eye tracking to understand preservice teachers' cognitive processes during debugging block-based programs. Hence, the present study addresses this gap in the research literature.

Purpose and Research Question

This study examined how early childhood preservice teachers used cognitive strategies while debugging block-based programs using eye movement data and cued retrospective reporting. This research question was investigated: What cognitive strategies do early childhood preservice teachers use during debugging block-based programs?

Method

This was a mixed methods case study (Leedy and Ormrod, 2013) in which we used qualitative and quantitative data to provide an in-depth understanding of early childhood preservice teachers' cognitive processes while debugging block-based programs. Quantitative and qualitative data were concurrently collected, had equivalent weight (Leech and Onwuegbuzie, 2009), and consisted of eye movement data and cued retrospective reporting transcripts respectively.

Research Setting

After approval by an Institutional Review Board (IRB), data was collected in two sections of a course on early childhood mathematics teaching offered at a large Southeastern university in the United States. One of the week-long modules of this course covers STEM education in early childhood. During this week, researchers hosted a 2-hour workshop on block-based coding during a class meeting in a computer lab. A total of 41 preservice teachers attended the workshop, 19 in course section A and 22 in course section B. During the workshop, preservice teachers learned to program with Scratch, a free block-based visual programming tool and language. Each preservice teacher worked individually in an assigned computer.

After an introduction about Scratch features (e.g., block palette, output), preservice teachers followed step-by-step instructions to create an animation in Scratch such as making a sprite fly and coding a knock-knock joke. Next, they completed two debugging activities in which they attempted to identify and fix a problem in a faulty program. They debugged the code to make a sprite move between two points on the screen and to make a sprite travel on a square pattern.

At the end of the workshop, preservice teachers were invited to participate in an individual data collection session about coding and debugging. A gift card was offered as incentive though only two participants, one from each course section, accepted to join the individual session.

Participants

Participants were two 21-year old female preservice teachers in the senior year of their Bachelor's degree in early childhood education. Pseudonyms are used in this paper. Convenience sampling was used given that these were the two preservice teachers who accepted to partake in individual activities after the workshop. Regarding ethnicity, Mila was White, and Emmy was African American. Prior to attending the Scratch workshop, neither one had previous experience with text- or block-based programming languages. According to a self-rated prior knowledge test administered before individual debugging activities, both participants reported that they knew the functions of most of the blocks used in the debugging activities.

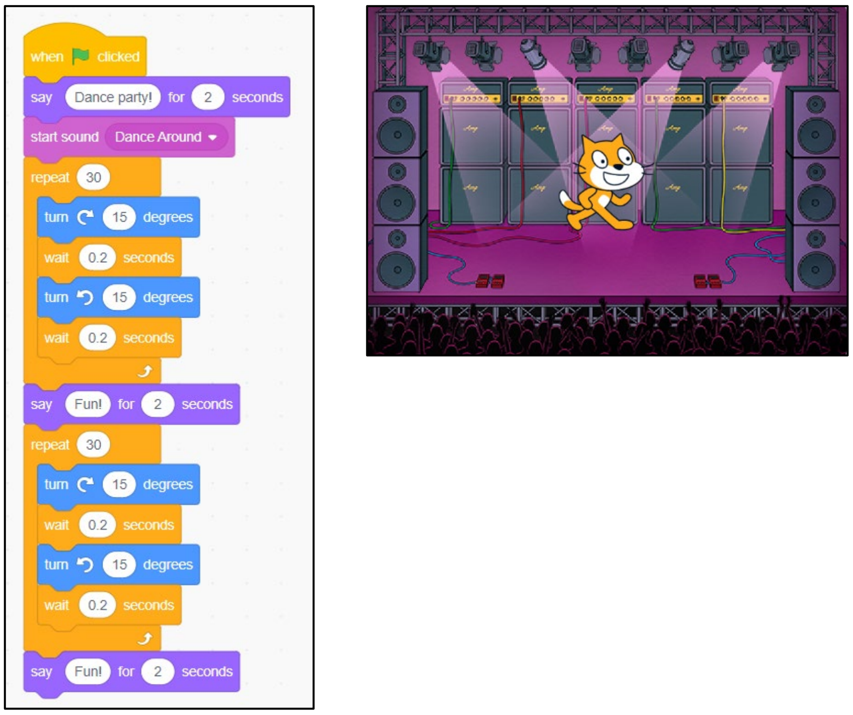
Study Procedures

Participants attended one individual data collection session in a human computer interaction lab. First, each participant was explained the purpose of the study, and their informed consent was collected. They were introduced to the eye tracking equipment (Tobii X3-120), which was attached to a 22 in computer monitor. Then, participants' eye movements were calibrated by following a red ball, called calibration dot, on the screen. This calibration adjusts the eye tracking system to the geometric characteristics of participants' eyes. After calibration, participants completed the first debugging activity in Scratch. While working on debugging activities, eye movements were tracked and video recorded. Participants had 10 minutes to identify and fix the fault in the code. Next, researchers replayed participant's eye movements and conducted cued retrospective reporting. Cued retrospective reporting was video recorded with the Tobii Pro Studio software. The same procedure was repeated for a second debugging activity.

Individual Debugging Activities. Two debugging activities were designed based on the content covered in the workshop. Ten minutes were allocated to each debugging activity, and participants worked independently, without validation from researchers. Participants were informed that they could add, delete, or reorder blocks, as well as undo their own actions to start over.

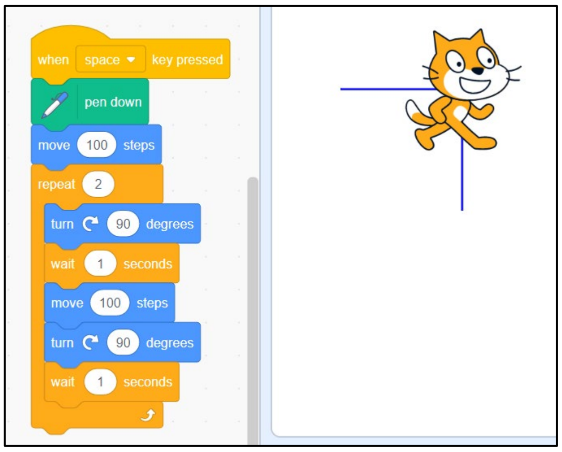
Instructions for the first debugging activity were "Liam wants to code his cat to dance until the end of the song, but he noticed the cat continues dancing after the song stops. Can you help Liam fix the bug in the code?". Figure 1 presents the faulty program and output. Reducing the number of loops in one of the repeat blocks would solve this problem.

In the second debugging activity, participants were instructed: "Lucia would like the cat to move following a rectangle path, though the code that she put together is not working. Let's help Lucia fix this code." Figure 2 shows the faulty program and output. To debug the error, participants could place the first move block inside the loop (repeat) right before the first turn block, and increase the number of steps in one of the move blocks.



```
when [flag] clicked
say Dance party! for 2 seconds
start sound Dance Around
repeat 30
    turn ↻ 15 degrees
    wait 0.2 seconds
    turn ↺ 15 degrees
    wait 0.2 seconds
say Fun! for 2 seconds
repeat 30
    turn ↻ 15 degrees
    wait 0.2 seconds
    turn ↺ 15 degrees
    wait 0.2 seconds
say Fun! for 2 seconds
```

Figure 1. First Debugging Activity in Individual Coding Section



```
when space key pressed
pen down
move 100 steps
repeat 2
    turn ↻ 90 degrees
    wait 1 seconds
    move 100 steps
    turn ↻ 90 degrees
    wait 1 seconds
```

Figure 2. Second Debugging Activity in Individual Coding Section

Data Collection Methods

To collect data that answers the research question about early childhood preservice teachers' cognitive strategies during debugging block-based programs, an eye tracking device and cued retrospective reporting were used. Also, participants' prior knowledge of programming was measured with a prior knowledge test.

Prior knowledge test. To assess prior knowledge of programming, participants rated 12 statements about their knowledge of specific blocks on a 5-point scale ranging from "I don't know at all" (score 1) to "I know very well" (score 5). Statements targeted blocks used in debugging activities. For example, "I know the functions of *repeat* block." and "I know the functions of *when I receive* block." Moreno and Mayer (1999) also used a similar self-assessment instrument on a different topic.

Eye tracking device. While working on the debugging activity, participants' eye movements were recorded with a Tobii X3-120 eye tracking device using the sample rate of 120 Hz. Tobii X3-120 is a screen-based portable eye tracker that attaches to the bottom of a computer monitor. It uses near infrared sensors to capture eye gaze. Tobii Pro Studio software was used to calibrate, record, and analyze eye movements.

Cued retrospective reporting. Cued retrospective reporting is a verbal reporting technique in which the participant watches their recorded eye movements and retrospectively verbalizes what they were thinking as well as their problem-solving strategies. This cued retrospective reporting technique is adopted from van Gog, Paas, van Merriënboer, and Witte (2005). Participants were told "Please watch the recording of your eye movements and tell me what you were thinking during debugging." While watching their eye movements, participants were asked questions, such as "Where in the screen did you look to fix the error?" and "What was the cause of the problem in the debugging activity?" Researchers paused or replayed the video recording as needed. Participants' reporting was video recorded with the retrospective reporting feature in the Tobii Pro Studio software.

Data Analysis Methods

Tobii Pro Studio software was used to analyze participants' eye movements. First, eight specific sections of the screen were assigned as Area of Interests (AOIs), and they remained constant throughout debugging activities (see Figure 3). Description of AOIs are provided in Table 1.

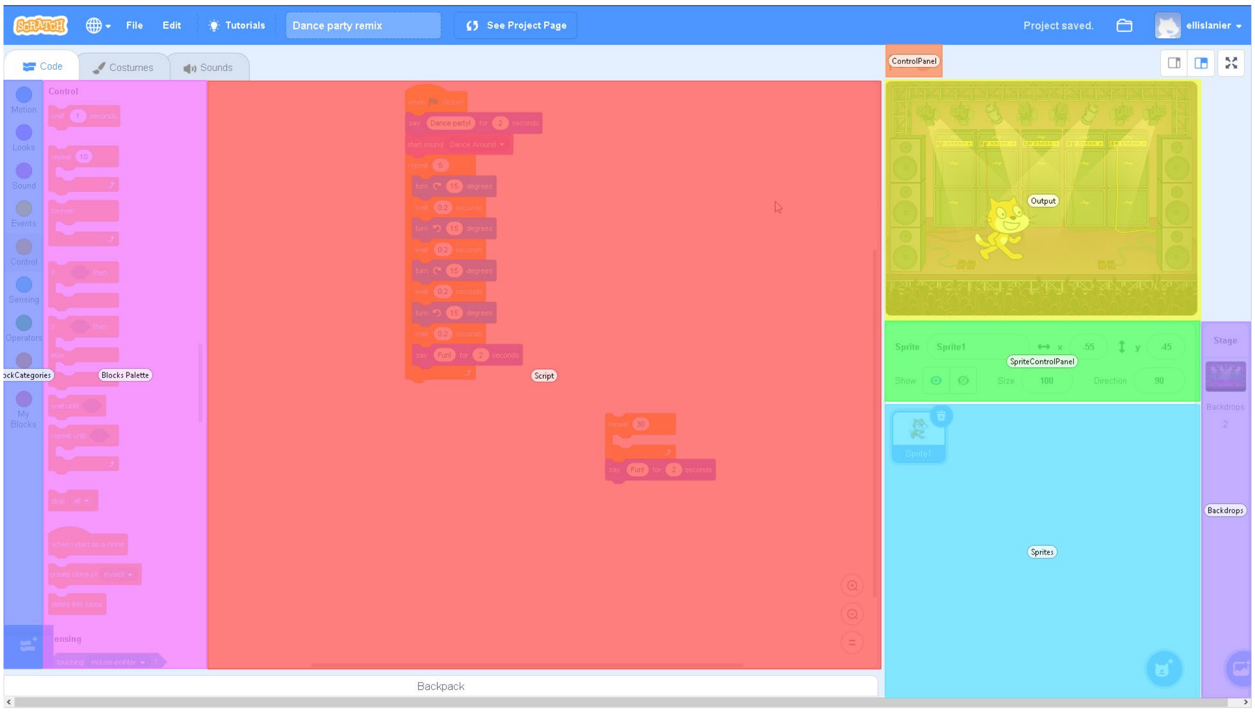


Figure 3. Eight AOIs of Scratch

Table I

Description of AOIs

AOIs	Description
Block categories	General categories of blocks, such as motion or sound
Blocks palette	Available blocks within a selected block category.
Script	Area where a program is created by dragging and connecting blocks.
Control panel	Buttons to control the program output. The green flag runs a program and the red octagon stops it.
Output	The visual output of a program is displayed in this area.
Sprite control panel	Controls to change sprite features, such as size and direction.
Sprites	Thumbnails of sprites (characters) that the program controls.
Backdrops	Thumbnails of backdrops used in the Scratch output

Quantitative data consisted of total fixation duration and total fixation count calculated for each AOI.

Fixation duration refers to how long participants looked at a specific area on the screen, while fixation count reflects the number of times participants looked at that same area. Percentage values were calculated given that participants completed debugging activities at different times. Also, transitions between AOIs were calculated to deepen the understanding of each participant's cognitive strategies during debugging. Two types of transitions were calculated: (1) between script and output and (2) between blocks palette and script. Specifically, transitions were computed by summing up the number of times the eye fixation is moved from the script to output and from output to script. Transitions between AOIs are often used as metrics in eye-tracking research (Sharafi, Soh, and Guéhéneuc, 2015), including studies that assess program comprehension or debugging (e.g., Bednarik, 2012; Lin et al., 2015; Papavlasopoulou et al., 2017).

Qualitative data included researchers' notes about trends and patterns in participants' eye movements, transcriptions of cued retrospective reporting, and heatmaps. Heatmaps provide visual evidence of participants' eye movements based on the distribution of eye fixations on the screen (Sharafi et al., 2015). The first and second authors used constant comparison methods from qualitative grounded theory research (Glaser and Strauss, 1967; Strauss and Corbin, 1998) to review eye movement videos and transcripts line by line. Open coding techniques were used as "a starting point to provide the researcher with analytic leads for further exploration" (Saldaña, 2013, p. 101). Specifically, salient patterns and interpretations from each data source were compared with patterns and interpretations from other data sources for triangulation of findings (Greene, 2007, 2008). For example, researchers noticed that (1) eye movements frequently alternated between code and output, (2) participant reported that she was trying to connect certain animated actions with specific blocks in the script area, and (3) the heatmap showed significant eye fixation on the specific block mentioned by the participant. Finally, all authors reviewed findings about participants and created qualitative themes (Braun and Clarke, 2006) to describe employed cognitive strategies while debugging block-based programs.

Results

Eye Movement Analysis

Mila. Mila completed the first debugging activity in 516.90 seconds. During this activity, Mila had more eye fixation on scripts (57.07%) and output (34.34%) followed by sprite control panel, blocks palette, and control panel in descending order (Table 2). Alongside that, Mila's fixation count relied predominantly on scripts (56.06%) and output (34.03%). Eye movement analysis revealed that Mila's eyes often alternated between scripts and output areas as an attempt to understand the script, especially at the beginning of the debugging activity. During cued retrospective reporting, Mila explained that she tried to focus on the output, but not the script, to inform hypothesis creation. As she said, "I just kept watching it to I guess just to see what he was doing. So I didn't really have an idea." In other words, Mila did not have a hypothesis about what caused the error, which was reflected in the patterns and number of eye transitions captured by the eye tracker. There were 129 transitions between the scripts and the output area but only 5 transitions between the block palette and the scripts area (Table 3). This suggests that Mila did not consider adding a new block and/or replacing an existing one in the program. Further, Mila often mentioned during cued retrospective reporting that she was "just trying things" as she moved blocks around and/or changed block parameters. She found a clue that the repeat (loop) block was the problematic one, and she tried two debugging strategies: (1) entering zero as parameter to prevent the block from running, and (2) deleting blocks within the repeat block, but not the repeat block itself. By trial and error, Mila was able to fix the error in the first debugging activity.

Mila mentioned that she found the second debugging activity easier, and she completed it in 416.306 seconds. Eye fixation duration was predominantly on the scripts (65.32%), and output (23.23%) and fixation counts were also mostly on scripts (61.60%) and output (24.58%) (Table 2). This time, however, Mila was more strategic in her debugging. When prompted to share if she had a hypothesis about how to fix the faulty program, Mila said that "There wasn't enough turns and moving, so that's why you had to add more.". Mila then searched for blocks in the palette to add to the code sequence, which explains the higher number of eye fixation on the blocks palette area (6.87%), and more transitions between code and palette area (n=34) compared to the first debugging activity (Table 3). When she encountered an unexpected error (e.g., sprite drawing a square rather than a rectangle), Mila knew that she had to change the move block to increase the number of steps for two sides of the shape. She was able to use the output (cat drawing a shape) and her mathematical knowledge about rectangles to inform her debugging strategies. In summary, she implemented three strategies while debugging: (1) deleting blocks, (2) changing the number of steps in the move block, and (3) deleting blocks within the repeat block.

Table II

Fixation Duration and Fixation Counts for Each Debugging Activity

Participant	Debugging activities	Backdrops	Block Categories	Blocks Palette	Control Panel	Output	Scripts	Sprite Control Panel	Sprites
Mila	Debugging Activity 1								
	Fixation Duration (%)	0.06	0.09	1.53	1.12	34.34	57.07	3.62	0.71
	Fixation Count (%)	0.07	0.07	2.61	0.07	34.03	56.06	3.35	1.21
	Debugging Activity 2								
	Fixation Duration (%)	0	0.15	4.77	3.24	23.23	65.32	1.90	0.20
	Fixation Count (%)	0	0.31	6.87	2.82	24.58	61.60	2.14	0.31
Emmy	Debugging Activity 1								
	Fixation Duration (%)	0.18	1.53	20.76	1.34	31.56	41.65	0.36	0.30
	Fixation Count (%)	0.38	2.36	25.97	1.45	23.08	42.50	0.61	0.61
	Debugging Activity 2								
	Fixation Duration (%)	0.04	1.28	8.80	1.13	31.08	53.72	1.42	0.90
	Fixation Count (%)	0.11	1.87	10.26	1.47	28.40	52.61	1.98	2.04

Table III

Transitions Between Block Categories and Script, and Script and Output

Participant	Block Palette-Script		Script-Output	
	Debugging Activity 1	Debugging Activity 2	Debugging Activity 1	Debugging Activity 2
Mila	5	34	129	100
Emmy	35	26	82	114

Emmy. Emmy fixed the error in the first debugging activity in 479.80 seconds. During this time, she fixated the most on three of the AOIs, scripts (41.65%), output (31.56%), and blocks palette (20.76%) (Table 2). Emmy seemed more strategic about debugging strategies. After examining the script from top to bottom, she reviewed the output to evaluate the outcome of the program. Then, her eye movements alternated between the script and output areas to find the error. After she figured out the error, she focused on the block palette to locate blocks to fix the error. During this process, her eye movements alternated between block palette and scripts 35 times. As Emmy was watching her recorded eye transitions between block palette and scripts, she explained that she was "trying to figure it out what [I] should add or take away." Further, she said that she created two hypotheses: (1) to make the cat stop dancing and (2) to stop the music. After she was not able to stop the cat, she tried the second hypothesis. While trying to test both hypotheses, Emmy added new blocks and made transitions between the blocks palette and scripts. The high number of fixation counts on scripts (42.50%) and blocks palette (25.97%) AOIs aligns well with her explanations. After Emmy updated the script, she assessed the output. After multiple trials, she went back to the top of the script to review it. During the first debugging activity, Emmy's transitions between script and output areas amounted to 82 times total (Table 3). Overall, she used three debugging strategies to fix the error in the first debugging activity; (1) changing the parameter of the repeat block, (2) adding a new block named stop at sounds, and (3) deleting the second repeat block along with all other blocks within that loop. Emmy successfully completed the activity after removing one of the loops from the code and using trial and error to tweak the number of loops that controlled the cat movement.

Emmy's eyes fixated on scripts (53.72%) and blocks palette (8.80%), and these parameters changed dramatically compared to her fixation durations in the first debugging activity. This time, Emmy focussed less on the blocks palette but more on the scripts. Her fixation durations on the output in both debugging activities were very similar (Table 2). Emmy found this activity harder as she explained that she could not remember how the pen down block works. To figure out its function, she disconnected all other blocks except for the pen down block and searched the block palette to find a clue. However, this strategy did not help her in identifying the block function. Compared to the first debugging activity, there were more transitions between the script and output (n=114) as Emmy tried to interpret the script and figure out the error. She removed blocks and changed parameters, and she was able to make the sprite draw a square. By using her mathematical knowledge, she created a hypothesis, but she did not know how to execute it in the script. As she explained, "I was trying to think how I could make the size shorter. So I put both 50 [number of steps in move block], but that didn't work the way I thought. And it just made the square bigger." Therefore, Emmy spent more time on the script (53.72%). Also, even though she did not add new blocks to the code, she had a high number of fixations on the blocks palette (10.26%). In her cued retrospective reporting, she explained "I didn't know what I was actually looking for." This confusion is also evident in her rapid eye movements from one area to another area on the screen. Emmy was not able to successfully complete the second debugging activity within the allocated time.

Cross-case Analysis: Qualitative Themes

1. Frequent and continuous eye fixation on output for hypothesis generation. Participants' eye fixation focused significantly on output, which served as an aid for hypothesis generation about the cause of the problem. Output was the second most fixated AOI for both participants in both debugging activities (Table 1). The pattern of their eye fixation on output is well represented in Figure 4, which is a heatmap of one participant's fixation. Participants ran the code multiple times (see fixation counts in Table 2) to craft a hypothesis/idea, and then they would review the program. When prompted to explain this, Mila said "I was trying to see why he [cat] was not stopping. I think at this point I didn't really know where to go, so I just kept watching it just to see what he was doing." This behavior is indicative of hypothesis generation through interpreting the output first and relating corresponding actions of the character to the code blocks to identify where the error(s) might possibly be located.

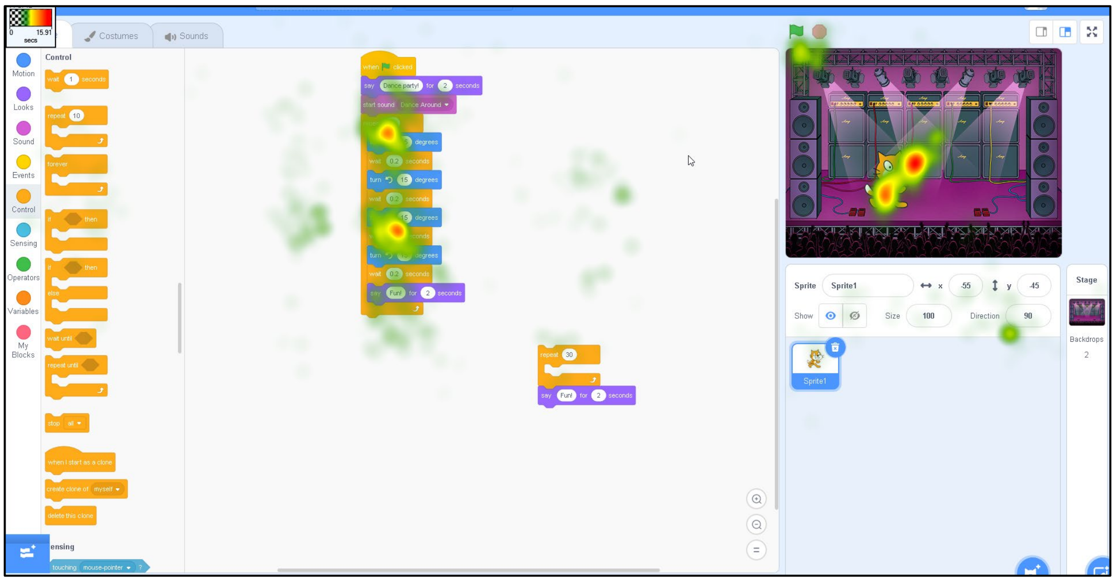


Figure 4. Heatmap as Example of Mila's Eye Fixations in the First Debugging Activity (based on fixation duration). Green Represents Short Fixation Duration While Red Represents Longer Fixation Duration

2. Simultaneous output visualization and top-down block review. Early in the debugging activity, participants played the program multiple times. Analysis of their eye movements revealed that they attempted to simultaneously observe the output and perform a top-down analysis of the block sequence. They used this strategy to understand the function of blocks even before debugging the code, and participants confirmed this during cued reporting. Indeed, the number of script-output transitions was much higher than the number of block palette-script transitions (Table 3). When showed her eye movements, Emmy recalled that she reviewed the code from "top to bottom", and that she was "looking at everything and then pressed play (...), trying to understand everything, looking at every step [block]."

This aligns with the literature on novice learners' program comprehension and debugging, as they often follow a linear, line-by-line approach.

3. Eye fixations and debugging targeted the bottom of a block sequence. Participants primarily focused on the bottom of a block sequence, which led to them adding and deleting blocks rather than reordering blocks or tinkering with other blocks at the top. When Emmy was asked about why her eye movements were focusing at the bottom of the code, she said "The most important stuff is usually up top." This suggests the participant may think of bugs as errors that occur towards the end/bottom of a code sequence. Because their debugging was often limited to adding/removing blocks at the bottom, they ended up with inefficient and long programs, even when they successfully accomplished the goal.

4. Eye fixation on faulty block only occurred if participant had a hypothesis. Participants were only strategic about reviewing a faulty code if they had a hypothesis for fixing the error. Their eye movements pointed straight to the hypothesized faulty block. For instance, Mila said that "The repeat part [block] was throwing me off because there were two repeats." When asked to elaborate on this, she added "I thought, well if it's repeating it, if it's just going and going, it must be an issue with like the repeats in the code." Although it was not possible to calculate fixation counts and time on specific blocks, the videos clearly showed Mila repeatedly fixating on the repeat block.

5. Random eye movements indicative of random trial and error. Participants often used trial-and-error techniques, and this was reflected in random eye movement transitions between AOIs, as well as along the block sequence. When asked about this, participants simply argued that they did not have a hypothesis. Interestingly, both participants acknowledged using trial and error. Mila added that another factor was helpful in fixing the faulty block: "a little bit of luck." Participants were unsure if they had a hypothesis, even when they actually had one. This could be indicative of a misconception in which they associate having a hypothesis with correctly understanding the cause of the problem and mastering a programming concept.

6. Recognizing a key block helps narrow down area to be debugged. Participants used cues in the output to identify the faulty block area. As they reviewed the output, they associated the fun speech bubble with the block "say fun for two seconds". As Mila said, "I thought that the problem was like after the fun." Eye movements showed participants alternating between output and the analogous block. This recognition led participants to narrowing down the part of the program to be debugged, given that they used debugging strategies such as adding/deleting blocks after that block.

7. Eye fixation relied more on block palette when they understood the program. When participants understood the program, they made more attempts to add new blocks to the script. Consequently, there was a higher number of eye fixations on blocks from the palette. Mila focused on the blocks palette AOI and added new blocks in the second debugging activity, which she confirmed was the easiest because she understood the block sequence. On the other hand, Emmy acknowledged that she did not understand the second debugging activity as much as the first activity, which led to fewer fixation counts on the blocks palette AOI. Emmy reported in the prior knowledge and cued retrospective interview that she did not know what the pen down block does. Analysis of her eye movements showed that she focused on the pen down block for a longer period of time as she attempted to understand it.

Discussion and Conclusion

This mixed methods study used eye movement data and cued retrospective reporting techniques to explore early childhood preservice teachers' cognitive strategies as they debugged block-based programs. Quantitative analysis of eye movements revealed that participants' eye fixation relied mostly on the output and scripts AOIs, and consequently, there were more script-output eye transitions. Participants primarily fixated on the output as an attempt to form hypotheses although participants often reported that they failed at it during retrospective reporting. In a study about cognitive strategies and visual attention during debugging, Bednarik (2012) found that novice programmers reviewed output first to form hypotheses while more experienced programmers used only the scripts or a combination of output and scripts. In a study with kids, Papavlasopoulou et al. (2017) found that those who spent more time on output and/or characters while coding in Scratch were outperformed in terms of learning gains by kids who spent more time reviewing scripts. It is important to support preservice teachers in purposeful and strategic consideration of both scripts and output for hypothesis generation. Follow-up research could feature scaffolding prompts (Ge and Land, 2004) that guide preservice teachers in associating certain events in the output with corresponding blocks/commands in the scripts. This could be helpful for program comprehension, identification of the faulty code area, and purposeful hypothesis generation.

Study participants had a significantly higher number of script-output transitions as they attempted to simultaneously review output and script. A study that compared programming performance in experts and novices found that experts switched between scripts and program more often than novices (Hejmady and Narayanan, 2012). Frequent eye movement between script and output is considered typical behavior of hypothesis testing as one modifies the program and refers to the output to evaluate it (Papavlasopoulou et al., 2017). However, Mila's and Emmy's debugging strategies were rarely informed by a hypothesis. They identified critical AOIs to focus on, but they could not systematically review output and block script. Further, they did not know how to structure their debugging process, as they would say "I didn't know where to go." Qualitative data analysis revealed that participants mostly resorted to trial and error techniques to identify and fix errors in a faulty code. Other debugging strategies include using a zero parameter in a block to prevent it from running, deleting blocks within a loop (repeat), and adding/deleting blocks at the bottom of a block sequence. Participants were not sure of the cause of the error even when they correctly completed the task.

Mixed methods analysis combined qualitative and quantitative findings into themes that provide in-depth accounts of participants' debugging. It was noticeable that participants (1) frequently and continuously fixated on output for hypothesis generation, (2) attempted to simultaneously visualize the output and review the program from top to bottom, (3) primarily fixated on and debugged the bottom of a program, (4) only fixated on faulty blocks if they had a hypothesis to inform such eye movement, (5) often engaged in random trial and error and random eye movements, (6) identified key blocks to help narrow down which area to be debugged, and (7) fixated more on the block palette when they properly understood the code sequence.

Creating a conceptually grounded hypothesis while programming is critical (Brooks, 1983) as it informs debugging actions. It was noticeable that participants in this study often engaged in random trial and error, which was also evident in random eye movements. This finding is extensively supported by the literature, which characterizes trial and error as a debugging strategy that is repeatedly used by novice learners (Fitzgerald et al., 2008; Jadud, 2005; Simon et al., 2008). Creating a hypothesis and strategy prior to debugging is typical of more experienced programmers (Gould and Drongowski, 1974). In this study, participants would only review the block palette AOI if they had a hypothesis and understood the scripts, such as Mila's idea to use a stop block to make the cat stop. She scrolled through block categories in search of that block.

Identifying where the bug is in a long code sequence is critical for effective and efficient debugging. Research has found that a line-by-line, top-down approach to understand and debug code is typical of novice learners (Alqadi and Maletic, 2017; Busjahn, Schulte, and Busjahn, 2011; Yusuf, Kagdi, and Maletic, 2007). Experts, on the other hand, are more strategic about processing the code, which occurs in a nonlinear manner compared to novices (Busjahn et al., 2011). Expert programmers often search and locate beacons (Crosby, Scholtz, and Wiedenbeck, 2002), which are defined in the computer science literature as a structure, statement, or operation that aid programmers in creating and testing a hypothesis (Lin et al., 2015). Both participants in this study successfully identified an event in the output (fun bubble) and an analogous block (say fun for two seconds) during the first debugging activity. They used the block as a beacon to realize that the faulty part of the code was located after that block. Studies on comprehension of text-based languages found that experienced programmers recognize beacons to create and verify hypotheses (e.g., Aschwanden and Crosby, 2006; Crosby et al., 2002). However, in most instances, participants in the present study tried to modify the bottom area of a code sequence as a strategy to debug the code. Perhaps it would be beneficial to provide not only visual cues on the screen to guide participants' attention to key information (Sun and Hsu, 2019), but also a guided framework for hypothesis construction, testing, evaluation, and revision. The latter of these two recommendations has been proposed by Kim et al. (2018), who identified preservice teachers' struggles with hypothesis-driven programming of educational robots.

Professional learning that prepares preservice teachers to address the learning needs of 21st century students is critical so preservice teachers can feel more confident in offering integrated STEM learning experiences to their students. In fact, it has been found that most K-12 students in U.S. are not offered instruction that features coding until they reach high school (Google and Gallup, 2015). Results of this study will inform and inspire teacher educators to design and develop professional learning on coding and debugging for teacher preparation programs so that preservice teachers can integrate developmentally appropriate coding instruction in early childhood grades and beyond.

Limitations and Future Research

There are several limitations to this study. First, this case study examined the cognitive strategies of only two early childhood preservice teachers, which limits the generalizability of findings to larger populations. Follow-up studies with larger populations are invited. Second, participants attended a 2-hour workshop about coding, which means they still had limited knowledge about Scratch and block-based coding. The duration and content of the workshop should be expanded in future research to provide preservice teachers with opportunities to develop a more sophisticated conceptual understanding of key computer science concepts (e.g., loops, variables) prior to data collection about debugging. Third, both study participants were novice programming learners. Follow-up studies could compare cognitive strategies used by novice and more experienced early childhood preservice teachers. Fourth, study participants worked on debugging activities individually. Literature on programming instruction states that pair programming is an effective strategy (Braught, Wahls, and Marlin Eby, 2011) given the potential for peer scaffolding. One possible avenue for research is to investigate pairs' co-construction and implementation of debugging strategies using interaction analysis (Jordan and Henderson, 1995), as well as to examine similarities and differences in pairs' eye movements (Pietinen, Bednarik, Glotova, Tenhunen, and Tukiainen, 2008). Finally, debugging activities were created by the researchers. However, early childhood preservice teachers will need to fix errors in programs created by children when they become in-service teachers. It is quite likely that children's programs may not produce an output and/or may include many blocks that were not supposed to be attached to each other. Therefore, to make the debugging experience more authentic, future research might consider using block-based programs developed by children.

Declarations

Acknowledgements:

Authors' contributions: L.V. and I.A.A. conceived the study idea. L.V. designed workshop activities and debugging activities. L.V., I.A.A., and F.A. refined workshop activities and debugging activities. L.V. and I.A.A. implemented workshop and debugging activities, collected and analyzed data, and interpreted findings. F.A. contributed to interpretation of findings. LV., I.A.A., and F.A. contributed to the final version of this manuscript.

Competing interests: The authors declare that they have no competing interests.

Funding: This study did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

Alqadi, B. S., & Maletic, J. I. (2017). An empirical study of debugging patterns among novice programmers. In S. Edwards, M. Caspersen, T. Barnes, & D. Garcia (Eds.), *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education—SIGCSE'17* (pp. 15–20). New York, NY: ACM Press. https://doi.org/10.1145/3017680.3017761

Aschwanden, C., & Crosby, M. (2006). *Code scanning patterns in program comprehension*. Paper presented at the 39th Hawaii International Conference on System Sciences. Retrieved from https://pdfs.semanticscholar.org/1153/854b79709f4d552959eebf9f7b2c0431f63a.pdf

Basu, S. (2016). *Fostering synergistic learning of computational thinking and middle school science in computer-based intelligent learning environments* (Unpublished doctoral dissertation). Vanderbilt University, Nashville, TN, USA.

Bau, D. (2015). Droplet, a block-based editor for text code. *Journal of Computing Sciences in Colleges*, *30*(6), 138–144. Retrieved from https://dl.acm.org/doi/10.5555/2753024.2753052

Bednarik, R. (2012). Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *International Journal of Human-Computer Studies*, *70*(2), 143-155. https://doi.org/10.1016/j.ijhcs.2011.09.003

Bednarik, R., Schulte, C., Budde, L., Heinemann, B., & Vrzakova, H. (2018). Eye-movement modeling examples in source code comprehension: A classroom study. In M. Joy, & P. Ihantola (Eds.), *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (pp. 1-8). New York, NY: ACM Press. https://doi.org/10.1145/3279720.3279722

Beller, M., Spruit, N., Spinellis, D., & Zaidman, A. (2018). On the dichotomy of debugging behavior among programmers. In P. Lago, & M. Young (Eds.), *Proceedings of the 40th International Conference on Software Engineering - ICSE '18* (pp. 1–12). New York, NY: ACM Press. https://doi.org/10.1145/3180155.3180175

Braught, G., Wahls, T., & Marlin Eby, L. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education, 11*(1), 1–21. https://doi.org/10.1145/1921607.1921609

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101. https://doi.org/10.1191/1478088706qp063oa

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. Paper presented at the 2012 Annual Meeting of the American Educational Research Association. Retrieved from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Brooks, R. E. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies, 18*(6), 543-554. https://doi.org/10.1016/S0020-7373(83)80031-5

Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education*, *4*(2), 121–135.

Busjahn, T., Schulte, C., & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. In A. Korhonen, & R. McCartney (Eds.), *Proceedings of the 11th ACM Koli Calling International Conference on Computing Education Research* (pp. 1-9). New York, NY: ACM Press. https://doi.org/10.1145/2094131.2094133

Code.org. (n.d.). *CS fundamentals unplugged*. Retrieved from https://code.org/curriculum/unplugged

Computer Science Teachers Association & International Society for Technology in Education (2011). *Computational thinking for all* (2nd ed.). Retrieved from https://www.iste.org/explore/articleDetail?articleid=152&category=Solutions&article=Computational-thinking-for-all

Cooper, S., Dann, W., & Pausch, R. (2000). Developing algorithmic thinking with Alice. In D. Colton (Ed.), *The Proceedings of the Information Systems Education Conference*, (pp. 506–539). Philadelphia, PA: Education Special Interest Group of the Association of Information Technology Professionals and the Foundation for Information Technology Education. Retrieved from https://pdfs.semanticscholar.org/15c0/016285c067f2f8dbbf2d7d59f9ffceb9237f.pdf

Crosby, M., Scholtz, J., & Wiedenbeck, S. (2002). The roles beacons play in comprehension for novice and expert programmers. In J. Kuljis, L. Baldwin, & R. Scoble (Eds.), *Proceedings of the 14th Annual Workshop on Psychology of Programming Interest Group* (pp. 58-73). London, UK: Brunel University.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: Finding, fixing, and flailing, a multi-institutional study of novice debuggers. *Computer Science Education, 18*(2), 93-116. https://doi.org/10.1080/08993400802114508

Ge, X., & Land, S. M. (2004). A conceptual framework for scaffolding III-structured problem-solving processes using question prompts and peer interactions. *Educational Technology Research and Development*, *52*(2), 5-22. https://doi.org/10.1007/bf02504836

Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Chicago, IL: Aldine. https://doi.org/10.1097/00006199-196807000-00014

Google (n.d.). *Computational thinking for educators [online course].* Retrieved from https://computationalthinkingcourse.withgoogle.com/course?use_last_location=true

Google, & Gallup. (2015). *Searching for computer science: Access and barriers in U.S. K-12 education*. Retrieved from http://g.co/cseduresearch

Gould, J., & Drongowski, P. (1974). An exploratory study of computer program debugging. *Human Factors, 16*(3), 258-277. https://doi.org/10.1177%2F001872087401600308

Greene, J. C. (2007). *Mixed methods in social inquiry*. San Francisco, CA: Jossey-Bass.

Greene, J. C. (2008). Is mixed methods social inquiry a distinctive methodology? *Journal of Mixed Methods Research*, *2*(1), 7-22. https://doi.org/10.1177/1558689807309969

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Hejmady, P., & Narayanan, N. H. (2012). Visual attention patterns during program debugging with an IDE. In S. N. Spencer (Ed.), *Proceedings of the Symposium on Eye Tracking Research and Applications (ETRA'12)* (pp. 197-200). New York, NY: ACM Press. https://doi.org/10.1145/2168556.2168592

Isong, B. (2014). A methodology for teaching computer programming: First year students' perspective. *International Journal of Modern Education and Computer Science*, *6*(9), 15–21. https://doi.org/10.5815/ijmecs.2014.09.03

Jadud, M. C. (2005). A first look at novice compilation behaviour using BlueJ. *Computer Science Education*, *15*(1), 25-40. https://doi.org/10.1080/08993400500056530

Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *The Journal of the Learning Sciences*, *4*(1), 39-103. https://doi.org/10.1207/s15327809jls0401_2

K-12 Computer Science Framework Steering Committee (2016). *K–12 computer science framework*. Retrieved from http://www.k12cs.org

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, *52*(1), 200–210. https://doi.org/10.1016/j.chb.2015.05.047

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, *9*(1), 522–531. https://doi.org/10.1016/j.procs.2012.04.056

Kim, C., Yuan, J., Oh, J., Shin, M., & Hill, R. B. (2016). *Productive struggle during inquiry learning*. Paper presented at the European Association for Research on Learning & Instruction SIG 20 & SIG 26 Meetings, Ghent, Belgium.

Kim, C., Yuan, J., Vasconcelos, L., Shin, M., & Hill, R. B. (2018). Debugging during block-based programming. *Instructional Science*, *46*(5), 767–787. https://doi.org/10.1007/s11251-018-9453-5

Leech, N. L., & Onwuegbuzie, A. J. (2009). A typology of mixed methods research designs. *Quality & Quantity*, *43*(2), 265–275. https://doi.org/10.1007/s11135-007-9105-3

Leedy, P. D., & Ormrod, J. E. (2013). *Practical research: Planning and design*. Boston: Pearson.

Leonard, J., Buss, A., Gamboa, R., Mitchell, M., Fashola, O. S., Hubert, T., & Almughyirah, S. (2016). Using robotics and game design to enhance children's self-efficacy, STEM attitudes, and computational thinking skills. *Journal of Science Education and Technology*, *25*(6), 860–876. https://doi.org/10.1007/s10956-016-9628-2

Lin, Y. T., Wu, C. C., Hou, T. Y., Lin, Y. C., Yang, F. Y., & Chang, C. H. (2015). Tracking students' cognitive processes during program debugging — An eye-movement approach. *IEEE Transactions on Education*, *59*(3), 175-186. https://doi.org/10.1109/TE.2015.2487341

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*(1), 51–61. https://doi.org/10.1016/j.chb.2014.09.012

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67–92. https://doi.org/10.1080/08993400802114581

Moreno, R., & Mayer, R. E. (1999). Cognitive principles of multimedia learning: The role of modality and contiguity. *Journal of Educational Psychology*, *91*(2), 358– 368. https://doi.org/10.1037/0022-0663.91.2.358

National Research Council. (2011). *Successful K-12 STEM education: Identifying effective approaches in science, technology, engineering, and mathematics.* Washington, DC: National Academies Press.

Obaidellah, U., Al Haek, M., & Cheng, P. C. H. (2018). A survey on the usage of eye-tracking in computer programming. *ACM Computing Surveys*, *51*(1), 1–58. https://doi.org/10.1145/3145904

Obama, B. (2016). *Computer science for all*. Retrieved from https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all

Papavlasopoulou, S., Sharma, K., Giannakos, M., & Jaccheri, L. (2017). Using eye-tracking to unveil differences between kids and teens in coding activities. In P. Blikstein, & D. Abrahamson (Eds.), *Proceedings of the 2017 Conference on Interaction Design and Children* (pp. 171-181). New York, NY: ACM Press. https://doi.org/10.1145/3078072.3079740

Paul, A. M. (2016). The coding revolution. *Scientific American*, *315*(2), 42–49. https://doi.org/10.1038/scientificamerican0816-42

Pietinen, S., Bednarik, R., Glotova, T., Tenhunen, V., & Tukiainen, M. (2008). A method to study attention aspects of collaboration: Eye-tracking pair programmers simultaneously. In K. Räihä, & A. T. Duchowski (Eds.), *Proceedings of the 2008 Symposium on Eye Tracking Research and Applications* (pp. 39-42). Savannah, GA: ACM Press. https://doi.org/10.1145/1344471.1344480

Proctor, C. (2019). Measuring the computational in computational participation: Debugging interactive stories in middle school computer science. In K. Lund, G. Niccolai, E. Lavoué, C. Hmelo-Silver, G. Gweon, & M. Baker (Eds.), *A Wide Lens: Combining Embodied, Enactive, Extended, and Embedded Learning in Collaborative Settings* (Vol. 1, pp. 104–111). Lyon, France: International Society of the Learning Sciences.

Project Growing Up Thinking Scientifically [GUTS] (n.d.). Retrieved from http://www.projectguts.org/resources

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Saldaña, J. (2013). *The coding manual for qualitative researchers*. Los Angeles, CA: SAGE.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351–380. https://doi.org/10.1007/s10639-012-9240-x

Sharafi, Z., Soh, Z., & Guéhéneuc, Y. (2015). A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, *67*(1), 79–107. http://dx.doi.org/10.1016/j.infsof.2015.06.008

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*(1), 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Simon, B., Bouvier, D., Chen, T., Lewandowski, G., McCartney, R., & Sanders, K. (2008). Common sense computing (episode 4): Debugging. *Computer Science Education*, *18*(2), 117-133. https://doi.org/10.1080/08993400802114698

Strauss, A., & Corbin, J. M. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Thousand Oaks, CA: Sage.

Sun, J. C., & Hsu, K. Y. (2019). A smart eye-tracking feedback scaffolding approach to improving students' learning self-efficacy and performance in a C programming course. *Computers in Human Behavior*, *95*(1), 66–72. https://doi.org/10.1016/j.chb.2019.01.036

van Gog, T., Paas, F., van Merriënboer, J. J., & Witte, P. (2005). Uncovering the problem-solving process: Cued retrospective reporting versus concurrent and retrospective reporting. *Journal of Experimental Psychology: Applied*, *11*(4), 237-244. https://doi.org/10.1037/1076-898X.11.4.237

Vasconcelos, L., & Kim, C. (2019). Coding in scientific modeling lessons (CS-ModeL). *Educational Technology Research and Development*. https://doi.org/10.1007/s11423-019-09724-w

Vihavainen, A., Airaksinen, J., & Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the Tenth Annual Conference on International Computing Education Research - ICER '14*, (pp. 19–26). New York, NY: ACM Press. https://doi.org/10.1145/2632320.2632349

Yen, C., Wu, P., & Lin, C. (2012). Analysis of experts' and novices' thinking process in program debugging. In K. C. Li, F. L. Wang, K. S. Yuen, S. K. S. Cheung, & R. Kwan (Eds.), *Engaging Learners Through Emerging Technologies* (Vol. 302, pp. 122–134). https://doi.org/10.1007/978-3-642-31398-1_12

Yukselturk, E., & Altiok, S. (2017). An investigation of the effects of programming with Scratch on the preservice IT teachers' self-efficacy perceptions and attitudes towards computer programming. *British Journal of Educational Technology*, *48*(3), 789–801. https://doi.org/10.1111/bjet.12453

Yusuf, S., Kagdi, H. H., & Maletic, J. I. (2007). Assessing the comprehension of UML class diagrams via eye tracking. In *Proceedings of 15th IEEE International Conference on Program Comprehension* (pp. 113–122). Alberta, Canada: IEEE. https://doi.org/10.1109/ICPC.2007.10