Kim *et al. Int J Educ Technol High Educ* (2022) 19:26
https://doi.org/10.1186/s41239-022-00319-9

International Journal of Educational Technology in Higher Education

RESEARCH ARTICLE Open Access

Debugging behaviors of early childhood teacher candidates with or without scaffolding

ChanMin Kim[1*], Lucas Vasconcelos[2], Brian R. Belland[3], Duygu Umutlu[4] and Cory Gleasman[5]

*Correspondence:
cmk604@psu.edu
[1] Learning, Design, and Technology, Educational Psychology, College of Education, The Pennsylvania State University, 314D Keller Building, University Park, PA 16802, USA
Full list of author information is available at the end of the article

Abstract

It is critical to teach all learners to program and think through programming. But to do so requires that early childhood teacher candidates learn to teach computer science. This in turn requires novel pedagogy that can both help such teachers learn the needed skills, but also provide a model for their future teaching. In this study, we examined how early childhood teacher candidates learned to program and debug block-based code with and without scaffolding. We aimed to see how approaches to debugging vary between early childhood teacher candidates who were provided debugging scaffolds during block-based programming and those who were not. This qualitative case study focused on 13 undergraduates majoring in early childhood education. Data sources included video recording during debugging, semi-structured interviews, and (in the case of those who used scaffolding) scaffold responses. Research team members coded data independently and then came to consensus. With hypothesis-driven scaffolds, participants persisted longer. Use of scaffolds enabled the instructor to allow struggle without immediate help for participants. Collaborative reasoning was observed among the scaffolded participants whereas the participants without scaffolds often debugged alone. Regardless of scaffolds, participants often engaged in embodied debugging and also used trial and error. This study provides evidence that one can find success debugging even when engaging in trial and error. This implies that attempting to prevent trial and error may be counterproductive in some contexts. Rather, computer science educators may be advised to promote productive struggle.

Keywords: Teacher learning, Computing education, Debugging, Robot programming

Introduction

Within early childhood curricula, there is often a focus on encouraging children to engage in structured and unstructured play with a variety of toys, tools, and other manipulatives (Ashiabi, 2007; Brooker et al., 2014; Smolucha & Smolucha, 2021). In so doing, children can learn cause-and-effect, methods to interact with and negotiate play with other children and adults, and more sophisticated language. This in turn has an outsize contribution to children's development. Robotics has been used within children's play (e.g., Breazeal et al., 2016; Çetin & Demircan, 2020; Kazakoff & Bers, 2014; Sullivan et al., 2017) as it affords the ability of a teacher to structure children's play while also allowing free exploration on the part of children. To facilitate learning with robotics among young children, it is critical to prepare early childhood teachers to work with

and program robots (Bers et al., 2013; Bers, 2018a). Within early childhood education, robots are controlled using block-based code, a form of programming that uses blocks representing actions of logic that can be assembled to instruct a robot to perform a sequence of movements. While block-based programming is assuredly less intimidating and easier to implement than such text-based programming platforms as C# and Python among non-computer science majors, it does require sound programming logic and can suffer from bugs (Lye & Koh, 2014). As such, it is critical for early childhood teacher candidates to learn to debug, defined as the ill-structured problem-solving process in which programmers determine the cause of and resolve a programming error (Kim et al., 2018). Debugging activities are often included purposefully in block-based programming learning contexts (Kim et al., 2021; Lytle et al., 2019; Neutens & Wyffels, 2020; Socratous & Ioannou, 2021).

Ill-structured problem solving is often best learned when learners receive scaffolding that structures and problematizes the problem solving process as they engage in the problem at hand (Reiser, 2004; Wood et al., 1976). Indeed, meta-analyses have indicated that scaffolding has some of the strongest between-subjects and within-subjects effects when used to support the learning of computer science concepts and processes (Belland et al., 2017a, b). But just as it is critical to examine between-subjects and within-subjects effects resulting from the use of scaffolding, it is also critical to examine how learners engage in debugging while supported and not supported by scaffolding. Given this gap, this study investigated how early childhood education teacher candidates approached debugging when provided with debugging scaffolds during block-based programming.

Conceptual framework

The conceptual framework of the present study was constructed based on the literatures on teacher learning of programming (Kim et al., 2015, 2018; Bers et al., 2013; Sullivan & Moriarty, 2009), scaffolding (Belland et al., 2013; Reiser, 2004; van de Pol et al., 2011), and hypothesis-driven debugging (Kim et al., 2018; Katz & Anderson, 1987; McCauley et al., 2008; Vessey, 1985; Yoon & Garcia, 1998). Each literature base and the conceptual framework are discussed in the following sections that also led to the research question of the study.

Teacher learning of programming for computer science for all

It is critical to include computer science in early childhood education (ECE) (Bers, 2019). This does not mean that all ECE teachers have to be a capable programmer. Rather, they need to understand computer science concepts and practice so these can be integrated into the existing curricula. There are numerous programming platforms that are not as overwhelming as text-based programming platforms (Bers, 2018b; Brennan & Resnick, 2013; Lye & Koh, 2014; Näykki et al., 2021; Trilles & Granell, 2020; Williams et al., 2019). These platforms are commonly termed block-based programming because they use blocks (i.e., icons with words and/or pictures that represent instructions for computers). Furthermore, block-based programming is commonly paired with robots or animations that perform the actions represented in the programmed code. For example, Scratch Jr. is a block-based programming platform that can be used by children who are at least 5 years old. Children code in Scratch, Jr. to

Fig. 1 Example block code at the prereader level making the robot change its movement when its light color changes

instruct an animated sprite to perform desired actions. Another example of block-based programming is Ozoblockly, which can be used to instruct Ozobots (i.e., small robots) to perform a set of actions. Ozoblockly also includes pre-reader blocks (level 1) that make learning of programming unintimidating (see Fig. 1 for an example). Despite many available tools and other resources, ECE teacher learning of computer science concepts and programming is still limited (c.f., Kim et al., 2018, 2021; Bers et al., 2013; Papadakis & Kalogiannakis, 2019). Where ECE teacher learning of programming was pursued, hands-on activities were used for them to experience authentic, block-based programming for teaching children (Kim et al., 2015, 2018; Bers et al., 2013). Robot programming was often used in ECE teacher learning contexts due to its unique affordances from having physical, tangible objects that can facilitate children's dramatic play (Kim et al., 2020).

Scaffolding

Simply inviting learners to address complex problems is not sufficient. Rather, one needs to provide scaffolding, which supports learners from cognitive and motivational perspectives as they address complex problems (Belland et al., 2013; Näykki et al., 2021; Wood et al., 1976). Scaffolding can accomplish this by simplifying task elements that are not central to learning goals, while drawing attention to task elements that are (Reiser, 2004). Meta-analyses indicate that scaffolding leads to stronger cognitive learning outcomes related to STEM than lecture (Belland et al., 2017a, b), and that pre-post gains

were strongest at the college level (Belland et al., 2017a). Scaffolding can take the form of question prompts, expert modelling, indicating important things to consider, and proving feedback (Belland et al., 2013; van de Pol et al., 2011; Wood et al., 1976). Within a computer science education context, scaffolding has been used to help students regulate their learning (Su, 2020), focus on key project requirements (Demetriadis et al., 2008) and to provide feedback in the form of hints to solve the problem (Holland, 2009). Scaffolding has a clear role in helping teacher candidates learn to program, but it also can play a role in teaching young children to code; the first application of the scaffolding metaphor in the context of education was in the context of play in early childhood education (Wood et al., 1976). In Wood et al (1976), scaffolding was proposed as the process by which adults temporarily support young children as they attempted to build a pyramid with wooden blocks. In teaching children to code, robots or animated sprites are in essence the wooden blocks. Rather than building a pyramid by manipulating wooden blocks with their hands, children need to manipulate the sprites' or robots' actions using code. Thus, scaffolding can help teacher candidates learn to debug but can also provide a model of productive interactions with children in the future.

Debugging

As mentioned earlier, block-based programming platforms are more inviting than text-based programming languages to novice programming learners. But block-based programming still involves debugging. Debugging is often tiresome and frustrating work, but is a natural part of programming (McCauley et al., 2008; Spinellis, 2018). While many computer scientists hold that using a hypothesis-driven approach to debugging is best, in reality most professional programmers and computer science instructors use unstructured methods (Michaeli & Romeike, 2019; Spinellis, 2018). This is in part because debugging is not often a central focus within computer science courses (Spinellis, 2018). Unstructured approaches to debugging are often termed tinkering (Quan & Gupta, 2020). Cautious tinkering can be defined as writing and iterating code to solve the bug while keeping track of the structure and function of the program (Perkins et al., 1986). Meanwhile, in haphazard tinkering, changes are often made but not tested, and/or made in a randomly picked spot in the program to see what happens (Perkins et al., 1986).

Debugging can serve to formatively assess and scaffold actual learning of novice programming learners (Kim et al., 2018). Especially during pair debugging, the dialogue in the pair reveals what they currently know and do not know. Besides, debugging is critical to both computer science and computational thinking education. There are some efforts to improve debugging by facilitating structured approaches through use of technological tool development such as Whyline (Ko & Myers, 2008), Gidget (Lee et al., 2014), Ladebug (Luxton-Reilly et al., 2018), and Visual Studio Code (Del Sole, 2019), and also scaffolding design (Ardimento et al., 2019; Ko et al., 2019). However, there is little research on how to scaffold teacher learning of debugging in higher education contexts. Considering that haphazard debugging is often observed among ECE teacher candidates with incomplete understanding of what caused bugs and what resolved them (Kim et al., 2015, 2018), scaffolding for structured debugging processes was expected to be a logical next step.

Study framework

Grounded in the aforementioned literature, the conceptual framework that guided this study had three main foci. First, the framework situates ECE teacher learning of programming through hands-on robot programming and authentic design for teaching children (Kim et al., 2015, 2018; Bers et al., 2013). Thus, the present study invited ECE teacher candidates to choreograph robots, and program and use them in their field experience teaching preschoolers (see details in the methods section). Second, the framework positions scaffolding as a tool to simplify task elements that are not central at the moment and draw attention to those that are central (Reiser, 2004). This was done in the present study by creating phases from code reading to hypothesis generation, testing, and reflection (Fig. 2). Modeling was also used (Belland et al., 2013; Ko et al., 2019; van de Pol et al., 2011) by giving example responses to scaffolding prompts. And justification was included to promote expectancy for success (Belland et al., 2013) by explaining that what they are being asked to do would lead to more successful debugging and why. Last, the framework positions hypothesis-driven debugging (Kim et al., 2018; Fitzgerald et al., 2008; Vessey, 1985) as an overall approach. Thus, a strategic scaffolding approach (Belland 2017; Hannafin et al., 1999) was employed in the present study centered around hypothesis generation and testing (Fitzgerald et al., 2008; McCauley et al., 2008; Vessey, 1986) (Fig. 2). It also incorporated scaffolding features recommended for reflective debugging specifically of block-based code (Kim et al., 2018). The strategies of reading before writing (deconstructing before debugging) (Griffin, 2016) was highlighted, and why questions (Ko & Myers, 2008) were embedded.

Research question

How do approaches to debugging vary between early childhood teacher candidates who were provided debugging scaffolds during block-based programming and those who were not?

Method

Setting and participants

Participants were recruited from two sections of a course on integrated arts in early childhood education at a large university in the southern region of the USA. The study was approved by the university IRB, and participants were read a recruitment script and invited to consider and sign an informed consent form. Course section A was provided

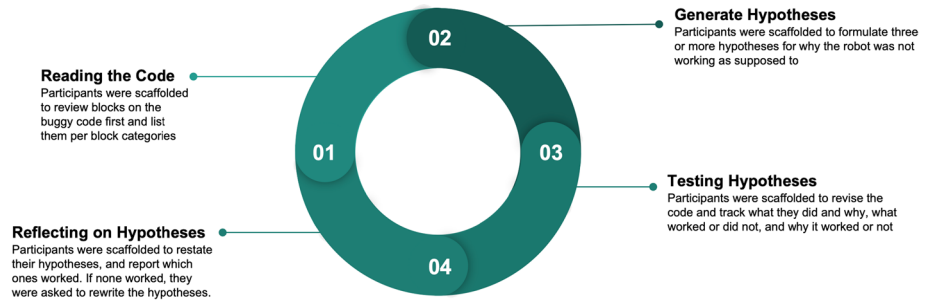


Fig. 2 Scaffolding for hypothesis-driven debugging in the conceptual framework of the present study

with written debugging scaffolds for hypothesis generation and testing as participants debugged code errors, while section B only had verbal instructor support as available. In total, 42 ECE teacher candidates participated in the study. In this qualitative case study, we focused on 13 participants who were video-recorded. Of these, 11 were females and two were males. One participant was Asian, and the others were White. Their mean age was 20.46 (SD = 1.01) years. Except for two participants (Jean and Meg) with prior robot programming experience, all reported no to little programming knowledge. During debugging activities, participants worked in small groups as shown in Table 1. Pseudonyms are used for all participants.

Robot programming unit

The robot programming unit aimed to facilitate participants' use of robotics technologies as a medium for communication, inquiry, and engagement among preschoolers. The unit consisted of three modules (see Table 2), each with a duration of three hours (i.e., one class session). Each module had corresponding learning objectives. Ozobot Bit was used for all programming and lesson design activities. Ozobots are programmed using a block-based programming platform, Ozoblockly, consisting of five levels that include increasingly complex functions (e.g., loops and variables) at the higher levels. Instruction and practice programming activities initially focused on level 3 (intermediate) and progressed to level 4 (advanced) during the second and third modules. Various blocks were taught to cover sequential, repetition, and selection control structures.

As described in the conceptual framework earlier, the unit was designed to engage participants in authentic programming learning that involved actual teaching in preschools. Thus, the unit invited participants to design and implement a lesson in pairs for preschoolers to engage in dramatic play with robots. Participants were given a sample lesson in which robots were used in children's learning of shapes through dramatic play in preschool classrooms. Participants learned to code choreographed movements

Table 1 Team and participants

Course section (scaffold)	Team	Pseudonyms (gender)	Age	Race	Recording	Interview	Computer programming knowledge	Prior robot programming experience
Section A (presence)	1	Gwen (F)	20	White	Yes	Yes	None	No
		Cole (M)	22	White	Yes	No	Low	No
	2	June (F)	20	White	Yes	No	None	No
		Kiara (F)	20	White	Yes	No	None	No
		Grace (F)	20	Asian	Yes	No	None	No
	3	Diane (F)	21	White	Yes	Yes	None	No
		Sue (F)	20	White	Yes	Yes	None	No
Section B (absence)	4	Jean (F)	20	White	Yes	No	Low to none	Yes
		Meg (F)	23	White	Yes	No	Low to intermediate	Yes
	5	Irina (F)	20	White	Yes	No	None	No
		Todd (M)	20	White	Yes	No	None	No
	6	Pearl (F)	19	White	Yes	Yes	None	No
		Gail (F)	21	White	Yes	Yes	Low	No

Table 2 A summary of the robot programming unit

Module	Summary of activities
1	Introduction to STEAM education using robots
	Review of a sample lesson engaging preschoolers in learning shapes through dramatic play with Ozobots
	Introduction to Ozobot Bit and OzoBlockly level 3
	Programming shapes using Ozoblockly level 3
	Practice teaching with the sample lesson
2	Lesson implementation reflection
	Programming shapes using Ozoblockly level 4
	Debugging tasks (with scaffolds in course section A)
	Lesson design in teams
	Practice teaching with the team lesson
3	Lesson implementation reflection
	Debugging tasks (with scaffolds in course section A)

(Fig. 3) that the robots needed to make in the sample lesson. They also learned to debug errors impeding the successful execution of code through three tasks (see Table 3). They practiced the lesson and implemented it in their field experience preschool classrooms. They then designed a team lesson about a topic of their choice. Next, they worked in teams to complete any unfinished debugging tasks. Participants in the course section A were provided with scaffolds designed based on the conceptual framework of the present study (see the conceptual framework section and Fig. 2). All debugging was done collaboratively, but every participant in the scaffolding condition was invited to respond to the scaffolding prompts.

Data collection

Data sources included video recordings of debugging work, responses to the debugging scaffolds, and semi-structured interviews. In the first week, the study was introduced, and participants were invited to consider and sign the informed consent. In the second and third weeks, participants' actions and dialogues along with their computer screens were video-recorded while debugging given programming tasks. For participants from course section A, responses to scaffolding prompts during the debugging activities were also collected. Sample scaffolding prompts included "Restate hypothesis 1", "Changes made to test hypothesis 1", "Why did you choose to make these changes specifically?", and "What happened after these changes were made?" Following the completion of the unit, participants were interviewed for 20–30 min using semi-structured questions.

Data analysis

Qualitative data analysis techniques were used to analyze video and audio transcripts from both course sections (see Table 1), and responses to the debugging scaffold used in course section A. First, data reduction involved coding using a theoretically-driven coding scheme developed based on the conceptual framework (described earlier) and refined through multiple rounds of coding. Example nodes and sample data are listed in Table 4. We coded data in NVivo 12. A subsample of data sources from each course section was assigned so that at least two authors independently coded the same file

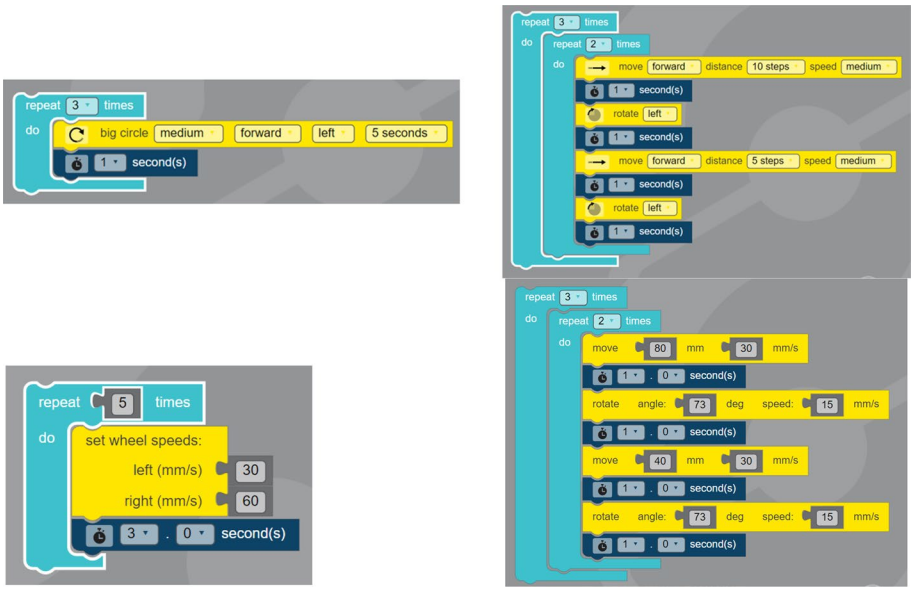
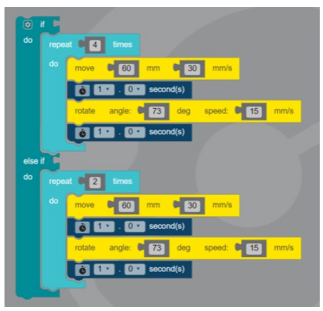
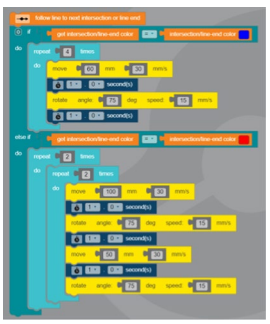


Fig. 3 Block code used in the sample lesson making the robot move in a circle (left) and a rectangle (right) using level 3 (top) and level 4 (lower)

Table 3 Debugging tasks

Debugging tasks	Buggy code	Debugged code example
Task 1: The code should make the robot trace the number, 4, but the code is problematic.		
Task 2: The code should make the robot trace the shape of a lollipop as the instructor demonstrates, but the code is problematic.		
Task 3: Mr. Johnson wants to use robots to teach students about colors and shapes. He draws two lines: a black line with a red end and a black line with a blue end. He wants the robot to follow the line and trace a square if it senses blue or trace two rectangles if it senses red. But the code does not work right.		

(Saldaña, 2016; Tracy, 2020). After multiple rounds of coding, three authors independently aggregated preliminary findings to generate salient observations supported by evidence from coded excerpts (Morse, 1994). Next, two other authors reviewed NVivo files and salient observations and discussed coding and coded data with each of the three authors individually (Saldaña, 2016). Then, the three authors went through revisions in their coding again independently, and one of them finalized all the coding. Finally, qualitative themes were crafted jointly (Boyatzis, 1998; Vaismoradi et al., 2016) to subsume salient observations that emerged from the data and to describe participants' debugging experiences with or without scaffolds.

Findings and discussion

Theme 1. With scaffolds, participants persisted longer with their efforts

Scaffolds that asked for hypotheses framed the debugging process as part of a hypothesis testing process. Participants were invited to enter three hypotheses. If more were needed, more could be added. Thus, when one hypothesis failed to help them with debugging, participants persisted to test another hypothesis. The persistence may have been enabled by controllability they perceived (Kim & Pekrun, 2014; Weiner, 1985). That is, the task of debugging may have been viewed as controllable through hypotheses. Justification provided in the scaffolding, as described earlier in the conceptual framework, may have promoted their persistence. Knowing that what they were asked to do (e.g., hypothesis writing and testing) would lead to successful debugging may have sustained their engagement. Expectancy for success is a critical factor for engagement (Bandura, 1997; Wigfield & Eccles, 2000). Diane and Sue (scaffolded group) demonstrated persistent testing of their changes even through multiple failures:

(In this episode, Diane and Sue worked on debugging task 3. They struggled with programming the robot to recognize line colors as part of tracing either a square or a rectangle.)

Diane: 03:07 So that didn't work. So now what do I do?
Sue: 03:10 We're really doing the one today, last time we did how many [tasks] three or two?
Diane: 03:45 It's funny. It's not even fricking letting me add this (block) now.

Table 4 Sample coding scheme nodes and data

Sample nodes	Sample excerpts
Rationale for a comment (suggestion, decision), or an action during debugging	"No, look if it's going that way, that means this wheel is faster. So I slow it down and I've been slowing it down and it's been going the same. I'll slow it down to like 10 and see what happens." (Pearl & Gale, Class 2 video)
Successful trial-and-error	"Bam, you're out of control. … Yay, it's working! Like in the world's smallest circle… I mean a square." (Irina & Meg, Class 2 video)
Unsuccessful trial-and-error	"Yeah. Mine literally does nothing… No matter whatever. Nothing..." (Diane & Sue, Class 3 video)
Did better than expected	"Yeah, I definitely think I did better than any would have expected... I wouldn't have expected that I would be able to program something and like identify a problem and fix it type thing." (Gwen, Interview)

Sue:03:49 You can't put it in. You have to do (inaudible) did the equal sign and that's where you put it in.
Diane: 04:00 Oh my gosh. Where do I put those?
Sue: 04:08 Put them up here and then put those that way and then do another one navigation line and then do get intersection line and color.
Diane: 04:46 How did you fix it? Like does anybody have problems with that? Do you have a recommendation? **Because it's really bad right now.**
(…)
Diane: 06:31 I think it's blinding.
Diane: 06:48 We're just talking just right. Alrighty. Here we go. Okay, great. It's moving in circles here.
Diane: 07:08 You've got to take it out there. (Diane told Sue to go test her robot on the line the instructor set up for testing)
(…)
Diane:07:42 Yeah. **Mine literally does nothing. So that happened to mine too, I just… No matter whatever.**
Diane:07:52 **Nothing changes for me. Nothing. My bot doesn't work.**
(…)
Instructor: 09:27 (To the whole class) So we've got some people figuring it out.
Instructor: 09:32 (To Diane) How's it going? Ok?
Diane: 09:38: Good. Everything's
Instructor: 09:39 Good? Everything's alright? Everything's great? Alright.
Instructor: 09:41 (To Sue) Did we figure it out back there?
Sue:09:43 No.
Instructor: 09:44 A little bit?
Sue: 09:45 It just moves in circles.
Instructor: 09:47 You're getting there.
Diane: 09:47 **So I did that, but shouldn't it move in a line when I wrote it? So it started on the line, alright, let me give it one more try *(Diane showed persistence).***
Diane: 11:27 Yeah, that happened.
Instructor: 11:35 So right now you guys are just telling it to pick up red, right?
Diane: 13:52 What color is the other one?
Sue: 14:00 I guess I could just use the graph colors….

As shown in the bolded discourse above, Diane and Sue went through unsuccessful cycles of trial and error. For example, Diane even reported that none of their changes in the code fixed the problem. She nonetheless told the instructor that everything was alright and suggested that her partner, Sue, try another change. Diane and Sue's process of trial-and-error could have disengaged them from debugging, but it did not. It seems that multiple changes (and thereby multiple failures on the way) were so natural for them to accept as part of hypothesis testing. During the interview, Diane was asked to compare debugging to other real-world problem solving, such as troubleshooting appliances. Diane hinted that scaffolded hypothesis testing was helpful, and it reminded her of multiple rounds of hypothesis testing in scientific experiments and guesses and checks in math problem solving, which seems to have contributed to her team's persistence.

Diane:17:34 I liked the hypothesis thing. Like why, what do you think needs to happen and then why did you, why do you think that? And then, what changes you would make? Yeah. Changes made.
Diane: 19:33 It kind of, it reminded me of something in science or math, like a guess and check almost like, okay, in science you have to come up with [a] hypothesis and then you have to test your hypothesis and then if it's wrong you gotta come back and redo it. So more so science than math and then math because it's like guessing and checking on this.

Without centering their debugging process around hypotheses, the participants without scaffolds may have perceived the debugging situation to be uncontrollable. While they were asked to reflect on challenges that they had experienced during the process of debugging, the participants without scaffolds were not asked to articulate hypotheses for causes of bugs and consequences of their changes in the code. Even if they were cognizant of what they were testing, they were not asked to create a formalized hypothesis after making changes. The process of testing changes may have felt like one long process compared to the scaffolded group who did testing hypothesis by hypothesis. The following discourses hint that the participants without the scaffolds desired to avoid their continuing process of debugging and move onto the next debugging task:

(In this episode, Gail and Pearl worked on debugging task 2. They worked on line navigation.)

Gail: 11:10 I'm getting some colored paper (Left her seat).
Pearl: 11:19 How did you do colors?
(…)
Pearl: 12:08 (In a sarcastic tone) You really thought, you really thought you knew how to do this. That was your fatal flaw right there.
Gail: 12:18 I just have no clue! (Singing).
Pearl: 12:21 Nope like where is that? How are you doing that?
Pearl: 12:36 Where's the if?
Gail: 12:37 It's
Pearl: 12:38 It's under logic. I think I was just there.
Gail: 12:52 I thought I would bring some fun! (She drew with a pink marker on paper rather than debugging).
(…)
Pearl: 13:07 **Hey, I'm just deleting this square thing. This entire thing in the garbage.**
Gail: 13:14 So how do you still have surface colors?
Pearl: 13:17 I just put that one there.
Gail: 13:24 (While drawing lines on a piece of paper with red and blue markers) It needs to be red and blue and red and blue. Oh no, I think I just [inaudible] it needs to be blue red, blue red.
Pearl:13:44 **Why won't this clip into place?** *(She said this because blocks did not connect)*
Gail: 13:49 Like we actually know how to do this (Gail giggles and shakes her hands) No, no we don't.

Gail 13:53 Those dots may be a little small (referring to dots in the color maze that the robot was supposed to trace)
Pearl: 13:53 Gail is not covering those dots (speaking about Gail's drawing).
Pearl: 13:59 I'm just gonna enjoy myself (avoidance toward debugging).

This episode of Gail and Pearl's debugging illustrates fragmented inquiries. That is, their inquiries about what works or what does not work were not answered because they did not follow through the problem that they were seeing at the moment. Rather, they simply deleted their problematic code or exhibited little attempt to answer their own question as shown above. During the interview, Gail described her team's debugging approach when the robot made unintended or less desired movements as follows: "We were just kinda like, I guess this is as good as it's gonna get." She explained a rationale as follows:

He [the instructor] was telling us about how this is kind of common with the Ozobots. So he was like there's not too much you can do. You can try to mess with the wheel speed, but they're never going to go like perfectly straight.

These comments hint that Gail and Pearl's perceived uncontrollability in robot programming drove their lack of follow-through. If they were asked to list the changes that they made and the reasons for each change in relation to testing a series of hypotheses, they may have studied further about their inquires during debugging even when they did not aim to perfect their robot movement. In contrast, the why questions to which the scaffolded group were invited to respond seemed to help them follow through because they were asked to revisit and address inquires related to why some code changes worked or not. The important role of both why and why not in debugging in the present study is aligned with that in the design of a debugging support tool for text-based programming (Ko & Myers, 2008).

Theme 2. Use of scaffolds enabled the instructor to allow struggle during debugging without immediate help

The persistence observed in the scaffolded participants may be attributed to the instructor who allowed struggles in teams rather than providing immediate help. As shown in the dialogue below between the instructor and the team of Kiara and June in the class with scaffolds, this team continues their work without asking the instructor for help:

(Kiara and June worked on debugging task 1 to fix turn angles and delay between turns in the code)

Instructor: 02:47 So you're going through generating hypotheses and then you're gonna return to those hypotheses. See why they work, why they didn't, and just make some brief notes.
Instructor: 03:12 So as you make changes, just write about them, and work through these hypotheses.
Kiara: 03:20 It says it is still a 127-degree angle.
June: 03:20 I don't know how to do it. Yeah, maybe you do two of them [pause] maybe 290.

Kiara: 03:33 Why?
June: 03:37 okay so you want to keep it 50.
Kiara: 03:41 What do you have to do with the one second?
June: 03:43 Fifteen
Kiara: 03:52 Maybe, if you want to be safe just do one second and load it, I think it should be fine now.

Even when participants in the class with scaffolds asked the instructor for help, no immediate help was given. As seen in Diane and Sue's debugging episode below, the instructor was asked to help but he rather asked questions back to Sue that made her explore and find answers:

Sue: 04:03 How do you decide which colors it is going to see? (She asked the instructor) I don't know how to put these colors in the code.
Sue: 04:11 I just don't understand that.
Instructor: 04:15 Okay. So first we needed to trace to be able to follow a line, how it follows a line. So if we go to ... ***(The instructor waited for Sue to explore the block categories on the programming platform)*** **what would tell us to follow a line?**
Sue: 04:31 Line navigation
Instructor: 04:32 Cool. **Where would you put ... Which one would you choose first?**
Sue: 04:36 The first one?
Instructor: 04:37 Where would we put that?
Sue: 04:39 I think you'd put it at the very beginning.
Instructor: 04:42 Why?
Sue: 04:45 Because it needs to be that first.
Instructor: 04:49 Perfect. So that makes sense.

The instructor's questioning also seems to prepare the participants to respond to scaffolding prompts. For example, his *why* question about the change that Sue was proposing asked not only her rationale for code changes but also prompted to connect to hypothesis generation that was asked in the scaffolding. Considering that the instructor explained the task of responding to the scaffolding prompts to the class, he was aware of the expected process of problem solving. His awareness may have enabled his approach of allowing the participants to take time, struggle without immediate help, and think through what they do. This finding is similar to the synergistic effect of written scaffolds with teacher classroom enactment found in middle school chemistry classrooms in McNeill and Krajcik (2009).

In contrast, struggle did not last long among the participants without scaffolds. Rather, the instructor intervened quickly. This was perhaps because participants sought help from the instructor immediately when facing difficulties as illustrated below in the debugging episode of Jean and Meg:

(In this episode, Meg and Jean debugged the code to make the robot perform different actions depending on surface colors during debugging task 3)

Meg: 11:13 I was trying to, I raised my hand earlier but he [the instructor] didn't see it.

Meg: 11:20 (The instructor came to Meg's team table) So our question is, we understand how to do it, but these [blocks] won't attach. (Meg asked about intersection color related blocks)

Instructor: 11:27 So I think there is a block you can put in there, though, maybe down in logic. Yeah like this one here? Try putting that in there. Okay. And then do. Yeah, put that there and then do it. I think there's a get surface color back up in the *(The instructor pointed at the block on the programming platform).* **So you get surface color. You put that in the first part.**

Meg: 11:54 Thank you!

Instructor: 11:56 Good deal. And you'll use it again.

Jean: 11:58 Which one was it again?

Meg: 12:00 I'll show you in one second.

Jean: 12:03 Well, everything is messed up, I don't know how to fix it.

As hinted also in the class video below, the instructor in the class without scaffolds often gave the answer rather than guiding participants to find the answer themselves. This is understandable considering the time and other constraints in the classroom.

(The instructor showed how to code move forward, turn, and delay blocks so to make the robot travel on a square path during level 4 practice)

Instructor: 00:07 Okay. It's definitely it's turning. So that's where... Go to the movement. Okay, drag this out here and let's use that. Get rid of the move and put this in. And then after this, put a timing. Bring the seconds down. Instead try setting it for about two seconds or three seconds. Maybe three. Okay. Now here's what's happening. This should go straight. You can adjust it to make it go straight and the distance it goes is determined by this block. The timing block. Makes sense? So if you don't want it to go as far, back it off to 2.5 or 2. If you want to go further, increase it and then play with the rotate and see how that goes. And if you know it's already turning some, you might want to go ahead and say left like 35 or something so that it tries to straighten it.

However, providing for meaningful struggle is critical to learning to program and debug. As indicated in Gail's comments in her interview below, Gail did not get to practice the process of locating bugs when fixing robot wheel speeds and turns because the instructor told where Gail's team should attempt to fix before they began searching for the bug location. A search for where the error is in buggy code can be even more difficult than fixing the error (McCauley et al., 2008), but Gail lost the opportunity to learn to do so.

(Gail described the instructor's help with fixing wheel speeds and turns)

Interviewer: 08:37 Well, did you review the OzoBlockly code from the top to bottom or from bottom up? Or did you go specifically to the part of the code that you thought it was problematic?

Gail: 08:48 With the wheel, I would say we went specifically to the problem. Part of that was because the instructor had told us and we had like, we were working in class, so we saw some other groups messing with it. So we knew like, oh we should mess with the wheel speed.

The scaffolded group had a plenty of opportunities to search for errors in each debugging phase in the scaffolding (code reading, hypothesis generation, hypothesis testing, and reflection; see Fig. 2) but also benefited from strategies for deconstructing before debugging (Griffin, 2016). Given that the instructor in the class without scaffolds did not know about the debugging scaffolds implemented in the other class, unawareness of scaffolding techniques could be associated with instructional approaches to questions in class emerged from debugging.

Theme 3: Without scaffolds, participants often engaged in debugging alone

Sole debugging was often observed among participants in the class without scaffolds as illustrated in the following episodes from the video data. For example, in the debugging episode of Meg and Jean, Meg stated that she was going to show Jean how to debug the error when she was done with debugging on her own.

(In this episode, Jean and Meg worked on debugging task 3 to make the robot perform different actions depending on surface colors)

> *Meg: 11:59 I'll show you in one second. (She continued her debugging attempts alone)*
> *Jean: 12:03 Well, everything is messed up, I don't know how to fix it.*
> *Meg: 12:03 (Meg did not respond to her partner, Jean, but then heard a nearby group who was frustrated that their code was not working; Meg then spoke to them) So go to logic and the second one down has the equal sign. Put that one with the finding if... I'll show you. It's probably going to be easier for me to show you.*
> *Meg: 12:30 (Meg showed her computer screen to the student from the neighboring group to whom she was talking and continued the debugging task alone)*
> *Meg: 13:17 We could try a variety of different things that the kids just kind of have it more like an exploration. Because we only do this during centers. I feel like it should be more fun than doing a square. (Meg began discussions with Jean about preschool lesson design ideas for their field experience)*
> *Jean:13:37 I think that we should pick people that have letters in their names that are easy to write. And then they can do this. Like, we could guide them on how to make letters (Meg showed a surprised yet receptive expression).*

Everyone was asked to work collaboratively with their partner(s) in all classes but sole debugging was often observed in the class without scaffolds. Interestingly, lesson design was done collaboratively even in the context in which sole debugging was observed. When there was little discussion on how to fix programming errors, as shown above, Meg and Jean discussed their preschool lesson design ideas for their field experience. The discourse in such lesson design discussions was collective unlike the unidirectional discourse in debugging where Meg seemed to play both roles of a navigator and a driver (Lewis & Shah, 2015). That is, both contributions of Meg and Jean were made into their collaborative lesson design but not in debugging. This may have been because partners in a team taught together in their field experience as teaching partners and they perceived an equal role and ability in teaching preschoolers in the team as enrolled in the same block of their ECE program. Considering that Meg indicated her prior programming

knowledge as low to intermediate, she may have calibrated her debugging ability better than Jean and her neighboring peer and thought that it would be efficient to debug alone. However, sole debugging was often observed among other participants in class without scaffolds who had reported no prior programming knowledge. For example, Irina, who had indicated no prior programming knowledge, debugged alone as well. As seen below, Irina's partner, Todd, was not invited to her debugging process, similarly to the episode of Meg and Jean above in spite that Irina verbalized her thought processes loud and clear.

(In this episode, Irina and Todd attempted to debug wheel speed to make the robot move straight and forward during level 4 practice)

Irina: 09:04 I can't tell if it's leaning any way.
Irina: 09:10 And I mean, can I draw a line or will it start to follow the line? (Irina talked to a student in a nearby team; Todd was not part of this conversation; he did something on his phone)
Irina: 09:16 It's leaning a little bit.
Irina: 09:52 Okay. If it's leaning right, that means the right. I think I should speed up the right [wheel]. This is very confusing.
Todd:10:08 (Logged into another computer)
Irina: 11:33 Okay. If it's, it's leaning, it's going way right. That means I need to make the right [wheel] faster? Oh.

In contrast, participants in the class with scaffolds often exhibited collaborative debugging. They frequently exchanged questions and ideas and made changes and tested them together. The episodes included in Themes 1 and 2 sections also illustrate the observations between the more collaborative teams of Diane and Sue and Kiara and June and the less collaborative teams of Gail and Pearl and Meg and Jean. These contrasting observations bring into question the role of the scaffolds. Because the participants in the class with the scaffolds were asked to write about what they were doing and why as well as what they did and what happened as a result, they needed to articulate their debugging actions and rationales during debugging. It seemed that they chose to communicate about those actions and rationales with their partners as they debugged together, and, in consequence, the scaffolding worked to frame the task of debugging collaborative. The following episode of Kiara and June who were in the class with scaffolds demonstrates the process of collaborative debugging through questioning and hypothesizing elicited by the scaffolds. That is, they asked each other questions about what happened and why as prompted in the scaffolds.

(In this episode, Kiara and June debugged turn angles and timing during level 4 practice)

Kiara: 00:51 Wait, wait. **What happened?**
June: 00:54 **Does that make it slower?**
Kiara: 01:00 No, we need to **make it faster?** *And it needs to be like 30.*
June: 01:09 **Why 30?**
Kiara: 01:09 Because **if we make it 30, it will be like one second.**
Kiara: 01:32 **Should we change the angle?**
June: 01:33 No, I think that'll be enough. It will be the right ratio.

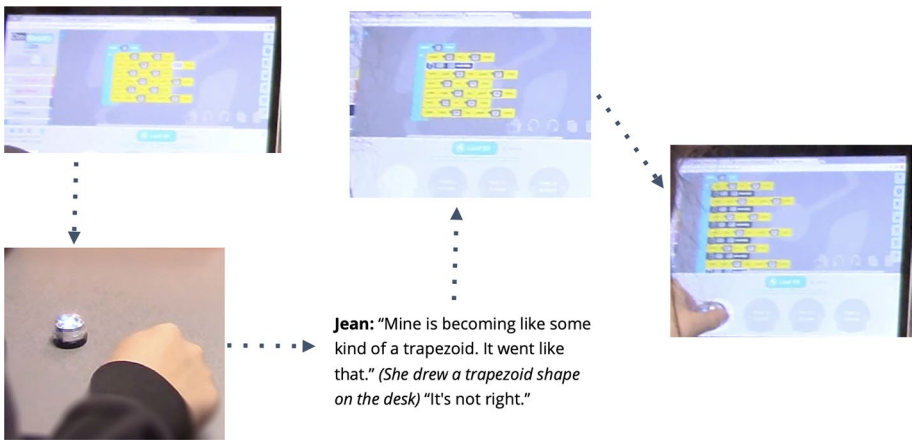


Fig. 4 Example trial and error during debugging

This process of collaborative reasoning during debugging observed among the participants with the scaffolds is a critical finding in this study given the importance of collective understanding in collaborative computing. This finding is critical also because our participants are future teachers whose role is crucial in facilitating children's reasoning and collaboration (Saye & Brush, 2002; van de Pol et al., 2015).

Theme 4. Regardless of scaffolds, all participants engaged in trial and error and used multimodal cues in embodied debugging

All participants used trial and error regardless of scaffolds. Figure 4 illustrates one round of trial and error in one of Jean's debugging episodes. She tested the code by observing her robot's performance, making another change in the code, and then loading the revised code to the robot to test its performance again. She and her partner, Meg, continued through multiple rounds of such trial and error, making incremental changes. The practice of trial and error is often considered unproductive and random (Grigoreanu et al., 2006; Ko et al., 2019; Murphy et al., 2008). However, when the scaffolded participants engaged in trial and error, they did so to test hypotheses. Their hypotheses were more general than specific. Their rationale for changes in the code was sometimes unsound but changes were not random. The responses of Kiara and June to the debugging scaffold depict intentional and cautious forms of trial and error. For example, Kiara hypothesized that the code needed a loop. Her explanation for the hypothesis was written in the present tense, which hints that she tried out the loop addition and saw its result before or while responding to the prompt about reasoning for the hypothesis. That is, each hypothesis involved incremental changes, and thus the rounds of hypothesizing and testing were the process of trial and error.

(Kiara's responses to a section of the debugging scaffold prompts during debugging task 1)

Hypothesis 1: Add a loop.
Reason for hypothesis 1: It needs two movements.
Hypothesis 2: Change the angle to 90°.

Reason for hypothesis 2: 73° was not enough.
Hypothesis 3: Add a movement (rotate).
Reason for hypothesis 3: *I need the robot to turn 180° angle.* **I think adding a loop works perfectly.**

(June's responses to a section of the debugging scaffold prompts during debugging task 1)

Hypothesis 2: Angles need to be changed.
Changes made to test hypothesis 2: 73° ---> 90° 127° ---> 90° & 90°
Could not be 180°
Why did you choose to make these changes specifically: Because our bot was not rotating enough.
What happened after these changes are made: Still was not making full turns.
Tried different angles until successful (110°) (120° & 100°)

A few researchers argue that there are distinctive forms of trial and error (Kim et al., 2020, 2021; Fitzgerald et al., 2010; Perkins et al., 1986). Considering that the scaffolded group eventually noted actual causes of the bugs, trial and error can be used in productive ways. This is a critical observation in contrast to the group without scaffolds who often concluded debugging by questioning their instructor about the problem in the code, as hinted in the discourse of the team below:

Irina: 08:55 I had tinkered with, I feel like every aspect of it, it's just like not working.
Todd: 09:01 Yes.
Irina: 09:02 I still cannot get mine to go in a square correctly and I don't know what I'm doing wrong. I'll show you. Watch. Okay, so I moved it down to just 31 and 30. ***(She now talked to the instructor to get help)***

This may have been because the instructor in the class without scaffolds often provided immediate help (see Theme 2 above), which made it easier and quicker for the participants to ask the instructor to explain why the code was not working than to figure it out for themselves. They were not asked to hypothesize why the code was not working. During the interview, Gail from the class without scaffolds explained that her robot did not move straight but veered left. When asked about the cause of the problem, she noted that her team had never figured out how to exactly code wheel speeds because of the instructor's immediate help but also his comments on the technical limitations of the robots (see the interview quote in Theme 1).

During trial and error, bodily movement was often observed in both classes of participants regardless of scaffolds. In the following episode of the team from the class without scaffolds, Jean assembled blocks in her code and performed hand gestures that simulate her robot tracing a circle based on different wheel speeds:

(In this episode, Meg and Jean worked to debug wheel speeds so the robot could make a circle as part of debugging task 2)

Jean: 05:48 I'm just gonna wing it.
Meg: 06:58 Oh, the circle is just setting the wheel speeds differently. Like so do one 30

and one 60.
Jean: 07:10 How do you know?
Meg: 07:10 Because one will go 60. One's gonna go faster, it's gonna like... (Jean performed hand gestures to show that different wheel speeds made the robot travel in a circle)

Participants often engaged in hand drawing while debugging. In the episodes, June suggested drawing the pattern the robot traced, Pearl suggested attaching a pen to the robot to draw its path, and Irina used curvy hand gestures to show the robot was off the desired right angle. In all episodes, participants used or expressed the desire to use multimodal cues to visualize the robot performance, which in turn would inform debugging:

(In this episode, June and Kiara worked to make the robot travel on a square path during level 4 practice)

June: 07:40 (She watched the robot's movement with Kiara) It was so close.
Kiara: 07:40 Let's draw it, it is hard to tell why. How about we start it in the same place every time.

(In this episode, Gail and Pearl worked to debug turn angles to make the robot travel on a square path during level 4 practice)

Pearl: 02:23 I need to attach a pencil to it so I can see what shape it makes.
Gail: 02:26 This is a good idea. We should do that. But then it would affect it.

(In this episode, Irina explained to her partner that the robot made right angles during level 4 practice)

Irina: 02:22 Now it's, it's so off it is not, it goes from a square to a diamond back to a square. Like I was it doing like, it's like the angles are like 180.
Irina: 02:53 Oh okay. Whoa, Whoa, Whoa. That is not a square. I don't even know what that is (laughing).
Irina: 03:13 It is way too far. So he's going like wooo... (She used curvy hand gestures to indicate robot movements that were not 90-degree turns).

The scaffolding design can be improved to include prompts for embodied reasoning (Abrahamson et al., 2011) during debugging. Embodied interactions (Fadjo, 2012; Fadjo et al., 2008; Romero et al., 2009) could be studied further to benefit embodied learning in programming. For example, the debugging scaffold could prompt participants to act out as if they were the robots when struggling with angle value in a rotate block to understand the angle of the robot's actual turn. It may be helpful to integrate strategies using embodiment in teaching debugging in unplugged activities (Ahn et al., 2021).

General discussion

Supporting ECE teacher candidates' ill-structured problem solving during debugging is critical, though the literature on scaffolded debugging of block-based code lacks. This study reports use of the initial design of a scaffolding prototype in an ECE teacher education undergraduate course on integrated arts in early childhood as part of design-based research. The scaffolding design was grounded in the conceptual framework of the

Table 5 A summary of findings and possible interpretations related to scaffolding design

Finding	Possible attribution to scaffolding design decisions
Persistence through debugging using scaffolds (Theme 1)	Strategic scaffolding (Belland et al., 2017a, b; Belland, 2017; Hannafin et al., 1999) that • Structured and problematized (Reiser, 2004) the debugging process through four-phased debugging activities • Promoted hypothesis-driven debugging (Kim et al., 2018; Fitzgerald et al., 2008; Vessey, 1985) • Promoted perceived controllability (Kim & Pekrun, 2014; Bandura, 1997; Wigfield & Eccles, 2000) enabled through alternative hypotheses • Justified prompted tasks to promote expectancy for success (Belland et al., 2013) • Modeled example responses to scaffolding prompts (Belland et al., 2013; Ko et al., 2019; van de Pol et al., 2011)
Productive struggle facilitated through synergy between scaffolds and the instructor (Theme 2)	Scaffolding offered • Multiple opportunities to search for errors (Kim et al., 2018) • Strategies of reading before writing (Griffin, 2016) • Why and why not questions (Ko & Myers, 2008)
Collaborative reasoning for debugging (Theme 3)	Scaffolding provided/promoted • Question prompts (Belland et al., 2013; van de Pol et al., 2011) • Why and why not questions (Ko & Myers, 2008) • Reflective debugging (Kim et al., 2018)
Trial-and-error and embodied debugging (Theme 4)	Scaffolding provided • No parameter for specificity in hypotheses in scaffolds • No prompts related to hands-on problem-solving with multimodal objects

Notes. Themes emerged when considering the current data in light of our framework, which was informed by the literature. Further detail on the connection between our findings and the literature are discussed in the findings and discussion section as well as the general discussion section

study described earlier. Early childhood teacher candidates in one course section were provided the scaffolding prototype, and their counterparts from another section were not. Findings are discussed above in each theme, and listed in Table 5, and also discussed collectively below.

The scaffolding in the present study seems to have worked as an adaptive motivator in that participants persisted more in debugging despite the puzzlement they experienced during the process. This may be from the prompts that asked participants to document testing multiple hypotheses. Such prompts may have helped participants see their struggles as momentary failures—a natural part of hypothesis testing. The debugging literature suggests that considering alternative hypotheses is a critical skill for effective debugging (Jonassen & Hung, 2006) though novice programmers rarely do so (Lee et al., 2014; Li et al., 2019; Murphy et al., 2008). The scaffolding may have contributed to the perception of hypotheses as malleable statements that can be fine-tuned or discarded, which in turn led to participants being flexible in generating alternative hypotheses and solutions, and therefore persisting more than participants without scaffolding.

The finding of the scaffolded group's persistence may be associated also with struggle that the instructor allowed. It was observed that participants without the debugging scaffold quickly looked for external help. The instructor intervened early and gave students the answer before they struggled through difficulties. This is decidedly

not a scaffolding approach, as providing answers directly represents a high level of teacher control, and when that is the primary strategy used, it represents a non-contingent form of instructional support (van de Pol et al., 2019). The absence of scaffolding and the instructor's approach to providing the answer to participants' questions undermined the development of strong debugging skills as learners did not experience the struggle and productive failure that are inherent to debugging (McCauley et al., 2008; Searle et al., 2018). These findings are supported by research on ill-structured productive failure designs, which indicates that those allowed to struggle and learn from failure outperform their counterparts who receive direct instruction (Kapur, 2015). The findings also align with those of McNeill and Krajcik (2009), who found that student learning was optimal when teacher guidance was generic and computer-based scaffolds context-specific, as opposed to when teacher guidance was context-specific and computer-based scaffolds generic. Furthermore, it can be seen that in the condition that used computer-based scaffolds, the support from the scaffolds and that of the teacher served together synergistically to support optimal student learning (McNeill & Krajcik, 2009; Tabak, 2004). These findings call for further studies of in what ways the scaffold facilitates novice debuggers' persistence and also the instructor's practice that yields productive struggle.

Participants without scaffolds often resorted to individual debugging despite being assigned to collaborative teamwork. Contrarily, those who had the scaffolding were more inclined to collaboratively work through exchange of ideas and negotiation of debugging strategies. Empirical studies suggest that pair programming is an effective approach that leads to meaningful learning gains for novice programmers (Kim et al., 2020, 2021; Denner et al., 2014; Durak et al., 2019; Wei et al., 2021). A meta-analysis of empirical studies on pair versus solo programming found similar results, and added that pair programming resulted in enhanced persistence if guidance on pair programming is available (Umapathy & Ritzhaupt, 2017). While explicit training on programming with a peer was not included in this study, findings suggest that scaffolding prompts about hypothesis generation, testing, and reasoning served to some extent as guidance for collaborative debugging.

Findings from this study also revealed that, regardless of the scaffolding condition, participants in both groups adopted embodied reasoning during debugging. Embodied cognition theories assert that one's bodily interactions with the surrounding environment are key to their cognitive processes (Anderson, 2003; Shapiro & Stolz, 2019). In the present study, participants used hand drawing and hand gestures to make sense of and explain robot performance to their peers. Research on embodied debugging is growing, and empirical results are promising. Ahn et al. (2017) found that exposure to different types of embodied cognition was more effective in helping children deal with code errors than no exposure at all. Fadjo (2012) found that middle schoolers' embodiment and performance of specific actions prior to programming led to increased use of code to program those same actions. Inclusion of scaffolds for embodied reasoning will be considered in future redesign of the scaffolding so that ECE teacher candidates can be introduced to embodied reasoning strategies during block-based programming practice and potentially use such strategies during debugging tasks.

Another intriguing finding was that hypothesis-driven debugging processes still involved much use of trial-and-error among the scaffolded participants. This finding is counterintuitive to the perspective in which trial-and-error is viewed as practice out of randomness (Grigoreanu et al., 2006; Ko et al., 2019; Murphy et al., 2008). Interestingly, participants in the scaffolded group performed trial and error to fine-tune robot movements such as making a turn during the process of testing a hypothesis. Also, the scaffolded group often noted the actual cause of the bugs. This may not be too unexpected in that the scaffolded participants were asked to write about the actual cause of the bug after testing their hypothesized bug. In this sense, these findings showcase productive ways of *reflective* trial and error. Scaffolding prompts specifically on what actually happened after changes and what actually caused the malfunctioning robot seem to have facilitated their noticing of the actual cause for bugs.

Conclusions

This study offers insights into ECE teacher candidates' experiences with and without scaffolds during debugging. The group using the scaffolding prototype experienced persistence, productive struggle, and collaborative debugging. Reflective trial and error practice was employed as debugging strategies in the scaffolded group. These findings inform the redesign of the scaffolding prototype. Findings are not generalizable but give insights for diverse debugging approaches which can be applied to the context in which computer science education is pursued for diverse populations including underrepresented groups in computer science.

The current study suggests several areas for future research. As computer science education efforts are made nationally and internationally, it is important to consider scaffolding the learning-to-code process to groups often unassociated with the computer science field. Non-computer science majors should not and cannot be dismissed from learning foundational computer science that is central to digital literacy in the future (Bers, 2019). This study provides insight into scaffolding techniques to help ECE teacher candidates debug. As advances are made across computing disciplines such as data science, cybersecurity, performance computing, and high-performance computing, the workforce should be prepared to meet rapid innovation demands. A major sector of this workforce are teachers, specifically early childhood teachers. Beginning computer science education early is crucial. Therefore scaffolding early childhood teachers to learn programming and debugging is crucial as well. This study findings suggest our scaffolding prototype helped with persistence, productive struggle, and collaborative debugging. Although further investigation is needed to understand how specific scaffolding techniques promote computer science for all, it is encouraging that early childhood teachers are practicing computational literacy.

Acknowledgements
Not applicable.

Authors' contributions
CK and BRB planned the study, secured the funding, collected data, analyzed the data, and wrote up the manuscript. LV, DU, and CG assisted CK and BRB with data collection and analysis. All five authors contributed to writing. The authorship order is in the order from the most to the least contributions. All authors read and approved the final manuscript.

Funding
This work has been supported by grants 1927595 and 1906059 from the National Science Foundation (NSF) in the United States. Any opinions, findings, and or conclusions are those of the authors and do not necessarily represent official positions of NSF.

Availability of data and materials
The instructional material is available upon request to the first author. The data of the video-recorded participants cannot be shared openly because they cannot be anonymized completely.

Declarations

Competing interests
The authors declare no competing interests.

Author details
[1]Learning, Design, and Technology, Educational Psychology, College of Education, The Pennsylvania State University, 314D Keller Building, University Park, PA 16802, USA. [2]Educational Studies, College of Education, University of South Carolina, Columbia, USA. [3]Educational Psychology, College of Education, The Pennsylvania State University, University Park, PA, USA. [4]Computer Education and Educational Technology, Boğaziçi University, Istanbul, Turkey. [5]Department of Curriculum and Instruction, College of Education, Tennessee Tech University, Cookeville, USA.

Received: 8 July 2021 Accepted: 10 January 2022
Published online: 10 March 2022

References

Abrahamson, D., Trninic, D., Gutiérrez, J. F., Huth, J., & Lee, R. G. (2011). Hooks and shifts: A dialectical study of mediated discovery. *Technology, Knowledge and Learning, 16*(1), 55–85. https://doi.org/10.1007/s10758-011-9177-y

Ahn, J., Mao, Y., Sung, W., & Black, J. B. (2017). Supporting debugging skills: Using embodied instructions in children's programming education. In *Proceedings of Society for Information Technology & Teacher Education International Conference* (pp. 19–26). Association for the Advancement of Computing in Education (AACE).

Ahn, J., Sung, W., & Black, J. B. (2021). Unplugged debugging activities for developing young learners' debugging skills. *Journal of Research in Childhood Education*. https://doi.org/10.1080/02568543.2021.1981503

Anderson, M. L. (2003). Embodied cognition: A field guide. *Artificial Intelligence, 149*(1), 91–130. https://doi.org/10.1016/S0004-3702(03)00054-7

Ardimento, P., Bernardi, M. L., Cimitile, M., & Ruvo, G. D. (2019). Reusing bugged source code to support novice programmers in debugging. *ACM Transactions on Computing Education, 20*(1), 1–24. https://doi.org/10.1145/3355616

Ashiabi, G. S. (2007). Play in the preschool classroom: Its socioemotional significance and the teacher's role in play. *Early Childhood Education Journal, 35*(2), 199–207. https://doi.org/10.1007/s10643-007-0165-8

Bandura, A. (1997). *Self-efficacy: The exercise of control* (pp. ix, 604). W H Freeman/Times Books/ Henry Holt & Co.

Belland, B. R. (2017). Instructional scaffolding in STEM education. Springer International Publishing. https://doi.org/10.1007/978-3-319-02565-0.

Belland, B. R., Kim, C., & Hannafin, M. J. (2013). A framework for designing scaffolds that improve motivation and cognition. *Educational Psychologist*, *48*(4), 243–270. https://doi.org/10.1080/00461520.2013.838920.

Belland, B. R., Walker, A. E., & Kim, N. J. (2017a). A Bayesian network meta-analysis to synthesize the influence of contexts of scaffolding use on cognitive outcomes in stem education. *Review of Educational Research*, *87*(6), 1042–1081. https://doi.org/10.3102/0034654317723009.

Belland, B. R., Walker, A. E., Kim, N. J., & Lefler, M. (2017b). Synthesizing results from empirical research on computer-based scaffolding in STEM education: A meta-analysis. *Review of Educational Research*, *87*(2), 309–344. https://doi.org/10.3102/0034654316670999.

Bers, M. U. (2018a). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge, Taylor & Francis Group.

Bers, M. U. (2018b). Coding, playgrounds and literacy in early childhood education: The development of KIBO robotics and ScratchJr. *IEEE Global Engineering Education Conference (EDUCON), 2018*, 2094–2102. https://doi.org/10.1109/EDUCON.2018.8363498

Bers, M. U. (2019). Coding as another language: A pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education, 6*(4), 499–528. https://doi.org/10.1007/s40692-019-00147-3

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.

Boyatzis, R. E. (1998). *Transforming qualitative information: Thematic analysis and code development*. Sage.

Breazeal, C., Harris, P. L., DeSteno, D., Kory Westlund, J. M., Dickens, L., & Jeong, S. (2016). Young children treat robots as informants. *Topics in Cognitive Science, 8*(2), 481–491. https://doi.org/10.1111/tops.12192

Brennan, K., & Resnick, M. (2013). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In C. Mouza & N. Lavigne (Eds.), *Emerging technologies for the classroom* (pp. 253–268). Springer, New York. https://doi.org/10.1007/978-1-4614-4696-5_17

Brooker, E., Blaise, M., & Edwards, S. (Eds.). (2014). *SAGE handbook of play and learning in early childhood*. SAGE.

Çetin, M., & Demircan, H. Ö. (2020). Empowering technology and engineering for STEM education through programming robots: A systematic literature review. *Early Child Development and Care, 190*(9), 1323–1335. https://doi.org/10.1080/03004430.2018.1534844

Del Sole, A. (2019). Running and debugging code. In: A. Del Sole (Ed.), *Visual studio code distilled: evolved code editing for windows, macOS, and Linux* (pp. 191–209). Apress. https://doi.org/10.1007/978-1-4842-4224-7_9

Demetriadis, S. N., Papadopoulos, P. M., Stamelos, I. G., & Fischer, F. (2008). The effect of scaffolding students' context-generating cognitive activity in technology-enhanced case-based learning. *Computers & Education, 51*, 939–954. https://doi.org/10.1016/j.compedu.2007.09.012

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education, 46*(3), 277–296. https://doi.org/10.1080/15391523.2014.888272

Durak, H. Y., Yilmaz, F. G. K., & Yilmaz, R. (2019). Computational thinking, programming self-efficacy, problem solving and experiences in the programming process conducted with robotic activities. *Contemporary Educational Technology, 10*(2), 173–197. https://doi.org/10.30935/cet.554493

Fadjo, C. L. (2012). *Developing computational thinking through grounded embodied cognition* [Columbia University]. https://doi.org/10.7916/D88058PP

Fadjo, C. L., Shin, J., Lu, M.-S., Chan, M., & Black, J. (2008). *Embodied cognition and video game programming*. 5749–5756. https://www.learntechlib.org/primary/p/29179/

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

Fitzgerald, S., McCauley, R., Hanks, B., Murphy, L., Simon, B., & Zander, C. (2010). Debugging from the student perspective. *IEEE Transactions on Education, 53*(3), 390–396. https://doi.org/10.1109/TE.2009.2025266

Griffin, J. M. (2016). Learning by taking apart: Deconstructing code by reading, tracing, and debugging. *Proceedings of the 17th annual conference on information technology education—SIGITE '16*, 148–153. https://doi.org/10.1145/2978192.2978231

Grigoreanu, V., Beckwith, L., Fern, X., Yang, S., Komireddy, C., Narayanan, V., Cook, C., & Burnett, M. (2006). Gender differences in end-user debugging, revisited: What the miners found. *Visual Languages and Human-Centric Computing (VL/HCC'06)*, 19–26. https://doi.org/10.1109/VLHCC.2006.24

Hannafin, M., Land, S., & Oliver, K. (1999). Open-ended learning environments: Foundations, methods, and models. In C. M. Reigeluth (Ed.), *Instructional design theories and models: Volume II: A new paradigm of instructional theory* (pp. 115–140). Lawrence Erlbaum Associates.

Holland, J. (2009). *A constraint-based ITS for the Java Programming Language* [University of Canterbury]. http://132.181.17.3/research/reports/MastTheses/2009/mast_0906.pdf

Jonassen, D. H., & Hung, W. (2006). Learning to troubleshoot: A new theory-based design architecture. *Educational Psychology Review, 18*(1), 77–114. https://doi.org/10.1007/s10648-006-9001-8

Kapur, M. (2015). Learning from productive failure. *Learning: Research and Practice, 1*(1), 51–65. https://doi.org/10.1080/23735082.2015.1002195

Katz, I. R., & Anderson, J. R. (1987). Debugging: An analysis of bug-location strategies. *Human-Computer Interaction, 3*(4), 351–399. https://doi.org/10.1207/s15327051hci0304_2

Kazakoff, E. R., & Bers, M. U. (2014). Put your robot in, put your robot out: Sequencing through programming robots in early childhood. *Journal of Educational Computing Research, 50*(4), 553–573. https://doi.org/10.2190/EC.50.4.f

Kim, C., & Pekrun, R. (2014). Emotions and motivation in learning and performance. In J. M. Spector, M. D. Merrill, J. Elen, & M. J. Bishop (Eds.), *Handbook of research on educational communications and technology* (pp. 65–75). New York: Springer. https://doi.org/10.1007/978-1-4614-3185-5_6.

Kim, C., Kim, D., Yuan, J., Hill, R. B., Doshi, P., & Thai, C. N. (2015). Robotics to promote elementary education pre-service teachers' STEM engagement, learning, and teaching. *Computers & Education*, *91*, 14–31. https://doi.org/10.1016/j.compedu.2015.08.005.

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.

Kim, C., Belland, B. R., & Gleasman, C. (2020). Playful coding and playful learning among future early childhood educators. In Gresalfi, M. & Horn, I. S (Eds.), *The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS) 2020.* (4), (pp. 2411–2412). Nashville, TN: International Society of the Learning Sciences.

Kim, C., Belland, B. R., Baabdullah, A., Lee, E., Dinç, E., & Zhang, A. Y. (2021). An ethnomethodological study of abductive reasoning while tinkering. *AERA Open*, *7*, 23328584211008110. https://doi.org/10.1177/23328584211008111.

Ko, A. J., LaToza, T. D., Hull, S., Ko, E. A., Kwok, W., Quichocho, J., Akkaraju, H., & Pandit, R. (2019). Teaching explicit programming strategies to adolescents. *Proceedings of the 50th ACM technical symposium on computer science education*, 469–475. https://doi.org/10.1145/3287324.3287371

Ko, A. J., & Myers, B. A. (2008). Debugging reinvented: Asking and answering why and why not questions about program behavior. *Proceedings of the 30th international conference on software engineering*, 301–310. https://doi.org/10.1145/1368088.1368130

Lee, M. J., Bahmani, F., Kwan, I., LaFerte, J., Charters, P., Horvath, A., Luor, F., Cao, J., Law, C., Beswetherick, M., Long, S., Burnett, M., & Ko, A. J. (2014). *Principles of a debugging-first puzzle game for computing education*. 57–64. https://doi.org/10.1109/VLHCC.2014.6883023

Lewis, C. M., & Shah, N. (2015). How equity and inequity can emerge in pair programming. *Proceedings of the eleventh annual international conference on international computing education research*, 41–50. https://doi.org/10.1145/2787622.2787716

Li, C., Chan, E., Denny, P., Luxton-Reilly, A., & Tempero, E. (2019). Towards a framework for teaching debugging. *Proceedings of the twenty-first Australasian computing education conference on—ACE '19*, 79–86. https://doi.org/10.1145/3286960.3286970

Luxton-Reilly, A., McMillan, E., Stevenson, E., Tempero, E., & Denny, P. (2018). Ladebug: An online tool to help novice programmers improve their debugging skills. *Proceedings of the 23rd Annual ACM conference on innovation and technology in computer science education*, 159–164. https://doi.org/10.1145/3197091.3197098

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*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Lytle, N., Dong, Y., Cateté, V., Milliken, A., Isvik, A., & Barnes, T. (2019). Position: Scaffolded coding activities afforded by block-based environments. *2019 IEEE blocks and beyond workshop (B B)*, 5–7. https://doi.org/10.1109/BB48857.2019.8941203

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

McNeill, K. L., & Krajcik, J. (2009). Synergy between teacher practices and curricular scaffolds to support students in using domain-specific and domain-general knowledge in writing arguments to explain phenomena. *Journal of the Learning Sciences, 18*(3), 416–460. https://doi.org/10.1080/10508400903013488

Michaeli, T., & Romeike, R. (2019). Current status and perspectives of debugging in the k12 classroom: A qualitative study. *IEEE EDUCON '19*, 1030–1038.

Morse, J. M. (1994). Designing qualitative research. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of qualitative inquiry* (pp. 220–235). Thousand Oaks.

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: The good, the bad, and the quirky: A qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin, 40*(1), 163–167. https://doi.org/10.1145/1352322.1352191

Näykki, P., Isohätälä, J., & Järvelä, S. (2021). "You really brought all your feelings out": Scaffolding students to identify the socio-emotional and socio-cognitive challenges in collaborative learning. *Learning, Culture and Social Interaction, 30*, 100536. https://doi.org/10.1016/j.lcsi.2021.100536

Neutens, T., & Wyffels, F. (2020). Analyzing coding behaviour of novice programmers in different instructional settings: Creating vs. Debugging. *2020 International conference on computational science and computational intelligence (CSCI)*, 892–897. https://doi.org/10.1109/CSCI51800.2020.00167

Papadakis, S., & Kalogiannakis, M. (2019). Evaluating a course for teaching introductory programming with Scratch to pre-service kindergarten teachers. *International Journal of Technology Enhanced Learning, 11*(3), 231–246. https://doi.org/10.1504/IJTEL.2019.100478

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research, 2*(1), 37–55. https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

Quan, G. M., & Gupta, A. (2020). Tensions in the productivity of design task tinkering. *Journal of Engineering Education, 109*(1), 88–106. https://doi.org/10.1002/jee.20303

Reiser, B. J. (2004). Scaffolding complex learning: The mechanisms of structuring and problematizing student work. *The Journal of the Learning Sciences, 13*(3), 273–304. https://doi.org/10.1207/s15327809jls1303_2

Romero, P., du Boulay, B., Robertson, J., Good, J., & Howland, K. (2009). Is embodied interaction beneficial when learning programming? In R. Shumaker (Ed.), *Virtual and mixed reality* (pp. 97–105). Springer.

Saldaña, J. (2016). *The coding manual for qualitative researchers* (3rd ed.). SAGE.

Saye, J. W., & Brush, T. (2002). Scaffolding critical reasoning about history and social issues in multimedia-supported learning environments. *Educational Technology Research and Development, 50*(3), 77–96. https://doi.org/10.1007/BF02505026

Searle, K. A., Litts, B. K., & Kafai, Y. B. (2018). Debugging open-ended designs: High school students' perceptions of failure and success in an electronic textiles design activity. *Thinking Skills and Creativity, 30*, 125–134. https://doi.org/10.1016/j.tsc.2018.03.004

Shapiro, L., & Stolz, S. A. (2019). Embodied cognition and its significance for education. *Theory and Research in Education, 17*(1), 19–39. https://doi.org/10.1177/1477878518822149

Smolucha, L., & Smolucha, F. (2021). Vygotsky's theory in-play: Early childhood education. *Early Child Development and Care, 191*(7–8), 1041–1055. https://doi.org/10.1080/03004430.2020.1843451

Socratous, C., & Ioannou, A. (2021). Structured or unstructured educational robotics curriculum? A study of debugging in block-based programming. *Educational Technology Research and Development*. https://doi.org/10.1007/s11423-021-10056-x

Spinellis, D. (2018). Modern debugging: The art of finding a needle in a haystack. *Communications of the ACM, 61*(11), 124–134. https://doi.org/10.1145/3186278

Su, J.-M. (2020). A rule-based self-regulated learning assistance scheme to facilitate personalized learning with adaptive scaffoldings: A case study for learning computer software. *Computer Applications in Engineering Education, 28*(3), 536–555. https://doi.org/10.1002/cae.22222

Sullivan, A., Strawhacker, A., & Bers, M. U. (2017). Dancing, drawing, and dramatic robots: Integrating robotics and the arts to teach foundational STEAM concepts to young children. In M. S. Khine (Ed.), *Robotics in STEM education: Redesigning the learning experience* (pp. 231–260). Springer International Publishing.

Sullivan, F. R., & Moriarty, M. A. (2009). Robotics and discovery learning: Pedagogical beliefs, teacher practice, and technology integration. *Journal of Technology and Teacher Education, 17*(1), 109.

Tabak, I. (2004). Synergy: A complement to emerging patterns of distributed scaffolding. *Journal of the Learning Sciences, 13*(3), 305–335. https://doi.org/10.1207/s15327809jls1303_3

Tracy, S. J. (2020). *Qualitative research methods: Collecting evidence, crafting analysis, communicating impact* (2nd ed.). Wiley Blackwell.

Trilles, S., & Granell, C. (2020). Advancing preuniversity students' computational thinking skills through an educational project based on tangible elements and virtual block-based programming. *Computer Applications in Engineering Education, 28*(6), 1490–1502. https://doi.org/10.1002/cae.22319

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education, 17*(4), 1–13. https://doi.org/10.1145/2996201

Vaismoradi, M., Jones, J., Turunen, H., & Snelgrove, S. (2016). Theme development in qualitative content analysis and thematic analysis. *Journal of Nursing Education and Practice, 6*(5), 100–110. https://doi.org/10.5430/jnep.v6n5p100.

van de Pol, J., Volman, M., & Beishuizen, J. (2011). Patterns of contingent teaching in teacher–student interaction. *Learning and Instruction*, *21*(1), 46–57. https://doi.org/10.1016/j.learninstruc.2009.10.004.
van de Pol, J., Mercer, N., & Volman, M. (2019). Scaffolding student understanding in small-group work: Students' uptake of teacher support in subsequent small-group interaction. *Journal of the Learning Sciences, 28*(2), 206–239. https://doi.org/10.1080/10508406.2018.1522258
van de Pol, J., Volman, M., Oort, F., & Beishuizen, J. (2015). The effects of scaffolding in the classroom: Support contingency and student independent working time in relation to student achievement, task effort and appreciation of support. *Instructional Science, 43*(5), 615–641. https://doi.org/10.1007/s11251-015-9351-z
Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies, 23*(5), 459–494. https://doi.org/10.1016/S0020-7373(85)80054-7
Vessey, I. (1986). Expertise in debugging computer programs: An analysis of the content of verbal protocols. *IEEE Transactions on Systems, Man & Cybernetics, 16*(5), 621.
Wei, X., Lin, L., Meng, N., Tan, W., Kong, S., & Kinshuk. (2021). The effectiveness of partial pair programming on elementary school students' computational thinking skills and self-efficacy. *Computers & Education, 160*, 15. https://doi.org/10.1016/j.compedu.2020.104023
Weiner, B. (1985). An attributional theory of achievement motivation and emotion. *Psychological Review, 92*(4), 548–573. https://doi.org/10.1037/0033-295X.92.4.548
Wigfield, A., & Eccles, J. S. (2000). Expectancy–value theory of achievement motivation. *Contemporary Educational Psychology, 25*(1), 68–81. https://doi.org/10.1006/ceps.1999.1015
Williams, R., Park, H. W., Oh, L., & Breazeal, C. (2019). Popbots: Designing an artificial intelligence curriculum for early childhood education. *Proceedings of the AAAI Conference on Artificial Intelligence, 33*(01), 9729–9736. https://doi.org/10.1609/aaai.v33i01.33019729
Wood, D., Bruner, J., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry, 17*, 89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x
Yoon, B.-D., & Garcia, O. N. (1998). Cognitive activities and support in debugging. *Proceedings Fourth Annual Symposium on Human Interaction with Complex Systems*, 160–169. https://doi.org/10.1109/HUICS.1998.659974

Publisher's Note