An Analysis of Middle Grade Teachers' Debugging Pedagogical Content Knowledge

Jennifer Tsan
University of Chicago
Chicago, IL, USA
jennifertsan@uchicago.edu

David Weintrop
University of Maryland
College Park, MD, USA
weintrop@umd.edu

Diana Franklin
University of Chicago
Chicago, IL, USA
dmfranklin@uchicago.edu

ABSTRACT

There is an increasing need for knowledgeable K-12 computer science (CS) teachers. It is necessary to inform teachers how to debug and help their students debug programs. Research has shown that debugging is difficult for novices because the process requires different skills from creating programs and instructing students how to debug can help them acquire these skills. To this end, we developed a CS professional development for middle grade teachers (grades 5th-8th/ages 10-13) that includes lessons on debugging. The teachers completed debugging activities that involved finding bugs in Scratch programs and explaining how they would help their students in debugging. We qualitatively analyzed their responses and found that teachers successfully identified the problem but they struggled to locate it in the code. In considering how they would help students who had such a bug, the teachers often focused on helping the student find a solution for the bug rather than on identifying the problem or its source. Finally, teachers' ability to identify bugs and the pedagogical strategies to engage students in this process differed based on CS teaching experience and prior CS knowledge. This work contributes to our understanding of teachers' debugging abilities and advances our knowledge on how to support teachers in teaching their students how to debug their programs.

CCS CONCEPTS

• **Social and professional topics** → **K-12 education**; **Computing education**;

KEYWORDS

debugging, K-8, teachers

ACM Reference Format:
Jennifer Tsan, David Weintrop, and Diana Franklin. 2022. An Analysis of Middle Grade Teachers' Debugging Pedagogical Content Knowledge. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3502718.3524770

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland

ACM ISBN 978-1-4503-9201-3/22/07…$15.00
https://doi.org/10.1145/3502718.3524770

1 INTRODUCTION

Bugs are a common part of writing programs regardless of whether the program is being written by a novice in Scratch or a professional programmer in Python or C++. However, when programmers start debugging their code, the differences between novices and experts become obvious [23, 45]. Identifying bugs in code is difficult for novices. This is in part due to the fact that replicating and isolating bugs requires skills that are different from programming knowledge [12, 25]. These skills include an understanding of the programming language being used, general knowledge about programming and how programs execute, and knowledge about common bugs and effective debugging strategies [12].

Since debugging is difficult for novices, students may need help from teachers to get "unstuck" [32]. Studies have shown that providing debugging instruction is effective for helping students learn debugging skills [9, 10]. While researchers have explored ways to teach students debugging methods [31, 33, 34], there has been little research on teaching CS teachers how to debug or to teach debugging, especially at the K-12 level [26]. As with any subject, teachers both need the skill themselves and knowledge on how to teach it in order to effectively teach debugging. Additionally, there is a gap in debugging research on how teachers at the K-12 level currently teach their students debugging. To this end, we developed a computer science (CS) professional development (PD) program for middle grade teachers (grades 5th-8th/ages 10-14) that includes activities to develop teachers' debugging pedagogical content knowledge. As part of the PD, teachers complete debugging activities that involved finding bugs in Scratch programs and are asked to explain how they would help their students debug those same Scratch programs. More specifically, in this paper we pursue the following research questions:

- After being introduced to the "WHAT?!? A MESS" strategy, are teachers able to identify bugs in Scratch programs?
- Are there differences in the ability to identify bugs between novice and more experienced teachers?
- What strategies do teachers plan on using to support students in identifying and fixing bugs?
- Are there differences in teacher support type between novice and more experienced CS teachers?

To answer these questions, we qualitatively analyzed the work teachers produced during our PD and their responses to open-ended prompts about their approaches to helping their students find bugs. Additionally, we used teachers' pre-knowledge assessments, years teaching CS and years teaching overall to identify similarities and differences in their debugging skills and their debugging pedagogical strategies based on prior experience. This work contributes to our understanding of teachers' abilities to help their students

learn how to debug and shed light on the role of prior experience in developing teachers' debugging pedagogical content knowledge.

2 PRIOR WORK

In this section, we review our theoretical framework and prior work on debugging. Our work builds upon research on novices versus experts in debugging and debugging instruction.

2.1 Theoretical Framework: PCK

In this work, we will frame our analysis and results using the Pedagogical Content Knowledge (PCK) framework [41]. Content knowledge refers to teachers' knowledge about the content they teach (e.g. knowledge of and ability to use specific productive debugging strategies). Pedagogical knowledge involves teachers knowledge about how to teach and support students in learning (e.g. Constructionism). Finally, pedagogical content knowledge refers to knowing the pedagogical practices in teaching the specific content (e.g. knowledge of how to teach others to understand how, when, and why to use particular debugging strategies).

Studying PCK in CS education is not new [3], but has seen a resurgence [20, 22, 48]. Hubbard calls for additional research on PCK in CS [20] and researchers recently used PCK to analyze teacher PD data [21, 46]. Yadav and Berges developed a survey to measure teachers' CS PCK through the use of vignettes [48]. They found differences in teachers' responses to the vignettes based on their level of overall and CS teaching experiences [48]. These researchers have contributed to the community's understanding of how to best help teachers learn and teach CS. Researchers have yet to use PCK to evaluate teacher debugging skills and pedagogical approaches.

2.2 Novices Vs. Experts

Early debugging research found that experts are more successful and faster than novices at finding bugs in a program [23]. Experts often pick a debugging strategy rather than switching frequently between strategies [45]; they were also able to debug the programs by viewing the system of the program as a whole [45] and the code in order of execution [23]. In contrast, novices tend to use erratic approaches to debugging [45], often read programs line-by-line [23], and seemed to focus on fixing the bug rather than taking the time to understand how the programs function [45].

Much of the recent research on debugging has focused on learning more about how novices debug code. Echoing findings from previous work, Fitzgerald et al. concluded that finding the bug is the most difficult part of debugging [14] and once students found the bug, they were often able to fix the problem [15]. Novices do not often have pre-existing skills that they can use to debug code [43] and use a variety of strategies to find bugs [38], including trial-and-error [25, 27, 49]. Alqadi and Maletic found that students who used trial-and-error were likely to introduce new bugs while they tried to fix existing ones [2]. Research has found that specific strategies were more helpful than others. Examples include encouraging novices to focus on understanding the program [14] and to make small to medium sized targeted changes to the code rather than making substantial changes to the code [24]. Researchers found there was an overlap in strategies and that some student seemed to use debugging strategies ineffectively [38]. Additionally, while many good novice debuggers were also good programmers, the opposite was often not true [15], which furthers the argument about the need to teach debugging skills.

Of the existing research on novices learning debugging, few researchers have focused on CS teachers. When debugging, novice teachers did not focus first on understanding the program [26]. Instead, they started by examining the output. Additionally, some teachers deleted segments of code rather than trying to fix them and those that did fix their code did not necessarily understand why the error occurred and why their changes fixed the code [26].

2.3 Debugging Instruction

Research on debugging instruction ranges from direct instruction to debugging games. Research on direct instruction has found a decrease in debugging time [10] and improvements on students' ability to find and correct bugs [33], even in K-6 (ages 5-12) [5].

Much of the work on supporting students while debugging focused on tools that were created to help with the debugging process [4, 11, 28, 36, 42]. These tools showed varying degrees of success in helping students with their debugging skills. To our knowledge, few researchers have explored ways for instructors to support their students while debugging. Wilson explored the use of Socratic approach where the instructor asks the student questions that encourages the student to change their perspective on the problem [47]. The goal is to guide the student towards understanding the problem and whether the proposed solution would work without explicitly explaining it to the student. Wilson also specified that the instructor should move from asking questions about the student's goal to how they believe their goal could be achieved (goal-oriented to procedure-oriented) [47]. Brusilovsky described a technique that involved providing students with various levels of assistance as needed [7]. The instructor starts with the least amount of assistance by showing the students the current buggy code and the results produced. Then, the instructor continues to give additional assistance, moving towards increasing levels of assistance until the student understands the error. This method has also been used to help middle school students learning Scratch [16].

Another area of debugging instruction involves exercises and activities. One activity involved having students work in teams to compete in finding bugs [8]. Another activity had students scaffolding in the form of worksheets to have students practice debugging strategies [9]. Students have also been tasked with finding and fixing bugs in the code of games in a game-design program [1]. Additionally, researchers have had students create projects with bugs for their peers to solve [13, 35]. These activities have shown varying levels of success in improve students' debugging skills and perspectives on software testing.

Some of the more recent work on debugging has introduced the idea of teaching debugging through games. Researchers developed Gidget [30, 31] and RoboBUG [34] for novice programmers. Including personified character who needs help with debugging code helped users solve more bugs in less time [30]. Additionally, including in-game assessments in Gidget increased engagement in users and led users to solve bugs more quickly [31]. An evaluation of RoboBUG revealed evidence of the game helping users learn

debugging techniques, fix buggy code, and was most helpful for those who scored lower on pre-test scores [34].

3 PROFESSIONAL DEVELOPMENT

We held a virtual, 8-week PD program (summer '20). Core CS concepts were taught through Scratch Encore, an intermediate Scratch curriculum [17] with Use→Modify→Create [18, 29] learning modules. Each module introduces the concept, progresses with a heavily scaffolded Use-Modify coding activity, and finishes with an open-ended Create project. Teachers completed one module each week, beginning with 30-minute synchronous introductory sessions that presented technical and pedagogical content. Teachers completed coding projects as asynchronous homework, and attended a one hour session for teachers to work together.

We introduced debugging using "WHAT?!? A MESS" during Module 3 (M3): Animation. Teachers were prompted to use it during the associated Modify activity. In addition, in modules 4 (Conditional Loops) and 5 (Decomposition), teachers completed scenario reflections (SRs), where they practiced debugging and imagined how they would help their students in debugging the programs.

"WHAT?!? A MESS" is a mnemonic instructional scaffold for students to bootstrap the debugging process; it was devised from patterns that emerged from analysis of the errors novice students made acting out Scratch scripts [19]. *Learners first identify the issue at the user level*: 1) **W**hat is the program trying to do?, 2) **H**ow did it go wrong?, 3) **A**nalyze what happened. *After step 3, learners then identify the bug's location in the code*: 3.1) **A**rguments, 3.2) **M**issing blocks, 3.3) **E**xtra blocks, 3.4) **S**crambled blocks, 3.5) **S**ubstituted blocks. Finally, if students are still stuck they are asked to use, 4) **T**hree before me (a common peer-first help-seeking strategy).

4 METHODS

We describe the debugging activities and details of our IRB-approved research study.

4.1 Debugging Activities

M3 Activity. For the M3 Modify task, teachers were given a buggy Scratch project and the following objective: "The programmer wanted both basketball players to dribble all the way across the basketball court." The teachers answered four multiple choice questions to identify which sprite(s) contained a bug, which event(s) had a bug, how the sprites' behaviors differed, and the code source of the bug (e.g., missing blocks). The teachers then fixed the program.

M4 and M5 Scenario Reflections. For each SR assignment, the teachers were given two scenarios, two buggy projects, the projects' goals, and pictures of the relevant code and the stage. The teachers were asked to identify the bug and discuss how they would help their student debug each program; both questions were open-ended. An example prompt is: "A student is very frustrated with his project. He thinks that he has done all of the coding correctly with a conditional loop and a wait until block, but when he triggers the project, the player sprite never stops, it just chases the cliff across the stage!" The accompanying code is displayed in Figure 1.

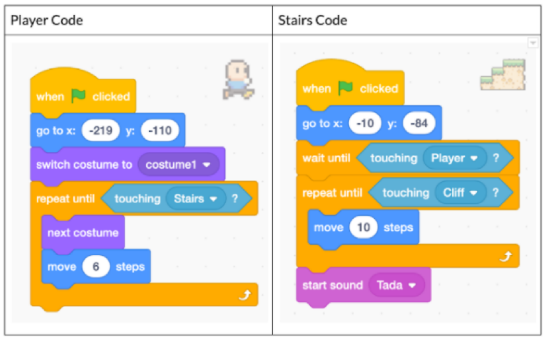

Figure 1: Scratch code provided for scenario reflection.

4.2 Participants and Data Collection

Forty-two teachers participated in our PD; 26 female, 14 male, 1 non-binary/third gender, and 1 preferred not to state. Twenty-nine participants were white, 7 were Black or African American, 2 were Hispanic or Latino(a), 2 were Asian, and 2 preferred not to state.

We collected several types of data throughout the PD: a pre-survey (including demographics and years teaching CS questions), pre- and post-assessments to measure CS content knowledge, M3 debugging worksheets, and the M4 and M5 SRs.

4.3 Data Analysis

M3 Worksheets. The debugging worksheets allowed the teachers to select multiple answers, therefore, answers could contain correct and incorrect selections. To assess the teachers in a way that accounted for all combinations of correct/incorrect answers, we adapted a grading rubric from [39]. Q1, Q2, and Q4 each had one correct answer; we graded the answers accordingly: **NO corr.**: *Circled none of the correct answers*, **BOTH corr. & incorr.**:, *Circled some corr. and some incorr. answers*, **ALL corr.**: *Circled all the correct and none of the incorrect answers.* For Q3, there were two correct options and **ONLY corr.**: *Circled correct but not incorrect answers* was added to the grading scheme. Two researchers graded every answer independently and achieved a κ of 0.98.

Scenario Reflections. Two researchers coded 40% of the data independently and compared the results. We coded the first part (identifying the bug) as correct or incorrect. The agreement for this question was 79-82%.

To assess the second part where the teachers describe themselves helping their students, we began by open-coding [40] and developed a three-dimensional coding scheme based on the open codes. *Focus* attends to where the teacher would try to guide the students towards the solution, problem, or process. *Level* identifies the level of help the teacher described: the program at an outcome (what the program should do) or code level (the code blocks themselves). *Action* explained what actions the teachers described (e.g. explaining concepts/solution, demonstrating concepts/solutions). Table 1 includes a full list of the codes and an example of each code. There is at least one code per dimension for each entry. For example, the example for the Problem code (*focus*) was coded as & Code, Outcome (*level*) & Ask, Guide (*action*). The agreement for the categories ranged from 68% to 100% agreement. The researchers

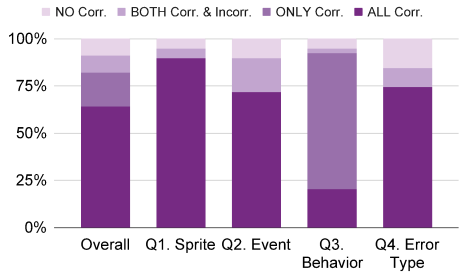

Figure 2: M3 debugging activity teacher responses.

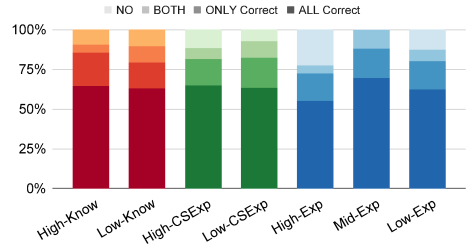

Figure 3: M3 activity results by teacher group.

coded the remaining 60% of the data independently. The codes for all submissions were resolved to complete agreement.

Teacher Knowledge Scores and Teaching Experience. To determine which teacher factors related their debugging skills and their strategies to help the students with debugging, we divided the teachers based on the average pre-assessment score (88.5%), average number of years (5.6 years) teaching CS, and number of years teaching (0-10, 11-20, 21-31). Nineteen and 17 teachers were in the high- and low-assessment score categories, respectively. Fourteen and 19 teachers were in the high- and low- CS teaching experience categories, respectively. Ten, 19, and 12 teachers were in the high-, mid-, and low- teaching experience categories, respectively.

We compared the teachers' strategies for each of the factors based on CS knowledge, CS teaching experience, and overall teaching experience using the Chi-square test of independence or its non-parametric alternative, Fisher's exact test. These tests compare the proportions of the values of the variables (e.g., number of teaches using each strategy for each CS knowledge level).

5 RESULTS

In this section, we first present the results of the debugging activities and compare novice and advanced teachers. Then, we present the teachers' approaches to help their students with debugging problems and how novice and advanced teachers' strategies differ. Our organization of findings is informed by our orienting theoretical framework in that we are first attending to Content Knowledge and then attending to Pedagogical Knowledge and PCK.

5.1 Content Knowledge: Identifying Bugs

Finding 1: Teachers successfully identified bugs in programs using "WHAT?!? A MESS". The majority of the M3 answers (Figure 2) were completely correct (64.10%) or only contained correct selections (17.95%). Answers that included correct and incorrect selections or no correct selections comprised about 18% of the data.

For the M4 and M5 reflections, 80% of the submissions correctly identified the bugs. Twenty-three correctly identified all four bugs (26 total completed both worksheets). Of the 14 teachers that completed one worksheet, 10 identified both bugs on the worksheet they completed. Teachers appeared to struggle more with the M5 bugs (accuracy 68%)than the M4 bugs (accuracy 92%). The results from the activities suggest that the teachers had the content knowledge that is necessary to debug programs.

Finding 2: Teachers were able to identify the sprite and event with the bug and at least one behavior and bug code location but had difficulty identifying all differences in the sprites' behaviors and where in the code the bug was located. Figure 2 shows the breakdown of the graded answers by question. Q1 had the most completely correct answers (89.74%) while Q3 had the fewest completely correct answers (20.51%). Q2 and Q4 have the most completely incorrect answers: 10.26% and 15.38%, respectively. This shows that the teachers had trouble identifying all the ways two sprites' behaviors differed (Q3) but most were able to identify at least one difference correctly. Teachers also had more completely incorrect answers for Q4.

Finding 3: Teachers' ability to identify bugs differed based on overall teaching experience. Figure 3 shows the teachers' M3 activity performance in the comparison groups. The two leftmost bars show the level of CS knowledge. The next two show the level of CS teaching experience. Finally, the rest show the level of overall teaching experience. From darkest to lightest, the bars represent the percentage of questions that were: completely correct; only correct; correct and incorrect; and completely incorrect.

That teachers performed similarly on the debugging activities regardless of prior CS knowledge (p=0.64, chi-square) and CS teaching experience (p=0.69, chi-square). There was, however, a statistically significant difference in performance based on overall teaching experience (p=0.002, fisher's). We completed post hoc pairwise comparisons to find the differences between the groups. The Bonferroni adjusted P value was p=0.0167. The difference between the high- and mid- experience teachers was statistically significant (p=0.0002) however, the differences between the high and low (p=0.7459), and mid and low, were not (p=0.02).

5.2 PCK: Scenario Reflections

Here, we focus on teachers' answers to how they would help a student who is having trouble finding bugs in a program. The goal is to explore the strategies teachers planned to use to help their students. We will present the results of our analysis in the categories that were introduced in Section 4: focus, level, and action.

Finding 4: Teacher responses tended to focus on the solution, was at the code level, and involved many types of actions.

5.2.1 Focus. The first aspect of the teachers' answers that we were interested in was the focus. We found that a majority, 61.36%, of responses given at least partially focused on the solution (Figure 4).

	Code	Description	Examples
Focus	Solution	Help focused on the solution or involves giving the student the solution	"I would ask her if she had tried other argument values/blocks."
	Problem	Help focused on the problem/bug	"I would ask him to think about where the car needs to be when the green flag is clicked and then ask him to look through his code and tell me what happens right after the green flag is clicked, hoping he would realize that he was missing the go to block."
	Process	Help focused on a debugging strategy or process	"Use the WHAT? debugging strategy and help him to see that adding a new event will help reset the car to the original location."
Level	Code	Feedback/help on the code/coding concepts or changes to the code	"I would show him that a go to block starts off just about every problem."
	Func.	Feedback/help on the outcome of the program	"Ask where should the car begin always? Where is the starting line? How can you get the car to always go back to the starting line?"
Action	Explain	Tell the student about problem, the solution, or the process	"I would tell the student that all sprites are actors and need to be told their starting postions when the green flag is clicked."
	Suggest	Suggest or ask the student to make specific changes or follow specific actions	"I would suggest that she clicks the color then the dropper and click on the gem to get the exact color."
	Ask	Ask leading question that identifies the bug concept or block	"I would ask the student where the code tells the car to start at the beginning. Since there is no code the student should notice to add it under the flag."
	Demo.	Show the student how to complete an action or show the student the solution.	"I would show him that a go to block starts off just about every problem"
	Guide	Walk through the actions without showing them. Student and teacher work together to find the problem/solution with back and forth interaction implied.	"How would help her go through the Analyze step and eventually help her to see she could use the substitute step and replace the sensing block."

Table 1: Three-dimensional Coding Scheme

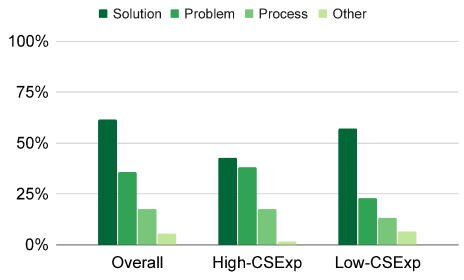

Figure 4: Teachers' Response Focus: CS Teaching Experience

Fewer answers involved the problem (35.61%) and process (17.42%). About 19% of the answers focused on multiple aspects.

Finding 5: There were differences in the teachers' focus based on CS teaching experience (Figure 4). There were no statistically significant differences based on CS knowledge (p=0.055) or overall teaching experience (p=0.59) in terms of the focus of the teachers' help. However, the teachers differed in this area based on CS teaching experience (p=0.0004). The teachers with less CS teaching experience focused on the solution more and on the problem less than the teachers with more CS teaching experience.

5.2.2 Level. The next aspect we investigated the level of help and whether their feedback direct students attention towards *code* or *outcome*. A majority of the answers (75%) discussed the code and 38.64% of the answers discussed the outcome.

Finding 6: There were no statistically significant differences based on CS knowledge (p=0.59), CS teaching experience (p=0.85), or overall teaching experience (p=0.91) in terms of the level of the teachers' responses.

5.2.3 Action. Finally, we explored the actions the teachers described. As shown in Figure 5, *ask* was the most common strategy the teachers wrote about (29.55%). *Suggesting* was the next most common (27.27%) followed by *guiding* (20.45%). The two least common strategies were *explaining* and *demoing* at 15.15% and 6.82%, respectively. Unfortunately, 26.52% of the answers were ambiguous, where we were unable to interpret how the teacher intended to help the student. These were often written as if the teacher was explaining how they would solve the problem themselves. For example, "Try using a different color or an actual character to have the cat stop at." Our results show that teachers vary in the way they approach helping students with debugging problems. Actions such as explain, suggest, ask, and guide are not necessarily CS or debugging specific where demonstration is likely to be closer to a CS pedagogical strategy. Further qualitative coding could reveal more information about which actions are PCK and which are PK.

Finding 7: There were differences in the teachers' actions based on pre-assessment scores. There were no statistically significant differences based on CS teaching experience (p=0.37) or overall teaching experience (p=0.48) in terms of the teachers' focus. However, teachers differed in this area based on CS knowledge (p<0.001) (Figure 5). The teachers in the low-know group had a greater percentage of answers with suggestions and demonstrations.

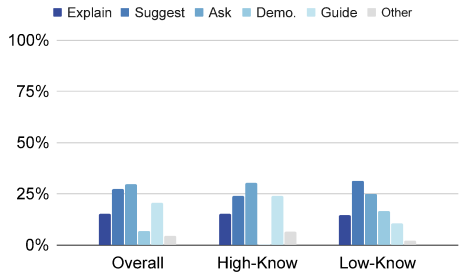

Figure 5: Teachers' Proposed Actions: CS Knowledge.

The teachers in the high-know. group had a greater percentage of answers with asking leading questions and guiding the students.

6 DISCUSSION

Our goals for this work were to evaluate teachers' debugging skills and their intended strategies for supporting their students' debugging learning process. Our analysis revealed that teachers showed promising debugging skills over the course of the PD and they proposed a variety of strategies to support their students. In this section, we review our findings and discuss the implications for future research and classroom practices.

Content Knowledge. Our work reveals that teachers show promising debugging skills. Teachers were able to identify what was wrong with a program, however, they struggled to locate the part of the code with the bug, which is the most difficult part of debugging and fixing programs [14, 15]. Teachers' actions in the reflections also differed based on prior CS knowledge. This suggests that the content knowledge affected the pedagogical strategies the teachers used, which is consistent with prior findings on the relationship between content knowledge and PCK [6].

Some of the most notable differences we found were with the teachers' use of demonstration and guiding in their responses to the scenario. High-know. teachers did not use demonstration, while low-know. teachers used demonstration in about 17% of their responses. High-know. teachers also used guiding strategies in 24% of their answers compared to low-know. teachers who used guiding in 11% of their responses. The guiding strategy is likely a more effective strategy since it involves the student and teacher collaborating with the teacher scaffolding the students' debugging process. Having the opportunity to reason through their thinking process is important for learners [37, 44]. While demonstration is also a valid pedagogical strategy, especially in CS, it likely needs to be paired with other strategies (e.g., explanation).

Pedagogical Knowledge and PCK. Surprisingly, we did not find differences in the strategies based on CS or overall teaching experience. This differs from prior work which found differences in teachers' responses to vignettes based on their CS and overall teaching experience [48]. Our results may differ because their vignettes were focused on student misconceptions while our scenarios were specific to debugging. Addressing student misconceptions likely requires a different set of skills than helping students find a problem in code and fix it. This points to the need to identify which pedagogical strategies are most effective for debugging.

Additionally, we found that teachers with more CS teaching experience had a more even number of responses that focused on solutions (43%) and problems (38%) than teachers with less CS teaching experience (solution: 57%, problem: 23%). Identifying the problem and solution is important in debugging. Since identifying the bug is more difficult than fixing the bug [14, 15], we hypothesize that the teachers who have taught CS longer have more experience and confidence in finding bugs, and therefore, are more comfortable with helping their students identify the bugs. This further supports the need to include debugging activities in training for CS teachers.

Supporting Novice CS Teachers. Our findings show that teachers did not differ in debugging skills based on prior CS knowledge and CS teaching experience. This differs from previous work where researchers found evidence of differences in debugging skills between novice and experts in CS [43]. This suggests that our activities and the "WHAT?!? A MESS" strategy are appropriate scaffolds for teachers new to CS and teaching CS, which is important since novices struggle with choosing and using debugging strategies [25, 27, 38]. We did however, find differences in debugging skills based on overall teaching experience.

Our findings echo previous findings on the connection between CK and PCK. In developing future materials for training novice CS teachers, it is important to include exercises where they learn how to debug and effective debugging pedagogical strategies.

Limitations. While this work provides insight into how teachers will potentially help their students debug in the classroom, we do not currently have data on the teachers' strategies when they are in classroom. Additionally, a portion of the teachers' answers for the scenario reflections were ambiguous in their action and were coded as other. Finally, as a qualitative study, there is always a potential of researcher bias. We worked together to resolve our differences through discussion in order to minimize those biases.

7 CONCLUSIONS

Teachers learning CS and how to teach CS need to simultaneously develop debugging skills (content knowledge) and skills to support their students in debugging (PCK). Without enough experience, teachers may struggle to do both in their classrooms. Our PD included explicit instruction for the teachers to learn and practice debugging and activities to reflect on how they may help their students complete debugging problems. We found that the teachers were able to identify the bugs in debugging activities regardless of their pre-assessment scores or number of years teaching CS. In their reflection activities, many teachers described multiple strategies for helping the students, ranging from giving explanations to guiding the students. Finally, we found that teachers differed in the focus of their support and their strategy based on prior CS teaching experience and knowledge. These results shed light on novice CS teachers' debugging content knowledge and PCK.

8 ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1738758. Thank you to Merijke Coenraad and Katie Sun for their help with this project.

REFERENCES

[1] Mete Akcaoglu. 2014. Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development* 62, 5 (2014), 583–600.

[2] Basma S Alqadi and Jonathan I Maletic. 2017. An empirical study of debugging patterns among novices programmers. In *Proceedings of the 2017 ACM Technical Symposium on Computer Science Education*. 15–20.

[3] Juliet Alice Baxter. 1987. *Teacher explanations in computer programming: A study of knowledge transformation.* Ph.D. Dissertation. Stanford University.

[4] Brett A Becker, Kyle Goslin, and Graham Glanville. 2018. The effects of enhanced compiler error messages on a syntax error debugging test. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 640–645.

[5] Marina Umaschi Bers, Louise Flannery, Elizabeth R Kazakoff, and Amanda Sullivan. 2014. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education* 72 (2014), 145–157.

[6] Sigrid Blömeke and Seán Delaney. 2014. Assessment of teacher knowledge across countries: A review of the state of research. *International perspectives on teacher knowledge, beliefs and opportunities to learn* (2014), 541–585.

[7] Peter Brusilovsky. 1993. Program visualization as a debugging tool for novices. In *INTERACT'93 and CHI'93 Conference Companion on Human Factors in Computing Systems*. 29–30.

[8] Renee Bryce. 2011. Bug Wars: a competitive exercise to find bugs in code. *Journal of Computing Sciences in Colleges* 27, 2 (2011), 43–50.

[9] Chiung-Fang Chiu and Hsing-Yi Huang. 2015. Guided debugging practices of game based programming for novice programmers. *International Journal of Information and Education Technology* 5, 5 (2015), 343.

[10] Ryan Chmiel and Michael C Loui. 2004. Debugging: from novice to expert. *Acm SIGCSE Bulletin* 36, 1 (2004), 17–21.

[11] Benjamin Cosman, Madeline Endres, Georgios Sakkas, Leon Medvinsky, Yao-Yuan Yang, Ranjit Jhala, Kamalika Chaudhuri, and Westley Weimer. 2020. PABLO: Helping Novices Debug Python Code Through Data-Driven Fault Localization. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1047–1053.

[12] M Ducasse and A-M Emde. 1988. A review of automated debugging systems: Knowledge, strategies and techniques. In *Proceedings.[1989] 11th International Conference on Software Engineering*. IEEE Computer Society, 162–163.

[13] Deborah A Fields, Yasmin B Kafai, Luis Morales-Navarro, and Justice T Walker. 2021. Debugging by design: A constructionist approach to high school students' crafting and coding of electronic textiles as failure artefacts. *British Journal of Educational Technology* (2021).

[14] Sue Fitzgerald, Gary Lewandowski, Renee McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education* 18, 2 (2008), 93–116.

[15] Sue Fitzgerald, Renée McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander. 2009. Debugging from the student perspective. *IEEE Transactions on Education* 53, 3 (2009), 390–396.

[16] Diana Franklin et al. 2013. Assessment of computer science learning in a scratch-based outreach program. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. 371–376.

[17] Diana Franklin et al. 2020. Scratch Encore: The design and pilot of a culturally-relevant intermediate Scratch curriculum. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 794–800.

[18] Diana Franklin, Merijke Coenraad, Jennifer Palmer, Donna Eatinger, Anna Zipp, Marco Anaya, Max White, Hoang Pham, Ozan Gökdemir, and David Weintrop. 2020. An Analysis of Use-Modify-Create Pedagogical Approach's Success in Balancing Structure and Student Agency. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 14–24.

[19] Diana Franklin, Jean Salac, Cathy Thomas, Zene Sekou, and Sue Krause. 2020. Eliciting Student Scratch Script Understandings via Scratch Charades. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 780–786.

[20] Aleata Hubbard. 2018. Pedagogical content knowledge in computing education: A review of the research literature. *Computer Science Education* 28, 2 (2018), 117–135.

[21] Aleata Hubbard and Katie D'Silva. 2018. Professional learning in the midst of teaching computer science. In *Proceedings of the 2018 ACM Conference on International computing education research*. 86–94.

[22] Peter Hubwieser, Marc Berges, Johannes Magenheim, Niclas Schaper, Kathrin Bröker, Melanie Margaritis, Sigrid Schubert, and Laura Ohrndorf. 2013. Pedagogical content knowledge for computer science in German teacher education curricula. In *Proceedings of the 8th workshop in primary and secondary computing education*. 95–103.

[23] Robin Jeffries. 1982. A comparison of the debugging behavior of expert and novice programmers. In *Proceedings of AERA annual meeting*.

[24] Chaima Jemmali, Erica Kleinman, Sara Bunian, Mia Victoria Almeda, Elizabeth Rowe, and Magy Seif El-Nasr. 2020. MAADS: Mixed-Methods Approach for the Analysis of Debugging Sequences of Beginner Programmers. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 86–92.

[25] Claudius M Kessler and John R Anderson. 1986. A model of novice debugging in LISP. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*. 198–212.

[26] ChanMin Kim, Jiangmei Yuan, Lucas Vasconcelos, Minyoung Shin, and Roger B Hill. 2018. Debugging during block-based programming. *Instructional Science* 46, 5 (2018), 767–787.

[27] Amy J Ko, Thomas D LaToza, Stephen Hull, Ellen A Ko, William Kwok, Jane Quichocho, Harshitha Akkaraju, and Rishin Pandit. 2019. Teaching explicit programming strategies to adolescents. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 469–475.

[28] Eric Larson and Rochelle Palting. 2013. Mdat: A multithreading debugging and testing tool. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. 403–408.

[29] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *Acm Inroads* 2, 1 (2011), 32–37.

[30] Michael J Lee and Amy J Ko. 2011. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*. 109–116.

[31] Michael J Lee, Amy J Ko, and Irwin Kwan. 2013. In-game assessments increase novice programmers' engagement and level completion speed. In *Proceedings of the ninth annual international ACM conference on International computing education research*. 153–160.

[32] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Computer Science Education* 18, 2 (2008), 67–92.

[33] Tilman Michaeli and Ralf Romeike. 2019. Improving Debugging Skills in the Classroom: The Effects of Teaching a Systematic Debugging Process. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*. 1–7.

[34] Michael A Miljanovic and Jeremy S Bradbury. 2017. Robobug: a serious game for learning debugging techniques. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education*. 93–100.

[35] Luis Morales-Navarro, Deborah A Fields, and Yasmin B Kafai. 2021. Growing Mindsets: Debugging by Design to Promote Students' Growth Mindset Practices in Computer Science Class. In *Proceedings of the 15th International Conference of the Learning Sciences-ICLS 2021*. International Society of the Learning Sciences.

[36] Christian Murphy, Eunhee Kim, Gail Kaiser, and Adam Cannon. 2008. Backstop: a tool for debugging runtime errors. *ACM SIGCSE Bulletin* 40, 1 (2008), 173–177.

[37] Laurie Murphy, Sue Fitzgerald, Brian Hanks, and Renee McCauley. 2010. Pair debugging: a transactive discourse analysis. In *Proceedings of the Sixth international workshop on Computing education research*. 51–58.

[38] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky–a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin* 40, 1 (2008), 163–167.

[39] Jean Salac, Max White, Ashley Wang, and Diana Franklin. 2019. An Analysis through an Equity Lens of the Implementation of Computer Science in K-8 Classrooms in a Large, Urban School District. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1150–1156.

[40] Johnny Saldaña. 2021. *The coding manual for qualitative researchers.* SAGE Publications Limited.

[41] Lee S Shulman. 1986. Those who understand: Knowledge growth in teaching. *Educational researcher* 15, 2 (1986), 4–14.

[42] Natalia Silvis-Cividjian, Marc Went, Robert Jansma, Viktor Bonev, and Emil Apostolov. 2021. Good Bug Hunting: Inspiring and Motivating Software Testing Novices. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 171–177.

[43] Beth Simon, Dennis Bouvier, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Kate Sanders. 2008. Common sense computing (episode 4): Debugging. *Computer Science Education* 18, 2 (2008), 117–133.

[44] Stephanie D Teasley. 1997. Talking about reasoning: How important is the peer in peer collaboration? In *Discourse, tools and reasoning*. Springer, 361–384.

[45] Iris Vessey. 1985. Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies* 23, 5 (1985), 459–494.

[46] Rebecca Vivian and Katrina Falkner. 2019. Identifying Teachers' Technological Pedagogical Content Knowledge for Computer Science in the Primary Years. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 147–155.

[47] Judith D Wilson. 1987. A Socratic approach to helping novice programmers debug programs. *ACM SIGCSE Bulletin* 19, 1 (1987), 179–182.

[48] Aman Yadav and Marc Berges. 2019. Computer science pedagogical content knowledge: Characterizing teacher performance. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–24.

[49] Ching-Zon Yen, Ping-Huang Wu, and Ching-Fang Lin. 2012. Analysis of experts' and novices' thinking process in program debugging. In *International Conference on ICT in Teaching and Learning*. Springer, 122–134.