Exploring the Impact of Cognitive Awareness Scaffolding for Debugging in an Introductory Computer Science Class

Jiwon Lee
jlee671@calpoly.edu
California Polytechnic State University
San Luis Obispo, California, USA

Ayaan M. Kazerouni
ayaank@calpoly.edu
California Polytechnic State University
San Luis Obispo, California, USA

Christopher Siu
cesiu@calpoly.edu
California Polytechnic State University
San Luis Obispo, California, USA

Theresa Migler
tmigler@calpoly.edu
California Polytechnic State University
San Luis Obispo, California, USA

ABSTRACT

Debugging involves the simultaneous application of a number of programming skills—reading code, writing code, problem comprehension, etc. This makes it a challenging activity for novice programmers. Unfortunately, debugging is rarely taught explicitly in introductory programming courses, and is often learned as an implicit goal through programming assignments. In this experience report we explore the impact of a cognitive awareness scaffold to help students monitor their progress as they debug their code. We created a simple form that students used to document their debugging process when they ran into bugs. The form asks questions that students are likely to be asked by course staff during office hours, e.g., "What have you tried so far?". This act of verbalizing errors and enumerating successful and unsuccessful strategies to fix them is meant to help students monitor their own debugging progress. We examined the cognitive awareness demonstrated in form responses, finding that responses were more superficial on projects of higher difficulty. Additionally, we gave students an exit survey to measure the perceived impact of the debugging form on students' ability to regulate their debugging process and their confidence while debugging. Students indicated that the form helped them better verbalize errors in their programs, and helped them surmount problems with which they would otherwise have needed help.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

computing education, debugging, cognitive awareness scaffolding

SIGCSE 2023, March 15–18, 2023, Toronto, ON, Canada

ACM ISBN 978-1-4503-9431-4/23/03…$15.00
https://doi.org/10.1145/3545945.3569871

ACM Reference Format:
Jiwon Lee, Ayaan M. Kazerouni, Christopher Siu, and Theresa Migler. 2023. Exploring the Impact of Cognitive Awareness Scaffolding for Debugging in an Introductory Computer Science Class. In *Proceedings of the 54th ACM Technical Symposium on Computing Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569871

1 INTRODUCTION

Debugging is an important skill that many novices find quite difficult. During a debugging task, a novice must have a fair amount of metacognitive awareness to help monitor their progress and keep track of strategies that have been successful and unsuccessful. There is evidence that suggests that scaffolding for metacognitive awareness helps with problem-solving. We report on an experience with cognitive awareness scaffold that we gave to students in an introductory programming course to help them monitor their debugging processes. We designed a form that students could use to document their debugging processes as they worked through course projects. This debugging form focuses on helping students to recognize the initial steps of debugging and to organize their thoughts. It was designed based on research about metacognitive awareness and the authors' own experiences as instructors to effectively guide students in debugging.

When students reach out to course staff for help fixing bugs in their code, they are often asked questions like "What do you understand about this bug so far?" and "What have you tried so far?" Questions like these induce students to take a step back from their ongoing approach and to consider the problem anew. The act of verbalizing the difference between what the program *should* do and what the program *does* do often leads to an "aha!" moment for students, following which they are able to locate and remove bugs with minimal guidance. Indeed, this is is the idea behind so-called "rubber-duck debugging" [5]. To facilitate this process, we propose a simple documentation method to help students maintain cognitive awareness as they approach a debugging task.

We gave students a form to use for this process. It was used by students in a 10 week-long introductory computer science class at a medium-sized primarily undergraduate public university in the USA. Students filled out the form for "major bugs" (defined as such by themselves) for three programming assignments in the class. At the end of the term, we deployed an exit survey to learn about the

perceived impact of the form. Additionally, we explored the level of metacognitive awareness demonstrated on the form responses and their relationships with project performance.

Students reported that the form helped them to verbalize their errors, and that using the form helped them fix bugs with which they would otherwise have needed help from course staff. While we observed the trend that students tended to display higher levels of metacognitive awareness on projects with higher average scores, this relationship did not hold up to statistical analysis.

2 BACKGROUND

Debugging. Debugging is the process of locating and fixing defects in a software project after they have been revealed. It is a natural and important part of programming.

Debugging can generally be broken down into four steps: Understand, diagnose, locate, and correct [8, 13]. Once a programmer realizes the program did not execute as they intended, they know a defect exists. Then they might start to build and check their hypotheses about what exactly is going wrong. After the diagnosis, the programmer tries to locate the bug and fix it.

Debugging is a challenging process for both experts and novices (e.g., [1, 10]). Programmers may go through the process above several times, during which they may run into a number of roadblocks. For example, a programmer might be unable to locate a bug after identifying that a defect exists. Or, they may successfully locate the bug but not know how to fix it. They may even introduce new bugs while investigating existing defects.

Novices in particular face a number of challenges while debugging. One major hurdle they must overcome is the discrepancy between natural language communication and the reading and writing of programs [16]. For example, Pea *et al.* describe novices expecting computers to somehow infer their intent from incomplete or incorrect code, leading to difficulties locating bugs in their code. Katz & Anderson observed that it is clear that debugging is not a single activity, but a set of activities where each component may be performed differently depending on the situation [8]. Murphy *et al.* portrayed novice debuggers as new drivers who "must learn to steer, accelerate, brake, etc. all at once" and observed that "novice debuggers must apply many new skills simultaneously" [14]. For example, several studies suggest that the difference between novice and expert programmers in debugging is due to differences in problem comprehension ability [4, 15, 20]. Gugerty & Olson observed that experts were faster and more successful at finding bugs in simple programs compared to novice subjects [4]. They suggest that "the primary reason for the experts' superiority was the ease with which they understood what the program does and is supposed to do" which allowed them to quickly isolate the bug. This suggests that both program and problem comprehension are important preconditions for successful debugging.

A plethora of tooling exists to aid in the debugging process. Layman *et al.* noted that 15 professional software engineers mentioned 16 distinct debugging tools varying according to their application domain. Murphy *et al.* [14] and Mansur *et al.* [12] note that students employ a number of tools and strategies to locate bugs, e.g., simple print statements, the in-IDE debugger, unit tests, or even manual program tracing. Both sets of authors note that some strategies were used ineffectively or inconsistently.

Metacognition. *Metacognition* is generally understood to be the knowledge one has about one's own cognition [3]. When a problem-solver is metacognitive, they are aware of strategies for accomplishing their task, they can monitor their progress, and can evaluate the effectiveness of different available strategies [17].

Metacognition is important in programming and programming education. For example, a number of papers (e.g., [18, 21]) have observed that novice programming students often incorrectly interpret the problem prompt, leading to them solving the wrong problem entirely without realizing it. After interventions in which students were asked to explicitly demonstrate their understanding of the problem before starting to write code (e.g., by creating test cases), researchers noted improved metacognitive skills and better understanding of problem prompts [18].

In another study, Loksa *et al.* proposed a new approach to explicitly teach problem-solving skills consisting of 1) explicit instruction on programming problem solving, 2) a method of visualizing and monitoring progress through six problem-solving stages, 3) explicit prompts for learners to reflect on their strategies when seeking help from instructors, and 4) context-sensitive help embedded in a code editor [11]. They found that students who received the explicit instruction were more cognitively aware than students who did not receive the instruction; they were better able to articulate solution strategies. Moreover, they were also more likely to attempt solution implementations before asking for help.

Guidance about explicit programming strategies is not only helpful to novices, but also to intermediate- and expert-level programmers. LaToza *et al.* found that when programmers were given explicit strategies to follow, they found their work to be more organized and systematic, and were more successful at completing design and debugging tasks [9].

These studies demonstrate that promoting metacognitive awareness, be it about checking one's own understanding of a problem prompt, or monitoring one's own progress through problem-solving, has a positive impact on one's ability to solve programming problems. So too for debugging: Murphy *et al.* suggest that such scaffolds ought to be incorporated into debugging instruction [14].

This paper reports on an experience with providing students with a cognitive awareness scaffold to be used during the debugging process. Students often seek help from various sources, including instructors in office hours, when they discover bugs in their programs [2]. Following Ren *et al.* [19], we designed a form to be filled by students when they run into bugs in their software, in which they are asked to self-assess their progress toward locating and fixing the bug. We studied the cognitive awareness demonstrated in these entries and their relationship with eventual project outcomes.

3 STUDY CONTEXT

Our study focused on students in a CS 2-level data structures course at Cal Poly, a medium-sized primarily undergraduate public university in the USA. Data was collected during the Spring (March–June) quarter of 2022.

This CS 2 course is a suitable test-bed in which to study the students' debugging awareness. Students in the course have previously completed a CS 0 and CS 1 course or equivalent material prior to entering our university. That is, they have had roughly two quarters of programming experience. This allows us to focus on issues specific to debugging programs, rather than elementary difficulties with reading and writing programs.

The five assignments in the course were completed using Python. Project 1 was excluded because its specification was published before we introduced our intervention. Project 5 was excluded because its deadline did not meet our timeline for analysis. Our intervention was therefore applied during the middle three projects (Projects 2, 3, and 4) in the quarter.

Projects generally involved implementing or applying data structures. Project 2 was based on stacks and the Shunting Yard algorithm; Project 3 was based on binary trees and Huffman coding; Project 4 was based on hash tables and text indexing.

Students were given roughly a week to work on each project, during which time they received limited feedback from an autograder. They received autograder feedback roughly four times per project, based on a suite of reference tests written by the instructors. If multiple errors existed, students were only given detailed feedback about the first one. If a student's submission did not include unit tests of their own with line coverage, they did not receive any detailed feedback at all. This helped avoid students' relying on the autograder for their own testing.

4 DEBUGGING FORM

We designed the debugging form to act as a conversation between the students and instructor. Murphy *et al.* suggested that "debugging instruction should incorporate these metacognitive factors, perhaps taking the form of self-questions". For example, they suggested "What else could I try?", "Is this too much to keep track of in my head?", and "What are other possible sources of the bug?". We tried to design the debugging form in a way that students would ask themselves such questions while filling out the form.

The form contained five fields: Issue Category, Issue Description, Previous Attempts, Solved by Myself, and Asked Question.

Issue Category The first part of debugging form is identifying which debugging stage students are stuck at.

Below are the categories of debugging stages we provided on the debugging form.

- Understanding an error
- Locating an error
- Testing an error
- Fixing an error
- Auto-grader Feedback
- Other

Students' project code submissions are graded by an auto-grader where students receive feedback based on the run. Since the auto-grader error messages are one of the resources students utilize in the course setting for debugging, we included this as one of the categories, "Auto-grader Feedback".

Issue Description This section is one of the most important sections of the debugging form. Students were instructed to explain the error in their own words so that they could reflect on their thoughts. Unlike simply copying the stack trace error message, describing it in their own words promotes metacognitive awareness. In essence, this section asks students to reflect on their thought processes on each debugging stage such as "What does this error mean?" or "Where is the error happening?".

Previous Attempts This section is another important section of the form. Students are asked to document what they have tried to solve the issue they identified in the previous sections "Issue Category" and "Issue Description". Students could list resources they utilized or simply write down what modifications they made in their code. We wanted this section to ask students "What have I tried?" and "What else could I try?".

Solved by Myself: Here students state whether they solved the bug by themselves or not.

Asked Question Here students state whether they asked the instructor about the particular bug or not.

5 METHODOLOGY

The debugging form was introduced to students in four sections of the CS 2 course described in Section 3, taught by two instructors. One instructor taught one section, and the other, three sections. All sections included identical content, assignments, and exams. The course involved relatively small lab assignments which were often completed in class during lab time. The course also involved projects which were larger and more complex than lab assignments, and were worked on individually on outside class.

Students voluntarily filled out our debugging forms and submitted them as a part of their project submissions. The submitted forms were not graded and their contents did not affect students' project scores. For each project students were offered 3% extra credit for submitting a form, but only if their form had at least two entries.

5.1 Form Instructions

To help students familiarize themselves with the idea of the debugging form, we suggested that it was a written version of what they would have explained to their instructors when asking questions.

We asked students to fill out the form during the debugging process because we believe that support should be provided while they are debugging to help with their thought processes. Kapa also noted that learning environments which supply metacognitive support during the process of problem-solving are significantly more effective than those that provide the same support at the end of the process or those that do not provide any support [7].

To be as unintrusive as possible, students were also instructed to fill out the debugging form when fixing "major" bugs. We let them define what "major" means, since we did not want to put too much restriction on what they should be logging.

Additionally, students were required to present specific entries from their debugging forms whenever they asked an instructor about a bug or asked an instructor to look at their code or error messages. Having students present their debugging forms also allowed us to observe that students were actually filling them out. This emulates one of the interventions designed by Loksa *et al.*, who asked students requesting help to describe their problem, their attempted solutions, and their current problem-solving stage [11]. We did not require students to present forms when asking questions

on the class forum, since formulating a post would (ideally) involve some self-reflection about what one has already tried.

We note that if a student has to ask their instructor about a bug, that must be a "major" bug, since seeking help from instructors tends to be a last resort for many students [2]. However, sometimes students simply do not know where to begin tracking down an error. We did not want to discourage students from asking questions due to the pressure of having to write something down beforehand. So we allowed for the "previous attempts" field on the form to indicate that the student did not know where to start. This non-committal response was not abused: there were only 3 such entries for Project 2, 2 for Project 3, and 0 for Project 4.

5.2 Exit Survey

We created an exit survey to understand how helpful the students found the debugging form. Question types included Likert type questions, Yes/No/Maybe questions, and one free-response question. In our survey, Likert scale response consists of a 5-point scale from 1-Extremely Disagree to 5-Extremely Agree. Survey questions and results are summarized in Table 1.

We used the survey to understand whether the debugging form helped students "be metacognitive" about the debugging process. Specifically, we would like to know if the form helped students to understand and verbalize their bugs and errors (Q2–Q3), and helped them monitor their progress toward addressing a bug or error (Q4). Anecdotally, we noticed several "aha" moments while helping students, where they themselves identify an approach they could take to fix the bug. In Q5, we would like to know if the process of filling out the form led to any of these moments, precluding the need to seek help from course staff. Although self-efficacy is out of the scope of this study, we wanted to know if filling out the form had an impact on students' confidence in approaching debugging (Q6). Finally, the survey gave students an opportunity to provide anonymous free-response feedback, which we discuss in Section 6.

6 SURVEY ANALYSIS

A total of 38 students (out of 140) filled out the exit survey. (Unlike the debugging forms, students were not incentivized to complete the exit survey at the end of the term.) Of these, 2 did not give us consent to use their responses, and 3 mentioned that they never used the debugging form in spite of taking the survey. After filtering out these responses, we ended up with 33 responses for our analysis. Of these 33, 2 did not consent to the use of their project grades in the analysis—their responses are not included in any analysis involving project grades.

The survey contained 4 Likert-type questions, with responses ranging from *Extremely disagree* to *Extremely agree*.

6.1 Survey Results

We present summary statistics for responses to each question.

Q1: Have you used this type of documentation method in the past? The method of documenting ones debugging process was unknown to most participants. Out of 33 responses, 4 participants responded *Yes* (12%) and 29 participants responded *No* (88%).

Q2: The debugging form helped me with verbalizing errors. The median student *Somewhat agreed* that the debugging form helped them to verbalize errors in their programs, suggesting that they seemed to find the form activity helpful.

Q3: The debugging form helped me with explaining the debugging progress to instructors We expected the distribution of responses to be similar to Question 2, because we expected that verbalization ability would be correlated with explanation ability. However, students were *Neutral* about how much the debugging form helped them explain the debugging process to instructors.

Q4: The debugging form helped me stay aware of my debugging progress. This question aimed to measure participants' knowledge of their debugging process, i.e., if participants were aware of the debugging stage they were in and the problem they were trying to solve. Again, the median participant was *Neutral* in their response to this question.

Q5: I have solved at least one bug that I was going to ask instructors about while filling out the debugging form.

Question 5 asked participants if they have solved at least one bug that they were going to ask instructors about while filling out the debugging form. Out of 33 responses, 24 participants responded Yes (73%) and 9 participants responded No (27%). This suggests that the debugging form may have promoted independence in participants' debugging processes.

Q6: I feel more confident in approaching debugging process because I learned and used the debugging forms. The median student was *Neutral* in their response to this question, suggesting that students did not feel that using the form made them feel more confident about approaching a bug in their program.

Q7: Do you see yourself utilizing this type of method in future computer science classes? Out of 33 responses, 9 participants responded Yes, 7 participants responded No, and 17 participants responded Maybe.

Q8 (Optional): Please feel free to share any comments in this section. Any positive/negative/neutral experience with debugging form?

From the optional free response question, we identified two confusions participants might have experienced.

First, we learned that some participants used the debugging form after they had completely solved the bug.

> The forms would be extremely helpful had I used them during the debugging process. However, I used the forms after the fact. –Participant 37

> Most of the time when I was using the debugging form, I had already solved the problem ages ago and was filling the form out afterward. –Participant 14

More monitoring and guidance about how and when to use the forms might have been warranted. For example, Participant 26 mentioned that using the form "broke up [their] thinking process by making [them] fill out the form" during debugging.

Second, there was a misunderstanding related to the "Issue Category" field on the form. The provided options for "Issue Category" were steps of debugging we identified through a review of the literature. Our intention was for participants to identify what stage they are at with solving the bug, for example "I am trying to understand the error" or "I understand the error but do not know how to fix it". However, we received comments that imply the "Issue Category"

Table 1: Survey questions and results. For the Likert-type questions, the median response is provided.

	Question	Results
1	Have you used this type of documentation method in the past?	12% *Yes*, 88% *No*
2	The debugging form helped me with verbalizing the errors.	*Somewhat agree*
3	The debugging form helped me with explaining the debugging progress to instructors.	*Neutral*
4	The debugging form helped me stay aware of my debugging progress.	*Neutral*
5	I have solved at least one bug that I was going to ask instructors about while filling out the debugging form.	73% *Yes*, 27% *No*
6	I feel more confident in approaching debugging process because I learned and used the debugging forms.	*Neutral*
7	Do you see yourself utilizing this type of method in future computer science classes?	27% *Yes*, 51% *Maybe*, 21% *No*
8	Please feel free to share any comments in this section. Any positive/negative/neutral experience with debugging form?	See Section 6

was expected to provide specific *bug types* as options, as opposed to the *stage of debugging*. However, identifying the debugging step and describing the bug in detail are connected yet separate tasks in the debugging form. Details about the specific bug at hand would be better placed in the next column, "Issue Description". Commenting on the "Issue Category" column, Participant 6 wrote: "The Other category was sometimes intimidating because I didn't even know how to describe the error and if there were more options maybe I would have been able to articulate my issue better". And Participant 31 said "More options for stuff like that in the 1-6 category other than Other might help articulate exactly what's wrong".

These responses provide invaluable feedback for future iterations of the form. For example, providing options based on a taxonomy of error types students are likely to run into on these assignments could help students articulate their bugs a bit better.

7 FORM ANALYSIS

Implementing a method to measure one's metacognitive awareness is not straightforward. Jacobse and Harskamp suggested that the development of measurement instruments be specifically shaped to fit certain domains due to the fairly domain-specific nature of metacognition [6]. We wanted to perform metacognitive awareness measurement analysis on how much *we* think students understood the bugs they logged on the debugging form. However, the existing methods to accomplish this goal was inaccessible or the debugging form was specific to our study context. Therefore, we created our own rubric to measure students' understanding of their bugs.

7.1 Form Analysis Rubric

The rubric has three levels where each entry has to meet specific criteria to qualify for one of the levels. Each assignment's level was assigned by the average of the levels (rounded it up if not not uniform). We chose to round up because the existence of higher level shows that they are capable of understanding the bug at that level. To qualify for one level, the entry has to meet the level's criteria on top of the previous levels'. Table 2 describes the rubric and provides examples responses for each level. The examples are extracted from submitted debugging forms. To avoid bias based on performance, project grades were not considered before submitted forms were assigned levels according to the rubric.

7.2 Results

Figure 1 presents frequencies of metacognitive awareness scores for each project, measured using the rubric described earlier. The total number of the forms for each project is different because 10 students did not submit the debugging form for every project.

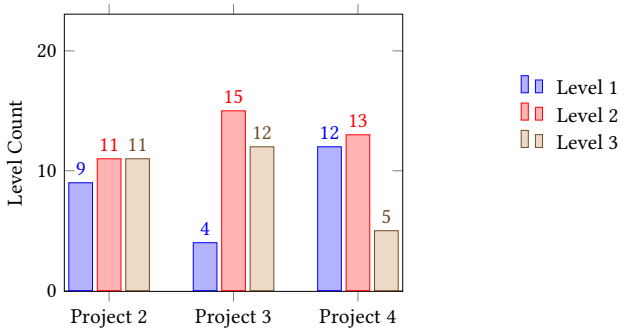

Figure 1: Distribution of form level placements for students on the three projects.

We did not necessarily expect students' level placement over time would improve because we did not provide a feedback on their debugging forms. We hypothesize that if students were given feedback on each debugging form, their level placement might have increased over time. However, we did expect that the level placement over time to stay at least consistent. Surprisingly, we observed that the students' understanding of the bugs decreased from project 3 to project 4 based on our rubric. We hypothesized that this trend might be correlated with project difficulty and decided to look at average grade of each project.

The average grade for project 2, 3, and 4 were 92.39%, 98.26%, and 92.71% respectively (see Figure 2). Visually, average grades and students' level placements appeared to increase from Project 2 to 3 but decreased from 3 to 4, suggesting a correlation between project

Table 2: Rubric for classifying form responses and (real) example responses for each level.

Level	Description	Example
1	Mentions basic information about the error.	"Wrong output"
2	Describe the faulty behavior of program at high level language.	"For the concordance functions, it would to print out the entire line over and over instead of each word."
3	Describe the faulty behavior of program at low level language and/or shows evidence of previous debugging stage hypothesis.	"In postfix_eval: Getting ValueError: could not convert string to float even though I have a try/except to handle this case. It also says During handling of the above exception, another exception occurred, followed by a KeyError: 'blah'."

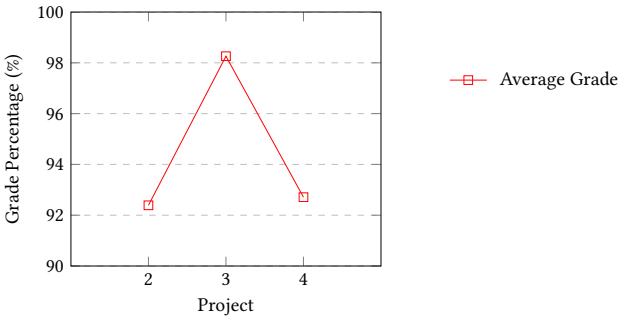

Figure 2: Average grade on each project

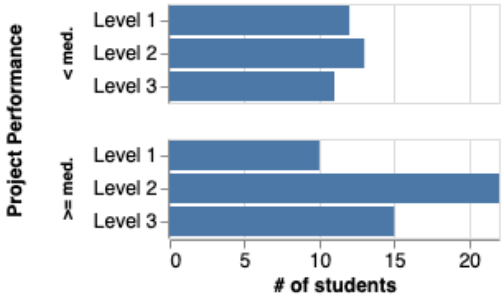

Figure 3: Students who scored above the the median were more likely to display Level 2 or 3 on their debugging forms.

performance and students' understanding of their debugging process. We tested this hypothesis with a Kruskal-Wallis analysis of variance, with awareness level as a categorical independent variable, and project score as a numerical dependent variable. However, we were unable to confirm a relationship between the two variables ($H = 2.56, p = 0.28$).

That said, project distributions were highly skewed, with a median score of 98%. Considering cognitive awareness scores for project performances below and above this threshold (50% each), it is visually apparent in Figure 3 that students who scored higher on metacognitive awareness also scored higher on the project.

This is worth exploring further, perhaps with an improved rubric to measure cognitive awareness (discussed in Section 8).

8 DISCUSSION

It appeared that a higher level of cognitive awareness on the debugging forms was accompanied by better performance on the projects. Specifically, grades on Project 3 were better than those on Projects 2 and 4. Accordingly, form submissions on Project 3 showed higher cognitive awareness scores than those on Project 2 and 4. We cannot say anything about the direction of causality project scores and cognitive awareness placements, but prior research suggests that students with higher cognitive awareness tend to be better at solving programming problems.

Based on the exit surveys (Table 1), students found that filling out the form helped them to verbalize errors, and may have helped them surmount problems that would otherwise have driven them to seek help in office hours. While our goal is not to discourage help-seeking, this added self-sufficiency is a positive.

Students were asked to report on "major" bugs in their projects, and were given the freedom to define "major" bugs. While this gave them fewer restrictions on what bugs they did or did not log, a narrower definition (e.g., based on types of errors, test cases failed, or time taken to fix them) would have standardized the kinds of bugs that students reported. However, the focus of this study was on students' awareness of their debugging processes, and not on the bugs themselves. We valued fewer restrictions over standardization.

We see a number of possible directions for future work. First, we plan to address some confusions that arose regarding the debugging form (see Section 6.1) by re-phrasing portions of the form headings. Additionally, we believe confusion regarding the instructions how to fill out the debugging form could be solved with a explicit lecture explaining the debugging stages, similar to the way Loksa *et al.* gave explicit instruction on programming problem solving. While we made the resources available for students to review, our brief introduction to the debugging form and debugging stages might not have been sufficient.

Next, students were asked to bring their forms with them to instructor office hours. It would be interesting to examine the relationship between aspects of the help session and the cognitive awareness demonstrated on the form. For example, how articulate were students in formulating their help requests after having used the form? Students self-reported *Neutral* to this question in the survey (Table 1), but the course staff's perspective would provide an interesting point of comparison.

Our rubric described in Section 7.1 is a potential source of threat to construct validity, particularly given the confusions described in

Section 6.1. In the future, we plan to give students explicit feedback based on their debugging form submissions, and to explore the impact of this feedback on their debugging process and effectiveness.

REFERENCES

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An Analysis of Patterns of Debugging among Novice Computer Science Students. *SIGCSE Bull.* 37, 3 (jun 2005), 84–88. https://doi.org/10.1145/1151954.1067472

[2] Augie Doebling and Ayaan M. Kazerouni. 2021. Patterns of Academic Help-Seeking in Undergraduate Computing Students. In *21st Koli Calling International Conference on Computing Education Research* (Joensuu, Finland) *(Koli Calling '21)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/3488042.3488052

[3] J H Flavell. 1976. Metacognitive Aspects of Problem Solving. In *The Nature of Intelligence*, L B Resnick (Ed.). Earlbaum, Hillsdale, NJ, 231–235.

[4] L. Gugerty and G. Olson. 1986. Debugging by Skilled and Novice Programmers. *SIGCHI Bull.* 17, 4 (apr 1986), 171–174. https://doi.org/10.1145/22339.22367

[5] Andrew Hunt. 1990. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley.

[6] Annemieke E. Jacobse and Egbert G. Harskamp. 2012. Towards efficient measurement of metacognition in mathematical problem solving. *Metacognition and Learning* 7, 2 (Aug 2012), 133–149. https://doi.org/10.1007/s11409-012-9088-x

[7] Esther Kapa. 2001. A Metacognitive Support during the Process of Problem Solving in a Computerized Environment. *Educational Studies in Mathematics* 47, 3 (Sep 2001), 317–336. https://doi.org/10.1023/A:1015124013119

[8] Irvin R Katz and John R Anderson. 1987. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction* 3, 4 (1987), 351–399. https://www.researchgate.net/publication/234780775_Debugging_An_Analysis_of_Bug-Location_Strategies

[9] Thomas D. LaToza, Maryam Arab, Dastyni Loksa, and Amy J. Ko. 2020. Explicit programming strategies. *Empirical Software Engineering* 25, 4 (July 2020), 2416–2449. https://doi.org/10.1007/s10664-020-09810-1

[10] Lucas Layman, Madeline Diep, Meiyappan Nagappan, Janice Singer, Robert Deline, and Gina Venolia. 2013. Debugging Revisited: Toward Understanding the Debugging Needs of Contemporary Software Developers. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 383–392. https://doi.org/10.1109/ESEM.2013.43

[11] Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 1449–1461. https://doi.org/10.1145/2858036.2858252

[12] Rifat Sabbir Mansur, Ayaan M. Kazerouni, Stephen H. Edwards, and Clifford A. Shaffer. 2020. Exploring the Bug Investigation Techniques of Intermediate Student Programmers. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '20)*. Association for Computing Machinery, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/3428029.3428040

[13] Renée 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 (06 2008). https://doi.org/10.1080/08993400802114581

[14] 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. 40, 1 (mar 2008), 163–167. https://doi.org/10.1145/1352322.1352191

[15] Murthi Nanja and Curtis R Cook. 1987. An analysis of the on-line debugging process. In *Empirical studies of programmers: Second workshop*. Norwood, NJ: Ablex, 172–184. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.935.3262&rep=rep1&type=pdf

[16] Roy D Pea, Elliot Soloway, and Jim C Spohrer. 1987. The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics* 9 (1987). https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.487.879&rep=rep1&type=pdf

[17] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 2–13. https://doi.org/10.1145/3372782.3406263

[18] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 531–537. https://doi.org/10.1145/3287324.3287374

[19] Yanyan Ren, Shriram Krishnamurthi, and Kathi Fisler. 2019. What Help Do Students Seek in TA Office Hours?. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 41–49. https://doi.org/10.1145/3291279.3339418

[20] Iris Vessey. 1986. Expertise in Debugging Computer Programs: An Analysis of the Content of Verbal Protocols. *IEEE Transactions on Systems, Man, and Cybernetics* 16, 5 (1986), 621–637. https://doi.org/10.1109/TSMC.1986.289308

[21] John Wrenn and Shriram Krishnamurthi. 2019. Executable Examples for Programming Problem Comprehension. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 131–139. https://doi.org/10.1145/3291279.3339416