EXPLORING THE IMPACT OF COGNITIVE AWARENESS SCAFFOLDING FOR DEBUGGING IN AN INTRODUCTORY COMPUTER SCIENCE CLASS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jiwon Lee

June 2022

COMMITTEE MEMBERSHIP

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

AUTHOR: Jiwon Lee

DATE SUBMITTED: June 2022

COMMITTEE CHAIR: Theresa Migler, Ph.D.
Assistant Professor of Computer Science

COMMITTEE MEMBER: Ayaan Kazerouni, Ph.D.
Assistant Professor of Computer Science

COMMITTEE MEMBER: Chris Siu
Lecturer of Computer Science

ABSTRACT

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

Jiwon Lee

Debugging is a significant part of programming. However, a lot of introductory programming classes tend to focus on writing and reading code than on debugging. They utilize programming assignments that are designed in ways such that students learn debugging by completing these assignments which makes debugging more of an implicit goal. In this thesis, we propose a cognitive awareness scaffolding in debugging to help students self-regulate their debugging process. We validate its effectiveness by conducting experiments with students in four sections of a Data Structures course, which is one of the introductory computer science classes at California Polytechnic State University, San Luis Obispo. In this form, students identified the debugging stage, described the bugs in their own words, and tracked their attempts to fix them. The exit survey responses that students filled out at the end of the quarter indicate that students seemed to find the debugging form helpful with self-regulation in debugging process. For further investigation, we attempt to measure students' understanding of the bugs explained on the form. Additionally, we also discuss potential improvements for the debugging form.

ACKNOWLEDGMENTS

Thanks to:

- Dr. Theresa Migler for providing endless support and guidance, especially while pivoting to new thesis.
- Dr. Ayaan Kazerouni for providing invaluable feedback, teaching me about computing education, and being patient with me.
- Professor Chris Siu for teaching me how to program from day 1 and being patient with me even though I make mistakes with autograder scripts.
- Professors Julie Workman and Paul Hatalsky for teaching me fundamentals of computer science and supporting my thesis.
- My family for supporting me no matter what.
- My friends for amazing friendship and always making me laugh.
- Students in CSC 202 Spring22 for participating in this research and making this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

I did not have any exposure to computer science until I came to Cal Poly. When I took my first ever computer science class at Cal Poly, I was often frustrated. I still remember myself often thinking "I have no idea why this code is not working" or "I don't know what I'm doing, I'm lost". I had to quickly learn computer science concepts and be able to apply them in programming assignments. The coding was already challenging, and debugging made it more difficult. As a beginner, I saw so many possibilities where my code could go wrong and I had to "guess" as a beginner where it looked suspicious if I was lost. Of course, some errors could be easily fixed, but the more complex the project was, the higher the chance for bugs. I went through the cycle of introducing bugs and fixing them over and over in different classes. I finally started to gain tips and tricks on how to approach bugs and fix them through this trial and error. I remembered the pain of being lost as a beginner, and with such skill, I wanted to help other students. This lead me to becoming a teaching assistant (TA) for introductory computer science classes at Cal Poly.

During my TA experience at Cal Poly, I have answered countless questions about students' lab assignments, project assignments, and general computer science concepts. Mainly I have helped students with debugging their lab or project assignments. In my experience, students struggled the most with understanding or fixing the bugs. A lot of their questions were "I do not know what is wrong" or "I do not know how to fix this". When I received such questions, I followed up with my own questions:

"What do you understand about this bug so far?" and "What have you tried to fix this bug?". These questions often led to the "aha" moment where they thought of an approach they could take as they were verbalizing and explaining the answers to my questions. Although sometimes students seemed a bit scattered with their hypotheses, attempting to answer my questions appeared to be helpful with gathering their thoughts.

One of my roles as TA was to guide students to understand the bugs and let them come up with their own fixes rather than fixing bugs for the students. Asking questions facilitated students' learning rather than giving statements or instructions. Students answering questions seemed to help with self-regulating initial steps of the debugging process. These experiences inspired me to propose a documentation method to act as *cognitive awareness scaffolding*. Cognitive awareness scaffolding is a support method that helps promoting one's cognitive awareness in problem-solving. The cognitive awareness scaffolding we designed, debugging form, will be discussed in detail in Chapter 4.

Additionally, despite the increase in demand for computer science education, introductory programming classes tend to focus on writing and reading code than on debugging. Learning debugging is more of an implicit goal in the programming assignments. In students' perspective, this could be seen as pressure to "pick up" debugging skills as they write code. This was part of our motivation, to support students in learning debugging.

1.2 Primary Goal

Students utilize different debugging methods [3, 4]. No matter what methods they utilize, the crucial part of debugging is *self-regulation*. Self-regulation refers to the

ability to manage or control one's own behaviour, emotions, work, learning, etc [5]. In context of education, self-regulation lets students manage and organize their thoughts in learning. In this thesis, we propose a documentation method to increase students' cognitive awareness of the debugging process in introductory computer science classes. We also validate its effectiveness by introducing this form to students and acquiring students' responses from an exit survey at the end of the school quarter. Proposing our debugging form method is not meant to oppose or discredit current academic practices of teaching in early computer science classes. Rather, we hope that this method could help students self-regulate their debugging progress and potentially have this method be integrated into existing introductory computer science classes if proven to be helpful.

Additionally, although everyone has a different definition of "good programmer", being able to work together is often mentioned as one of the characteristics of "good programmer" because, in the real world, software engineers work together. The key to teamwork is delivering one's thoughts effectively. They need to step out of their "zone" and deliver their thoughts. When stepping out of that "zone", they need to know what to bring out from the "zone" and organize them to deliver it. We expect our debugging form will help students put together their thoughts which will be a valuable skill.

1.3 Research Questions

We want to learn about the impact of our debugging form as cognitive awareness scaffolding for debugging, focusing on students' experiences. Our main research questions are:

- Is this type of method known to students in introductory computer science classes?
- Will it help students self-regulate their debugging process?
- Will it increase students' confidence in approaching problems in debugging?
- Will students use this type of method in the future?

These questions were taken into consideration when the debugging form and exit survey were designed, which will be discussed in detail in Chapter 4 and Chapter 5 respectively.

1.4 Main Contribution

Our main contributions to this thesis are the following:

- Design of debugging form as cognitive awareness scaffolding
- An experiment to validate the debugging form
- An attempt to measure students' understanding of the bugs in the debugging form

We designed the debugging form based on our thorough research about how to increase metacognitive awareness in computer science education and my personal experience as a TA. We conducted experiments with students in an introductory computer science class to learn about the impact of our debugging form. We validated its effectiveness by exit survey and our rubric to measure students' understanding of the bugs.

Chapter 2

BACKGROUND

2.1 Debugging

Debugging is a natural part of programming, followed after failure to execute the code as the programmer intended to do so. While the time spent on debugging depends on different factors such as the complexity of the program or the programmer's background, it is reasonable to say that debugging is a big part of the programming.

Most of the time, beginners lack the skills and experiences that experts gain over time. No matter what field, we probably all remember the challenge we faced as beginners. Debugging is not an exception. Experts gain experiences that help them determine the approach they should take in debugging. However, debugging isn't something that programmers could easily master, if it is even something that can be mastered at all. For example, interviews with 15 professional software engineers at Microsoft showed that even professionals still do face challenges in debugging as well [6]. Therefore, it is reasonable to assume that beginners have a harder time debugging than experts do. Pea *et al.* noted that the major hurdle a novice programmer must overcome is the major discrepancy between natural conversation and reading or writing programs. The natural language "debugging" where the context or use of other natural language devices, like asking back to help humans understand, is not available for interaction between human and computer therefore creates disconnect [7]. Furthermore, several studies suggest that the difference between novice and expert programmers in debugging is the ability of problem comprehension [8, 9, 10]. For example, Gugerty and Olson observed that expert subjects were faster and more

successful at finding the bug in simple programs compared to novice subjects [8]. They noted that their studies suggest "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 find and isolate the bug.

Additionally, debugging requires several skills at the same time. Katz and Anderson observed that it is clear that debugging is not a single activity, but a set of activities where each component of which may be performed differently depending on the situation [11]. Murphy *et al.* portrayed novice debuggers as new drivers since new drivers "must learn to steer, accelerate, brake, etc. all at once" and "novice debuggers must apply many new skills simultaneously" [3]. Ducassé and Emde proposed a classification of debugging knowledge though whole knowledge is not needed all the time: knowledge of the intended program, knowledge of the actual program, understanding of the programming language, general programming expertise, knowledge of the application domain, knowledge of bugs, and knowledge on debugging methods [12].

The steps of debugging differ slightly by researchers, but the general idea can be broken down into four steps: Understand, diagnose, locate, and correct [13, 11]. Once a programmer realizes the program did not execute as they intended, they know a bug or an equivalent error exists. Then they might start to build and check their "hypotheses" or "theories" of what is the issue they are trying to debug. After the diagnosis, the programmer tries to locate where the bug is happening in the code and fix it. Programmers may go through this loop several times or introduce a new bug while doing so. Any type of confusion could come in at any stage of debugging. For example, a programmer might have successfully located a bug but does not know how to fix it. Or a programmer might not know where the bug is even coming from. They could utilize debugging tools to overcome the step they are stuck at.

2.1.1 Debugging Tools

The utilization of debugging tools depends on the domain of the problem, varying from writing simple print statements to utilizing IDE debugger. There is no one-size-fits-all tool for debugging, but rather a toolbox [6]. Layman *et al.* noted that 15 professional software engineers mentioned 16 distinct debugging tools varying according to their application domain. Although students' debugging domain is simpler than professionals', students also do apply reasonable strategies, more than one technique, to investigate bugs [3, 4]. However, both Murphy *et al.* and Mansur *et al.* noted that some strategies were used ineffectively or inconsistently.

2.2 Metacognition

Although *metacognition* is defined in different ways by various researchers, it is commonly defined as the awareness and understanding of one's thought processes [14]. In simpler words, it is "thinking about thinking". It was first introduced by John Flavell who defined it as "among other things, to the active monitoring and consequent regulation and orchestration of [information processing activities] in relation to the cognitive objects or data on which they bear, usually in service of some concrete goal or objective" [15].

2.2.1 Metacognition in Debugging

Debugging, which is part of problem-solving in programming, could be intimidating to beginners. Several studies suggest that support to promote metacognition helps with problem-solving [16, 17, 18]. For example, Safari and Meskini noted that "the results of the students' performance in each problem solving component showed that

metacognitive approach to problem solving instruction significantly improved the experimental group's performance" [16]. Such results suggest that metacognition plays an essential role in self-regulating the process of problem solving. Therefore, we wanted to design a support method that would promote metacognitive awareness in debugging process.

Chapter 3

RELATED WORKS

Multiple studies have been done to see the effect of metacognition training in various areas. For example, Haller *et al.* and Kramarski and Mevarech mentioned that metacognition helped in students' learning or achievement in the area of reading and mathematical reasoning, respectively [19, 20]. In this section, we focus on metacognition-related research in the computer science field.

3.1 Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance

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 progression 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 [21]. The interventions were made in the experimental group of the traditional web development camp where the control group did not receive such interventions.

The first intervention was the 1-hour problem-solving lecture, given only to the experimental group. This lecture included teaching the six programming problem-solving stages. Then, the physical handout with the previously mentioned problem-solving stages was provided as a second intervention so that campers could track the current state of problem-solving. During the third intervention, upon receiving help from instructors, campers were first asked to describe the problem in the question in de-

tail, what they have tried so far to fix it, and what problem-solving stage they are in with it. The last intervention Idea Garden was implemented in a panel of the Cloud9 IDE. Idea Garden helps programmers to consider new ideas when they are stuck. With such interventions, researchers had predicted that the problem-solving instruction would help campers be more aware of the strategies they used. The first three interventions were applicable to our research context so we decided to replicate them in a similar manner, which will be explained in details in later chapters.

Loksa *et al.* counted the number of end-of-day responses that described a specific strategy other than asking an instructor for help. The researchers found that campers in experimental group were more cognitively aware; they were significantly more likely to write an explicit description of a problem strategy with significantly more words. Researchers noted that the "difference in proportions of help request types was not large" between control and experimental group but based on the request types, the "campers in the experimental group were more likely to select a solution and implement it independently, allowing them to progress to evaluation before requiring help". In terms of productivity, campers were assigned prescribed tasks and had self-initiated tasks for their projects. Although campers have finished similar amounts of prescribed tasks in the same amount of time in both groups, the experimental group completed substantially more self-initiated tasks. Researchers noted that they did not find a significant association between help requests and productivity in the experimental group, meaning campers in the experimental group did not rely on the helpers.

	Control	Experimental
Correct completion rate	52.94%	76.19%
Mean time (minutes)	23.82	22.62
Mean code submissions	7.59	4.48

Figure 3.1: Task performance of control and experimental group [1]

3.2 First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts

Prather *et al.* conducted a controlled experiment to investigate a method that would promote better metacognitive awareness and if it should be incorporated into Automated Assessment Tool (AAT) for better metacognitive training [1]. The students in the experimental group were asked to solve a randomly generated test case immediately after reading the problem prompt in a one-on-one session where researchers observed them thinking out loud. Once they have passed the test case quiz, they were allowed to start writing code.

Prather *et al.* found that students in the experimental group had a higher task completion rate, a lower average time of completion, and a lower average of attempts of completion.

Although it did not completely get rid of metacognitive difficulties, students in the experimental group tended to show higher metacognitive skills and behaviors compared to students in the control group, such as verbalizing the correct understanding of the problem promptly. Based on feedback from students' in experimental group, researchers noted that understanding why the intervention took place seems to correlate with success.

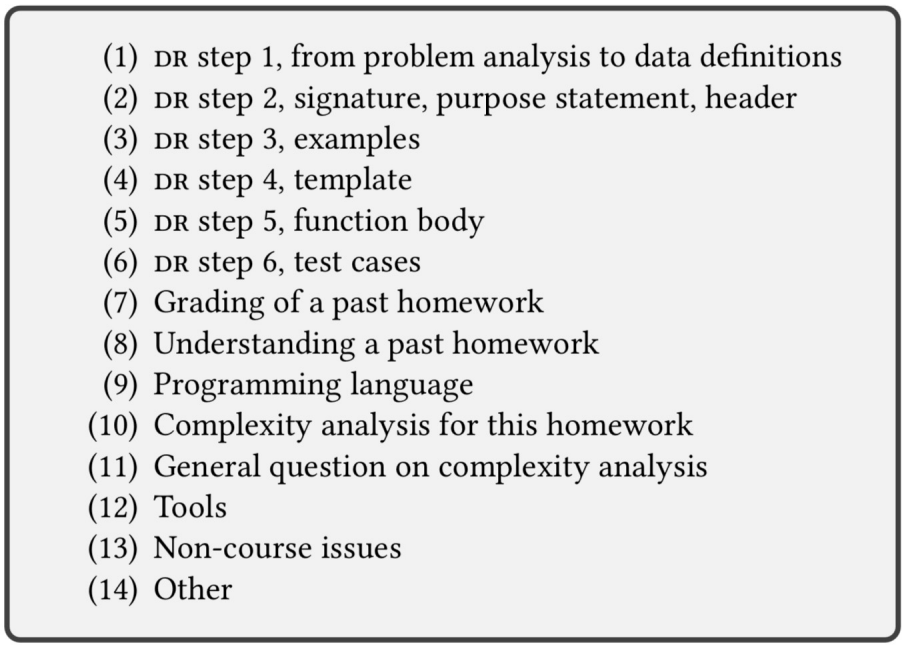

Figure 3.2: Design Recipe options [2]

3.3 What Help Do Students Seek In TA Office Hours

Ren *et al.* presented an approach to studying the technical component of insight into what kinds of help students seek at Teaching Assistant (TA) office hours [2]. They presented two forms that center around the Design Recipe (DR) [22], one for TAs to fill out and another for students.

TAs were instructed to fill out an "exit survey" form after every student meeting. They selected options for what the question is about, designed around the DR steps as Figure 3.2 shows. Researchers adjusted and added more options to fit the course context as DR steps were not sufficient.

TA chose multiple options from Figure 3.2 they discussed during the meeting. For each option, they were given three points *classification* scale to capture the type of help students needed: instruction ("how do I do ..."), clarification ("what is expected for ..."), and verification ("could you check my work on ..."). The last question

asked how the student is performing on the assignment, though this was not further discussed by the authors because it showed a low coefficient of variation.

Students who were attending TA office hours were given a variant of the TA form where students simply chose the items they wanted to discuss. Unlike the TA form, the student form did not include a classification scale due to concerns regarding time spent filling out the form and its effect on student morale. Students filled out the form when they signed up for the TA office hour slot and TA was able to view it on the dashboard queue of the office hour. TA's form was pre-populated with students' chosen entries and the TA filled the form after helping the students.

Ren *et al.* calculated the Hamming distance between DR entries since they form an equal-length bit-vector, and found that the average Hamming distance per assignment decreased over time. However, students matching the TA's entries were more meaningful, not the other way around as TAs are more experts. Therefore, they calculated a *directed* distance, the latest step on which the student and TA disagreed (positive means TA helped a student with a later stage and negative means TA stopped with an earlier stage). If Hamming distance was 0, they called it a *match*. They found that the ratio of matches improved significantly over time. Although the authors did not specifically mention the aspect of self-regulation, this might show that students were able to monitor and assess their questions better over time.

The focus of this research was not aimed towards the metacognition area, their research was applicable to our research since their method was integrated into normal office hour settings, just like how we are trying to with our method. Ren *et al.* shared some considerations in experiment method and design. For example, they mentioned how they initially were considering students fill out the classifiers like TA does but they decided not to because they wanted to "minimize the time spent filling out the

form" and were "concerned about the effect on students morale". Such considerations motivated us to view our method on students' perspectives as well.

Chapter 4

DEBUGGING FORM

Essentially, we wanted the debugging form to act as a "conversation" between the students and instructor, as if students are asking questions about debugging which promotes metacognitive awareness. Murphy *et al.*'s suggestion was one of the inspirational foundation in the debugging form design. They 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.

4.1 Form Design

Similar to the Design Recipe [22], a structured approach to problem-solving in six steps, we created a five debugging stage for our form: Issue Category, Issue Description, Previous Attempts, Solved by Myself, and Asked Question.

4.1.1 Issue Category

The first part of debugging form is identifying which debugging stage students are stuck at. This would let students think about the next action they need to take in order to successfully debug instead of directly jumping into certain stage of debugging without going through the previous stages.

Below are the debugging stage categorization we provide on our debugging form.

1. Understanding an error
2. Locating an error
3. Testing an error
4. Fixing an error
5. Auto-grader Feedback
6. Other

Additionally, we added a specific category called "Auto-grader Feedback". 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. Details about the auto-grader are explained in Chapter 6.

4.1.2 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?".

4.1.3 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?".

4.1.4 Solved by Myself

This section is straightforward where students check whether they solved the bug by themselves or not.

4.1.5 Asked Question

This section is also straightforward where students check whether they asked the question to the instructor about the particular bug or not.

Chapter 5

EXIT SURVEY

To validate the effectiveness of the debugging form, we created an exit survey.

5.1 Exit Survey Design

The exit survey was sent out at the end of the quarter to gain insights into their experiences. The survey questions were designed to answer our research questions and let participants share their experiences. Possible response types were the Likert scale, Yes/No/Maybe, and free-response. In our survey, Likert scale response consists of a 5-point scale where each point represents following: 1-Extremely Disagree, 2-Somewhat Disagree, 3-Neutral, 4-Somewhat Agree, and 5-Extremely Agree.

5.1.1 Questions

Q1. Have you used this type of documentation method in the past? (Yes/No)

Question 1 attempts to identify if students in introductory computer science class are aware of this type of method to help with their debugging process.

Q2. The debugging form helped me with verbalizing the errors (Likert Scale)

Question 2 attempts to identify the impact of debugging form on students' ability to verbalize the debugging progress including descriptions of the bugs and attempts to solve them.

Q3. The debugging form helped me with explaining the debugging progress to instructors (Likert Scale)

Question 3 attempts to identify the impact of debugging form on students' ability to explain the bugs to instructors since they have already written down their thought processes on the form.

Q4. The debugging form helped me stay aware of my debugging progress (Likert Scale)

Murphy *et al.* studied students' debugging strategies and noted that "many unproductive activities appeared to stem from insufficient metacognition. Some students did not recognize when they were stuck, thus they did not know to try a different approach" [3]. This shows that it is crucial to self-regulate the problem-solving progress. Question 4 attempts to identify the impact of the debugging form on students' ability to monitor their debugging progress.

Q5. I have solved at least one bug that I was going to ask instructors about while filling out the debugging form (Yes or No)

We noticed "aha" moment several times while helping students, where students realize an approach they could take to fix the bug. Question 5 attempts to identify students' independence, the "aha" moments before asking questions.

Q6. I feel more confident in approaching debugging process because I learned and used the debugging forms (Likert Scale)

Several studies developed and validated self-efficacy measurement [23, 24]. Although self-efficacy is out of the scope of our research, we believe it is still a valuable field to explore. There are questionnaires with multiple questions that measure general self-efficacy, but we wanted to specifically focus on the impact on confidence in approaching debugging. Although responses to question 6 cannot represent the whole self-efficacy in debugging, it attempts to measure students' confidence level.

Q7. Do you see yourself utilizing this type of method in future computer science classes? (Yes/No)

Question 7 attempts to identify number of students who would utilize our method in the future. We believe students' willingness to use this method again is one of the ways to validate its effectiveness.

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

Instructors and researchers did not receive any complaints from students. However, we believe students are less likely to share negative feedback without anonymity during the quarter. Therefore, we created question 8 to give students a chance to share their thoughts anonymously.

Chapter 6

EXPERIMENT DESIGN

6.1 Course Context

The academic year at Cal Poly San Luis Obispo consists of three quarters: Fall, Winter, and Spring. Each quarter consists of 10 weeks of instruction and an additional finals week. Since most introductory courses are a sequence of prerequisites, students take one computer science course each quarter until they finish the chain, after which they can take technical elective courses. Although there isn't a formal definition of "introductory" courses, we consider the following four courses "introductory" at Cal Poly SLO.

- CSC 123: Introduction to Computing
- CSC 101: Fundamentals of Computer Science
- CSC 202: Data Structures
- CSC 203: Project-Based Object-Oriented Programming and Design

Most computer science, software engineering, and computer engineering majors are required to take these courses in sequence at Cal Poly SLO, although there are different ways to fulfill them. Most computer science classes are composed of lecture and laboratory sections. Students attend lectures first and then have a laboratory section which serves as a teaching assistant or professor office hour where students can ask questions or work on their own.

6.1.1 CSC 202: Data Structures

We have decided that CSC 202 is suitable for introducing our debugging form as a cognitive awareness scaffolding. Students who are taking CSC 202 should have experiences in programming, either through equivalent credit or taking prerequisite programming courses. CSC 202 teaches data structures (in Python3 at the time when this thesis was written) and the programming assignments require such knowledge to complete them. There are two types of programming assignments in CSC 202: lab and project. Labs are smaller programming assignments that require them to have a basic understanding of the data structures concept they learned during the lecture. Projects are larger programming assignments with more complex specifications. Therefore, students were typically given more time to complete projects than labs.

For this research, the debugging form was introduced to students in four sections of CSC 202. The sections were taught by two different instructors but with identical course material with the same assignments and exams. One instructor taught one section and the other instructor taught three sections. Students filled out the debugging forms and voluntarily submitted them as a part of project submissions through GitHub repositories, but did not submit the form for labs. Projects are larger and more complex assignments which involve more coding and debugging, therefore we decided projects are more suitable to observe the effectiveness of the debugging form. The debugging forms were not graded and did not affect their grades negatively in any way. However, students were offered 3% extra credit for each form per project, but only if they had a minimum of two entries on the debugging form. Everyone in the class was offered an opportunity to receive extra credit if they submitted a qualifying debugging form for each project.

6.1.2 Projects

Projects in CSC 202 are intended to be greater length and more complex than labs. Unlike labs where students are encouraged and allowed to work together, projects have to be completed individually and no collaboration is allowed. Students submitted their project code on their GitHub repositories and submissions were graded four times per day on the day of the deadline. The credit was given based on the number of test cases that students passed and scaled by which deadline the submission qualifies for. Students were eligible to earn 105% of the credit the day before the due date (only one midnight run), 100% on the due date, and 85%, 70%, 50% on the next class day after each due date. Except for the 105% extra credit deadline, students had four different days where their project code was graded. We collected the debugging forms for following projects: project 2, 3a, 3b, and 4 where project 3 had two parts: a and b. However, project 3 took code submissions for both parts in same GitHub repository, causing some debugging forms for 3a being replaced by debugging forms for 3b. It was almost impossible to track down when it was replaced therefore we excluded project 3a from the analysis. In the rest of this paper, project 3b will be referred as project 3. Project 1 was excluded because the project specification was already released before we got to introduce the debugging form. Project 5 was also excluded because the submission deadline did not fit our timeline for analysis. However, survey responses still reflect students' experience of utilizing debugging form for up to 5 projects. Only the analysis of debugging form and grades involve 3 forms.

6.1.3 Auto-Grader

The project assignments are graded by an auto-grader developed by Christopher Siu, a lecturer at the Department of Computer Science and Software Engineering of Cal

Poly. Students submit project code on their GitHub repository and the auto-grader pulls their submissions to run assignment test cases, assigns points based on the test cases they passed, and provides feedback. There are two types of feedback: Coverage and Traceback. First, the auto-grader tests the coverage of students' test cases. If students failed one of their tests or their tests did not cover 100% of their code, they are informed about it in the feedback. Testing the coverage of students' test cases utilizes open source coverage.py [25]. Another part of the feedback is Traceback, which can be caused by one of three different reasons: failed assertions, failed diff, and timeout. Traceback is basically what students would have seen from their terminal when they ran their code with certain tests on their machine.

6.2 Instructions

To help students familiarize themselves with the idea of the debugging form, we suggested that it is a written version of what students would have explained to instructors if they were to ask questions.

We asked students to fill out the debugging form during the debugging process of each bug because we believe the support should be provided while they are debugging to help with their thought process. Kapa also noted that learning environments which supplies metacognitive support during the process of problem-solving are significantly more effective than the learning environments that provide same support at the end of the process or do not provide any metacognitive support [26].

Students were also instructed to fill out the debugging form for 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 or not to be as unintrusive as possible.

Additionally, students were required to present specific entries from the debugging form about the bug when they were asking instructors about it and if it requires instructors to look at their code or error messages. Because at minimum, if students have to ask instructors about the bug, that is "major". However, we did not want to discourage students to ask questions because they were required to present the form to "prove" their attempt to solve the bugs when they do not know where to start. Therefore, they were allowed to put previous attempts along the line of "Do not know where to start" to not discourage them from asking questions. Having students present the debugging form also acted as a "guard" to "observe" that students are actually filling out the debugging form. This is one of the interventions designed by Loksa *et al.* in a web development camp that we mentioned. Loksa *et al.* asked students questions upon their help request where they asked students to describe the details of the problem, what they have tried so far, and what problem-solving stage they are in [21]. In our experiment, such questions were replaced by presenting the debugging form. Our students were not required to present them when they were asking questions on the class Piazza, since ideally writing a post should already involve metacognition.

6.3 Limitations

It is important to recognize a few limitations of this experiment and the other factors that we have omitted for this research.

- Just like any learning environment, students' prior programming experiences are different. As previously mentioned, there are different ways to fulfill the "introductory" computer science courses at Cal Poly SLO. For example, one of the common tracks is taking a sequence of CSC 123 → CSC 101 → CSC 202

or skipping CSC 123 and CSC 101 because they have AP Computer Science or equivalent credit. However, the scope of this thesis is limited to proposing and checking the validity of the debugging form, therefore we do not attempt to gain individual student's background information as part of the analysis. It is expected that majority of students is taking CSC 123 → CSC 101 → CSC 202 sequence since Spring quarter is the "on-quarter" to take CSC 202. Mostly, students with credit to skip prerequisite courses would have taken CSC 202 earlier than Spring quarter.

- As mentioned in section 6.2, students have the freedom to define what "major" bug means. Although this was done to be as unintrusive as possible, it means everyone has different definition of "major" bug to decide what to write on the debugging form or not.

- Researchers were not able to monitor students filling out the form since the work is done outside the classroom. This unmonitored system causes obvious problems such as entry without a detailed description of the bug even though students were instructed to put as many details as possible in their own words. It increases chance of "slacking off". This could have limited the effectiveness of the debugging form activity, as we hypothesized that writing down the bug details would promote metacognitive awareness.

- When we introduced the debugging form and the instructions to use it, we also informed students about the potential benefits of the debugging form to motivate them. Prather *et al.* noted that understanding why the intervention took place seems to correlate with success among students in experimental group [1]. Therefore, us informing students about the potential benefit of the debugging form may have caused bias toward their survey responses.

Chapter 7

SURVEY ANALYSIS

In this chapter, we provide data points of exit survey responses and analyze them to answer our research questions. There were a total of 38 participants in the exit survey. Out of 38 responses, we filtered out five responses. Two participants did not give us consent and three participants mentioned they never used the debugging form or never submitted it yet took the survey. Therefore, we ended up with 33 responses for our analysis. Additionally out of the final 33 responses, two participants refused to release their project grades. Those two responses are still included in the debugging form and survey analysis but excluded in any project grades related analysis.

We had 4 questions that had Likert scale as response types. In Likert scale, the lower scales mean negative response where it is closer to disagreeing with the question statement. However, in the context of our survey questions, we want to note that lower scale is closer to "neutral" rather than negative because disagreeing with our question statement does not necessarily mean that debugging form was harmed them.

7.1 Statistics on Each Questions

In this section, we identify statistics on responses to each question.

7.1.1 Question 1: Have you used this type of documentation method in the past?

Question 1 asked if participants have heard of this type of documentation method. Out of 33 responses, 4 participants responded Yes (12%) and 29 participants responded No (88%). Figure 7.1 represents pie chart of the responses. Although our debugging form type of documentation method seemed to be unknown among our participants, we were surprised by number of participants who were aware of it given they are in an introductory computer science class. It is possible that similar method was taught in one of the prerequisite courses.

It depends on the instructors but explicitly teaching how to debug a program is not part of the required curriculum. Moreover, it is not too long ago that metacognition gained attention in problem-solving stages and it is only one of many support methods that exist. Therefore this was not a particularly surprising result to us.

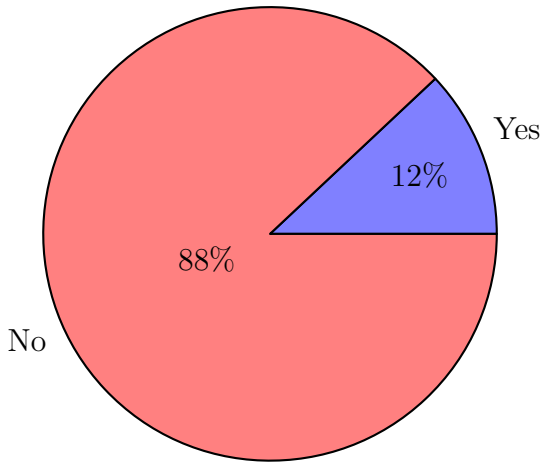

Figure 7.1: Pie chart of participants' familiarity with documentation method in debugging

7.1.2 Question 2: The debugging form helped me with verbalizing the errors

Question 2 asked about the impact of the debugging form on verbalization. Figure 7.2 presents the distribution of the responses. The mode and median were both 4, being the highest out of a set of questions attempting to measure the impact of the debugging form (questions 2, 3, and 4). Based on this result, participants seem to find the debugging form activity helpful with verbalization of the debugging process.

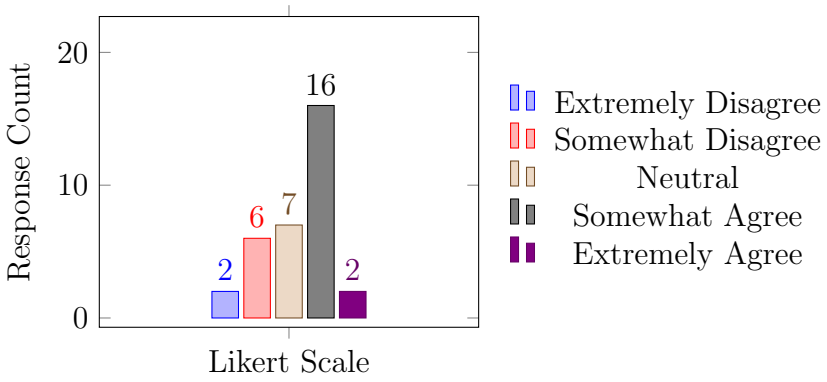

Figure 7.2: Column chart of responses about debugging form's verbalization support

7.1.3 Question 3: The debugging form helped me with explaining the debugging progress to instructors

Question 3 attempted to measure the impact of the debugging form on bug explanation. Figure 7.3 presents the distribution of the responses. We expected the distribution to be similar to Question 2 because we imagined the verbalization ability would be correlated with the explanation ability. However, the plots showed different trends and we hypothesize that this might be because there were participants who did not ask questions to instructors based on their debugging form entry. The question specifically included wording "explaining the debugging progress to instructors" and we believe this might have affected the responses. Nevertheless, we did not filter

out the responses by those participants because their responses could have been still based on the situation of if they were to explain.

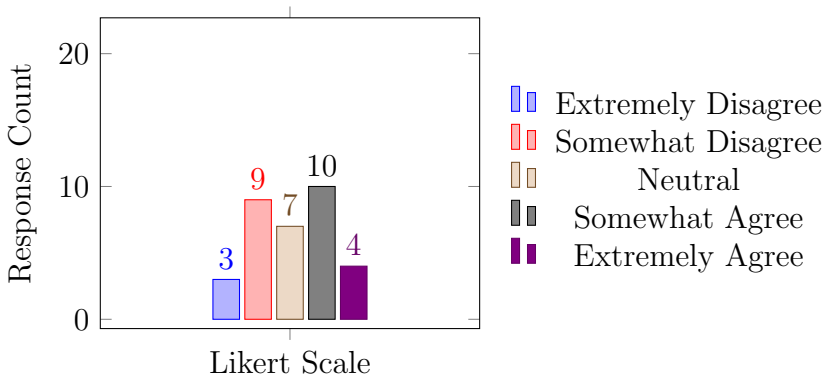

Figure 7.3: Column chart of responses about debugging form's explanation support

7.1.4 Question 4: The debugging form helped me stay aware of my debugging progress

Question 4 attempted to measure participants' acknowledgment of debugging process; in simpler words if participants are aware of what debugging stage they are in and what they are trying to solve. Figure 7.4 presents the response distribution. We saw a similar trend with Question 3's responses with this question.

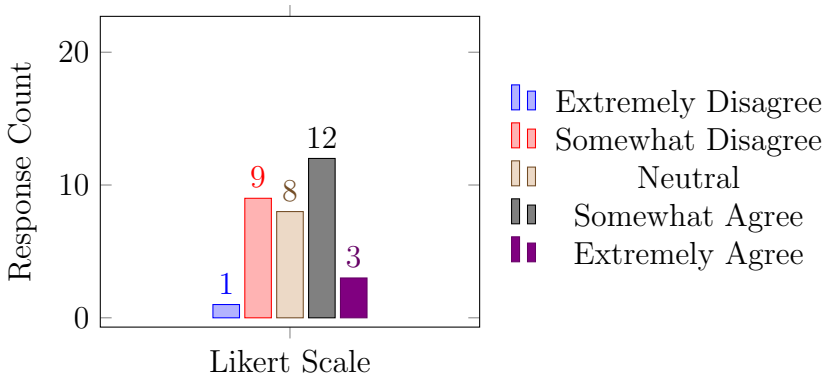

Figure 7.4: Column chart of responses about debugging form's support for acknowledgement of problem

7.1.5 Question 5: 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%). Figure 7.5 represents pie chart of the responses.

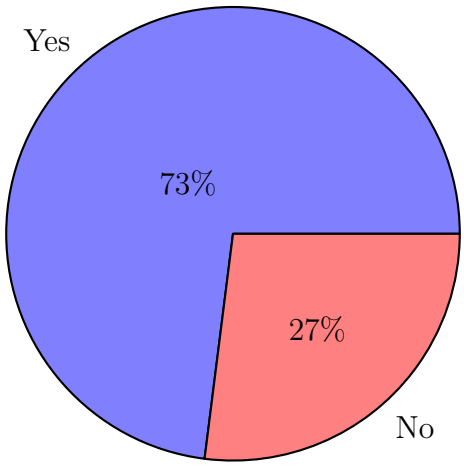

Figure 7.5: Pie chart of participants solving at least one bug that they were going to ask instructor about

A significant number of participants indicated that they had solved at least one bug that they were going to ask instructors while they were filling out the debugging form. This might mean that the debugging form promoted independence in participants' debugging processes.

Additionally, in my TA experience, there were times where the help queue was too long that not every participant who needed help could get help during the laboratory section. If the debugging form helps with promoting independence and support problem-solving, it would be extremely helpful to integrate such support into real classroom settings.

This result also supports our motivation of this thesis that participants often get "aha" moments as they are verbalizing the details about their debugging process of the problem. Participant 26 said they "think that learning to understand the problem that you are facing before you ask for help is very valuable".

7.1.6 Question 6: I feel more confident in approaching debugging process because I learned and used the debugging forms

Question 6 asked participants if they feel more confident in approaching the debugging process because they have learned and used debugging forms. Figure 7.6 presents the distribution of the responses.

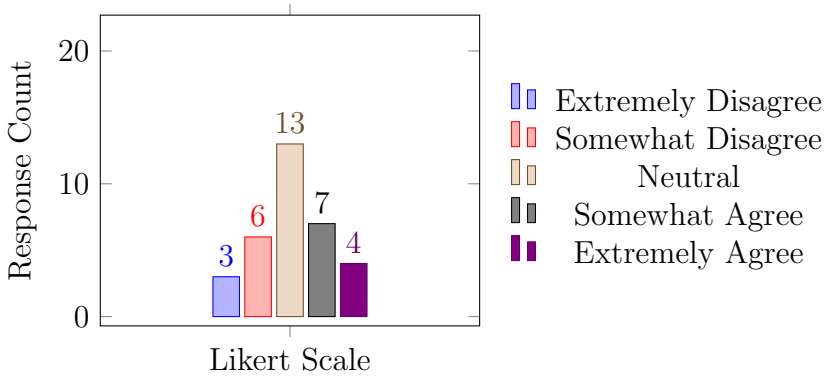

Figure 7.6: Column chart of participants' response about future use of the debugging form

Although the mean is slightly higher than 3, both the median and mode were 3. We believe there is so much more that is related to confidence in programming and debugging and the debugging form cannot resolve all of potential causes.

Table 7.1 presents the summary of statistics (mean, mode, median, min, and max) of Questions 2, 3, 4, and 6 which had Likert scale responses.

Table 7.1: Statistics of question 2, 3, 4, and 6 responses

	Mean	Min	Max	Mode	Median
Question 2	3.30	1	5	4	4
Question 3	3.09	1	5	4	3
Question 4	3.21	1	5	4	3
Question 6	3.09	1	5	3	3

7.1.7 Question 7: Do you see yourself utilizing this type of method in future computer science classes

Question 7 asked participants if they see themselves utilizing this type of documentation method in future computer science classes as a support in the debugging process. Out of 33 responses, 9 participants responded Yes, 7 participants responded No, and 17 participants responded Maybe. Figure 7.7 represents pie chart of the responses.

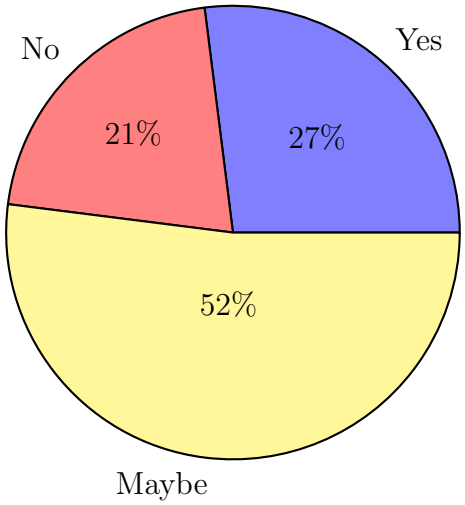

Figure 7.7: Pie chart about participants' willingness to use the debugging form in the future

This suggests that around half of the participants are not sure about using the debugging form in the future. Since everyone is different and there will not be one ultimate method that works for everyone, it is expected to have participants who did not find this method helpful. However, we believe 8 weeks without monitoring might have been too short for participants to decide if this method will be helpful in their future

computer science classes because it is easy to "slack off" researcher is not observing the participants.

Although the "subjective" responses, responses about what participants believe, might suggest that participants did not benefit as much, the "objective" response being 74% participants solving at least one bug without help that they thought they needed the help for show significant impact of the debugging form.

7.1.8 Question 8 (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.

We learned that some participants used the debugging form after they had completely solved the bug. Participant 37 mentioned "The forms would be extremely helpful had I used them during the debugging process. However, I used the forms after the fact". Additionally, participant 14 also mentioned "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". We intended for participants to use the debugging form as they try to solve the bugs. However, there seems to be a disconnection regarding when to use debugging forms. This might explain participant 26's comment, "I don't feel like the form did a good job of facilitating that as it broke up my thinking process by making me fill out the form". However, they did not specify when they have filled out the form throughout the debugging process. Therefore, this disconnection might have caused "side effect" due to such misunderstanding.

The provided options for "Issue Category" were steps of debugging we identified through research. Our intention was for participants to identify where they are at

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" was expected to provide specific bug types as options. Participant 6 commented "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". However, identifying the debugging step (what the column was meant for) and describing the bug in detail are connected yet separate tasks in the debugging form. If participants wanted to mention specific types of bugs, it would be the next column, "Issue Description". Participant 31 also said "More options for stuff like that in the 1-6 category other than Other might help articulate exactly what's wrong". Based on these two participants' comments, we identified the existing disconnection on tasks.

However, this gave us insights about how providing specific possible bug types could have worked as a guide to identifying the bugs. Additionally, one of the tutoring strategies as a TA was ensuring participants that bugs inevitably occur and are common to help with participants' morale and self-efficacy. Based on the feedback, we believe providing specific bug types upfront could ensure participants that the bugs they are experiencing are expected and normal.

7.2 Improvements

Based on participants' feedback, we identified potential improvements on the design of the debugging form and the experiment.

7.2.1 Form Design

Instead of calling the first column "Issue Category", we believe calling it "Debugging Stage" would reduce participants' confusion. Based on participants' feedback, we believe the word "Category" gave the impression of participants having to choose a specific type of bug. Once such understanding is firm, we could explicitly instruct participants to identify the previous debugging stage's findings on issue description which will show the evidence of understanding their status with current debugging.

Additionally, participant 26 noted that the debugging form "broke up my thinking process by making me fill out the form". We see that this is a potential issue for some participants who find writing down the process difficult as it breaks their thought process. One of potential solutions is integrating the online version of debugging form embedded on IDE where user could indicate the start of debugging session and let them explore specific code block. The session could log changes, user's thoughts, flow of debugging session, and more. This might decrease the pressure or hassle of having to go back and forth with the debugging form and their code. Additional research is required in this area.

7.2.2 Experiment Design

Unlike Loksa *et al.* who gave explicit instructions on programming problem-solving stages in their study, we only briefly touched on stages of debugging when we gave instructions on how to fill out the debugging form. Although the debugging form had participants identify the debugging stages of each entry, they might have lacked a fundamental understanding of them.

Furthermore, if the researcher is the instructor or TA, they certainly will be able to do more observation on participants. This will decrease the disadvantage of not being able to observe participants closely as interview-based research allows to yet still integrate it into normal course settings.

Chapter 8

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 [27]. Therefore, we have created a tool to measure metacognitive awareness in debugging. Aside from self-report survey results by students' perspective, we wanted to perform metacognitive awareness measurement on how much *we* think students understood the bugs they logged on the debugging form. However, it is possible that students did not put the full description of their understanding on the debugging form. Therefore, students might have more or less understanding of our measurement. It is important to note that we have limited information and our measurement is solely based on what students have written down on the debugging form.

8.1 Form Analysis Rubric

We created a "rubric" to measure their understanding of their bugs. The rubric has three levels where each level has to meet specific criteria to qualify for it. The level was assigned on each entry of the debugging form. If each entry's level was different, we took the average and rounded it up. We chose to round up because the existence of higher level shows that they are capable of understanding the bug at that level. Table 8.1 presents which criterion needed to be qualified to be assigned each level.

Table 8.1: Rubric for the form analysis

Criteria	Level 1	Level 2	Level 3
Mention basic information about the error	✓	✓	✓
Describe the faulty behavior of program at high level language		✓	✓
Describe the faulty behavior of program at low level language and/or shows evidence of previous debugging stage hypothesis			✓

Submissions qualified for level 1 understanding if the entry only mentioned the basic error type of the bug and nothing else. To qualify for level 2 understanding, on top of level 1's qualifications, the entry must describe the faulty behavior of the program in a high-level language. Level 3 implies the most advanced understanding of the bug in our rubric. First, on top of level 2's qualifications, the entry must explain the faulty behavior of the program in a low-level language like specifying the code block of the error or mentioning algorithms of the code. Additionally, showing the evidence of previous debugging stages explicitly or implicitly (if applicable) was part of level 3's criteria as well. We were a bit lenient with this category since each entry was at different debugging stage therefore we cannot expect same amount of information to be mentioned.

8.1.1 Rubric Usage Example

To better explain that qualifies for each level, we present Table 8.2 which shows entry examples of each level. The examples are taken from the actual debugging forms submitted by students.

Table 8.2: Entry example for each level placement

Example	Level 1	Level 2	Level 3
Wrong output	✓		
For the concordance functions, it would to print out the entire line over and over instead of each word		✓	
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'			✓

8.1.2 Results

Figure 8.1 presents the results of measuring students' metacognitive awareness using our rubric.

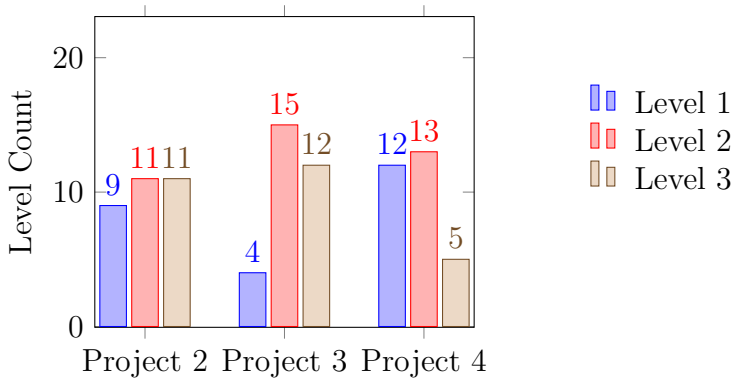

Figure 8.1: Column chart of students' Project 2, 3, and 4 debugging form level placements

We want to note that total number of the forms for each project is different because 10 students did not submit the debugging form for every project. Students who never turned in the form were already filtered out in the beginning.

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. We expected the level placement over time to stay at least

consistent. However, we observed that, based on our rubric, students' understanding of the bugs decreased from project 3 to project 4. We hypothesized that this trend might be correlated with project difficulty and decided to look at average grade of each project.

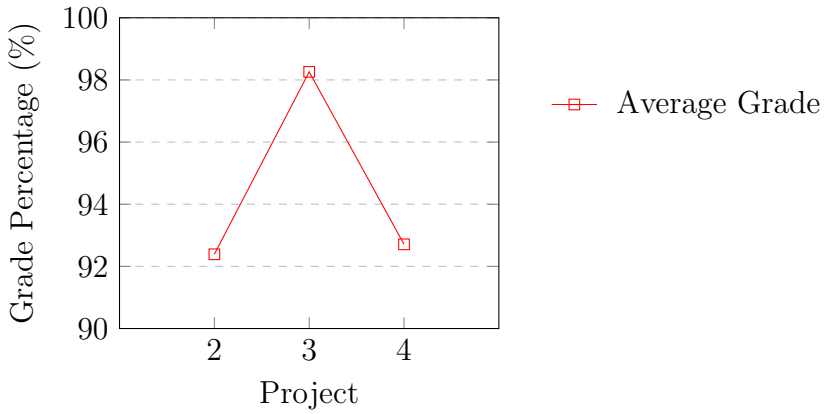

Figure 8.2: Graph of average grade on each project

Figure 8.2 presents the trend of average grade on each project. The average grade for project 2, 3, and 4 were 92.39%, 98.26%, and 92.71% respectively. We want to note that we did not observe individual or average grade until we placed every submission into certain level to avoid any bias. We see a trend that the average grade increased from project 2 to 3 but decreased from 3 to 4. This is similar to students' level placement trend where lower level decreased and higher level increased from project 2 to project 3 but reversed for from project 3 to project 4. We interpret this similarity in trends as a correlation between average grades and students' understanding of bugs; students' understanding seemed to decrease as the difficulty of project increase.

Chapter 9

THREATS TO VALIDITY

9.1 Self-Selection Bias

Self-selection bias refers to a distortion of an expected statistical result due to self-selection of subjects [28]. Offering extra credit for submission of debugging forms and voluntary participation in the exit survey were double-edged swords in this experiment. Our intention of offering extra credit was to motivate students to use debugging forms. Doing so could have brought balance to our sample between students who were going to participate no matter what and those who would not plan to. However, voluntary participation means students self-selected whether or not to participate in the survey. Thus, our sample size got affected by it.

Additionally, students were eligible to take the survey only if they have submitted at least one debugging form throughout the quarter. Therefore students' responses might not carry the same weight because number of the submitted debugging forms differ by students.

9.2 Limited Information

Unlike similar studies where researchers were able to observe participants, we were not able to do the same because of the nature of normal classroom setting where students work on assignments at their own time. Therefore, our analysis relies on what students have provided on the debugging form. Furthermore, we had average

70 students who turned in the debugging forms yet only half of them participated in the survey.

9.3 Lack of Peer Review Process

All experiment design, survey questions, rubric, and level placements based on the rubric were designed by one researcher. While peer review process is the ideal way to perform such analysis or design, we did not have enough time and resource to go through such process. Schraw noted that although it is relatively easy to collect the data, it is far more challenging to implement a rubric to capture one's thought process in data [29]. To design reliable measurement tools, peer review process would be crucial especially because metacognition is complex to measure.

Chapter 10

CONCLUSION

In this thesis, we developed a cognitive awareness scaffolding, debugging form, to promote metacognitive awareness in debugging for students in introductory computer science class. We validated its effectiveness through analysis of students' survey responses, "measured" students' metacognitive awareness on bugs, and proposed improvements on the debugging form. We found that the debugging form seem to act as meaningful support and help with verbalization in problem descriptions. Although one specific method cannot completely resolve difficulties in debugging or support entire debugging process, we argue that the debugging form has a high potential to be integrated into real classroom setting to support students' debugging process based on our analysis.

10.1 Future Work

To further validate its effectiveness of the debugging form in introductory computer science classes, the next step is to conduct an experiment with control and experimental group under same material and instructor to compare the results between two groups. We anticipate that students who are taking the course in the same quarter to have "closest" programming background at least at Cal Poly SLO. With such environment set up, the only major difference between two groups would be providing the debugging form as cognitive awareness scaffolding or not. We were not able to conduct such experiment because one of the instructors who teach three sections let students to come to any lecture or laboratory sections which makes it impossible for

the instructor to keep track of which student should be presenting the debugging form entry when they are asking questions. Furthermore, we could not guarantee balance between the number of students in each group since the participation was voluntary.

Additionally, like previously mentioned in the section 9.3, our research lacked peer review process. In the future, everything we mentioned in section 9.3 should go through peer review process to validate the design of methods and analysis using such methods.

Finally, another research question to further investigate is "Is there correlation between assignment completion time and metacognitive awareness?". We believe assignment completion time is another indicator of students' performance along with project grades, and investigating the correlation between students' level of metacognitive awareness and project completion time would provide insight on the importance of metacognitive awareness in problem-solving as well.

BIBLIOGRAPHY

[1] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 531–537, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Yanyan Ren, Shriram Krishnamurthi, and Kathi Fisler. What help do students seek in ta office hours? In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, page 41–49, New York, NY, USA, 2019. Association for Computing Machinery.

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

[4] Rifat Sabbir Mansur, Ayaan M. Kazerouni, Stephen H. Edwards, and Clifford A. Shaffer. 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 Calling '20, New York, NY, USA, 2020. Association for Computing Machinery.

[5] self-regulation, n.

[6] Lucas Layman, Madeline Diep, Meiyappan Nagappan, Janice Singer, Robert Deline, and Gina Venolia. Debugging revisited: Toward understanding the

debugging needs of contemporary software developers. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 383–392, 2013.

[7] Roy D Pea, Elliot Soloway, and Jim C Spohrer. The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics*, 9, 1987.

[8] L. Gugerty and G. Olson. Debugging by skilled and novice programmers. *SIGCHI Bull.*, 17(4):171–174, apr 1986.

[9] Iris Vessey. Expertise in debugging computer programs: An analysis of the content of verbal protocols. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(5):621–637, 1986.

[10] Murthi Nanja and Curtis R Cook. An analysis of the on-line debugging process. In *Empirical studies of programmers: Second workshop*, pages 172–184. Norwood, NJ: Ablex, 1987.

[11] Irvin R Katz and John R Anderson. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction*, 3(4):351–399, 1987.

[12] M. Ducassé and A.-M. Emde. A review of automated debugging systems: Knowledge, strategies and techniques. In *Proceedings of the 10th International Conference on Software Engineering*, ICSE '88, page 162–171, 1988.

[13] Renée Mccauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. Debugging: A review of the literature from an educational perspective. *Computer Science Education*, 18, 06 2008.

[14] metacognition, n.

[15] J H Flavell. Metacognitive aspects of problem solving. In L B Resnick, editor, *The Nature of Intelligence*, pages 231–235. Earlbaum, Hillsdale, NJ, 1976.

[16] Yahya Safari and Habibeh Meskini. The effect of metacognitive instruction on problem solving skills in iranian students of health sciences. *Global Journal of Health Science*, 8(1):150–156, Jan 2016.

[17] Catherine Aurah, Setlhomo Koloi-Keaikitse, Calvin Isaacs, and Holmes Finch. The role of metacognition in everyday problem solving among primary students in kenya. *problems of education in the 21st century*, 30, 01 2011.

[18] Gökhan Özsoy and Ayşegül Ataman. The effect of metacognitive strategy training on mathematical problem solving achievement. *International Electronic Journal of Elementary Education*, 1(2):67–82, 2009.

[19] Eileen P. Haller, David A. Child, and Herbert J. Walberg. Can comprehension be taught? a quantitative synthesis of "metacognitive" studies. *Educational Researcher*, 17(9):5–8, 1988.

[20] Bracha Kramarski and Zemira R. Mevarech. Enhancing mathematical reasoning in the classroom: The effects of cooperative learning and metacognitive training. *American Educational Research Journal*, 40(1):281–310, 2003.

[21] Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 1449–1461, New York, NY, USA, 2016. Association for Computing Machinery.

[22] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to design programs: an introduction to programming and computing*. MIT Press, 2018.

[23] Mark Sherer, James Maddux, Blaise Mercandante, STEVEN PRENTICE-DUNN, Beth Jacobs, and Ronald Rogers. The self-efficacy scale: Construction and validation. *Psychological reports*, 51:663–671, 10 1982.

[24] Gilad Chen, Stanley M. Gully, and Dov Eden. Validation of a new general self-efficacy scale. *Organizational Research Methods*, 4(1):62–83, 2001.

[25] Ned Batchelder. Coverage.py.

[26] Esther Kapa. A metacognitive support during the process of problem solving in a computerized environment. *Educational Studies in Mathematics*, 47(3):317–336, Sep 2001.

[27] Annemieke E. Jacobse and Egbert G. Harskamp. Towards efficient measurement of metacognition in mathematical problem solving. *Metacognition and Learning*, 7(2):133–149, Aug 2012.

[28] self-selection, n.

[29] Gregory Schraw2. Measuring self-regulation in computer-based learning environments. *Educational Psychologist*, 45(4):258–266, 2010.

[30] Abdulaziz Alaboudi and Thomas D. LaToza. An exploratory study of debugging episodes. *arXiv:2105.02162 [cs]*, May 2021. arXiv: 2105.02162.

[31] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. An analysis of patterns of debugging among novice computer science students. *SIGCSE Bull.*, 37(3):84–88, jun 2005.

[32] Anthony Robins, Patricia Haden, and Sandy Garner. Problem distributions in a cs1 course. *Conferences in Research and Practice in Information Technology Series*, 52, 01 2006.

[33] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and correcting java programming errors for introductory computer science students. volume 35, pages 153–156, 01 2003.

[34] response, n.

[35] Gregory Schraw and David Moshman. Metacognitive theories. *Educational Psychology Review*, 7:351–371, 12 1995.

[36] L R Izzati and A Mahmudi. The influence of metacognition in mathematical problem solving. *Journal of Physics: Conference Series*, 1097:012107, Sep 2018.