Read the Debug Manual: A Debugging Manual for CS1 Students

Rita Garcia, Chieh-Ju Liao, Ariane Pearce
University of Adelaide
Adelaide, South Australia
rita.garcia@adelaide.edu.au
chieh-ju.liao,ariane.pearce@student.adelaide.edu.au

***Abstract*—Debugging manuals help programmers develop strategies to reduce program issues. These manuals assume the reader has prior programming knowledge to comprehend the presented strategies. While this assumed knowledge is above Introductory Programming (CS1) students' programming abilities, it does illustrate a variety of strategies they can use during the debugging process to help them realise different approaches that are available to successfully debug a problem. In this research paper, we present a debugging manual, *Read the Debugging Manual* (RTDM), that is designed for CS1 students. The manual presents a variety of strategies that can be applied across the debugging process without assumed prior programming knowledge. A pilot study was conducted to evaluate the manual, where CS1 students were provided the manual as a debugging intervention for a programming activity. Students were invited to share their experiences and feedback on the manual by participating in a survey. The results show students wanting worked examples in the manual to practice strategies. Providing students with the manual seems to be supporting Self-Regulated Learning (SRL) strategies, helping students recall existing strategies and introducing them to new ones. The paper concludes with future research opportunities to improve the manual.**

***Index Terms*—CS1; debugging; debugging manual; Self-Regulated Learning**

I. Introduction

Providing Introductory Programming (CS1) students guidance during the debugging process can help them better understand the type of mistakes they make, why they make them, and how to avoid them [1], [2]. Debugging manuals are available for guidance, helping programmers develop strategies to reduce programming issues [3], [4]. These manuals assume the reader has prior programming knowledge, which makes the manuals better suited for experienced programmers and not CS1 students with limited programming knowledge. The manuals bring in industry scenarios to emphasise strategy application, but for novice programmers with limited or no experience in programming, these manuals' presentation of strategies and skills could be a challenge for them to comprehend. Yet, such manuals can still be beneficial for CS1 students, because they can illustrate a variety of strategies to apply during the debugging process, helping them realise different approaches are available to successfully debug a problem. In this paper, we present a debugging manual, *Read the Debugging Manual* (RTDM), designed for CS1 students. The goal of this manual is to bring together strategies for the debugging process that do not require prior programming knowledge, to help CS1 students realise different strategies can be applied throughout the debugging process.

The strategies for the manual were identified using a literature review and collated into sections based on similar goals. The manual's presentation was influenced by a debugging model [5], demonstrating to students how these strategies can be applied during the debugging process. A pilot study was conducted to evaluate the manual, which seeks to answer the following research questions:

- **RQ1:** *How does the manual support CS1 students during the debugging process?*
- **RQ2:** *What strategies do CS1 students prefer in the manual?*

To answer the research questions, the manual was given to students as a debugging intervention for a CS1 programming activity. Students were encouraged to reference the manual during the activity before seeking outside assistance. After completing the programming activity, students completed a survey to provide feedback on the manual. The results show students wanting worked examples to practice the strategies presented in the manual. The results also show the manual seems to be supporting Self-Regulated Learning (SRL) strategies, helping students to recall strategies they previously applied and to introduce them to new strategies. The paper concludes with presenting our future plans to improve the manual, which includes incorporating students' feedback and improving the manual's presentation to better demonstrate a debugging workflow that students can adopt for future debugging situations.

II. Background

The debugging process involves understanding the program, testing the program for correct behaviour, locating the issues causing the problems, and rewriting the code to repair the issues [5]. CS1 students find debugging programs a challenge, because they sometimes have difficulty locating runtime issues [6] and finding these issues within the source code [7]. Students get discouraged during the debugging process, because it takes time and effort to find and fix these issues [8]. Students' fragile programming knowledge can contribute to their struggles during the debugging process [9], sometimes resulting in them solving problems with their existing internal knowledge and generalised problem-solving strategies, rather than using instructional materials to help them form new strategies and knowledge [10].

Programming knowledge has been classified into three knowledge types: syntactic, conceptual, and strategic knowledge [11]. Syntactic knowledge is the understanding of the language basic rules for creating a program, such as defining strings and variables. Conceptual knowledge is the understanding of how the programming constructs work together to form a working program. Strategic knowledge brings syntactic and conceptual knowledge together to solve programming problems. "Planning, writing, and debugging programs including tracing or explaining code require strategic knowledge, in addition to knowing syntax and concepts of a programming language." [12, p. 4]. Experts have a strong understanding of syntactic, conceptual, and strategic knowledge, which helps them to successful perform these programming tasks, such as planning and debugging programs. However, novices find these tasks to be a challenge, since they are still learning and gaining syntactic, conceptual, and strategic knowledge [13].

To compensate for their fragile programming knowledge, CS1 students might apply Poor Learning Tendencies (PLTs) to solve programming problems [14]. PLTs are impulsive actions performed

by the student that lack reflective thinking. Students might focus on surface aspects of the problem, such as performing lexical analysis of the problem rather than trying to understand the problem at the semantic level [15]. PLTs can negatively influence students' problem solving, which can sometimes introduce more issues to the source code [16]. Instructional materials demonstrating effective strategies can positively influence students' debugging techniques [17], potentially reducing PLTs to solve problems.

III. Related Work

Learning tools that support aspects of the debugging process are available to CS1 students. For example, questioning has been applied as a debugging intervention to encourage students to reflectively think about the problem [2]. By applying questioning during the debugging process, students have a better understanding of the code, which helps them identify the issue in the source code. Debugging exercises help students practise locating errors in the source code [18]. The debugging exercises are short and contain defects for students to find and resolve. Students engaging in these exercises were shown to have better debugging skills and spent less time debugging their programs.

To help identify debugging tools that support the learning of debugging knowledge, a teaching framework [19] was adapted. Through the framework, the debugging tools were evaluated against six categories: domain, system, strategic global, strategic local, experience, and iterative. Li et al. evaluated the tools against the framework, where they discovered "several gaps between the knowledge required for debugging and the debugging knowledge taught by these tools" [19, p. 85]. The research showed that the tools did support the use of global strategies, which are context-specific strategies that can be applied without the use a debugging system, such as an Integrated Development Environment (IDE). The support of global strategies allows the students to transfer this debugging knowledge to other programming languages and problem-solving situations.

Visualisation tools have been designed to support CS1 students' through areas of the debugging process. A review of visualisation systems has been previously performed [20], so we highlight a few to represent the aspects of the debugging process they support. JIVE [21] is a plugin for the Eclipse IDE that provides students with a visual representation of the runtime process. JIVE is an interactive tool that allows students to step through their programs at runtime to visualise the program's state. Online Python Tutor [22] is another visualisation tool that provides runtime support to step through program execution. In addition to the visualisation, the Online Python Tutor replaces the error messages generated by the Python interpreter with beginner-friendly error messages. Students found the beginner-friendly messages helpful in understanding the issue. The benefits of scaffolded error messages have also benefited compiled languages [23]. When students were presented with shortened, beginner-friendly compiler error messages with resolution hints, the debugging times for the students were reduced, demonstrating that scaffolded error messages support repairing errors.

To make debugging engaging for visual learners, a video repository was developed, presenting content focused on finding and fixing issues commonly encountered by CS1 students [24]. The repository provides a scaffolded environment to help students find a successful solution, since the use of the internet that can produce results containing explanations above the students' programming abilities. The short videos focus on particular errors CS1 students might encounter when programming in Java, such as casting, null pointer exception, and out of scope variable errors. These videos provide the solutions for finding and fixing the errors within the Java language construct. Although the debugging tools presented in this section are designed to support particular aspects of the debugging process, helping students recognise that the debugging process involves different strategies might help them to develop strategies for all aspects of the the debugging process.

IV. Read the Debug Manual

This section describes constructing the *Read the Debugging Manual* (RTDM) as a debugging intervention. Section IV-A describes how the manual was created, reviewing literature to collect best practices for CS1 students. Section IV-B presents the resulting manual, bringing together the literature in a meaningful presentation to help students through the debugging process.

A. Manual Development

A scoping review was performed to identify debugging strategies appropriate for CS1 students, mapping literature to particular topics [25]. For the scoping topics, we selected best practices applied by novice programmers that helped them successfully debug their programs. Murphy et al. [17] identified ten good debugging practices, but for the scoping topics, we selected five practices that do not require prior programming knowledge. The selected five strategies are tracing, testing, understanding the code, isolating the problem, and gaining domain knowledge. Within the search criteria, we applied an inclusion criteria that helped identify appropriate strategies for compiled languages, since we piloted the manual in a CS1 course using C++.

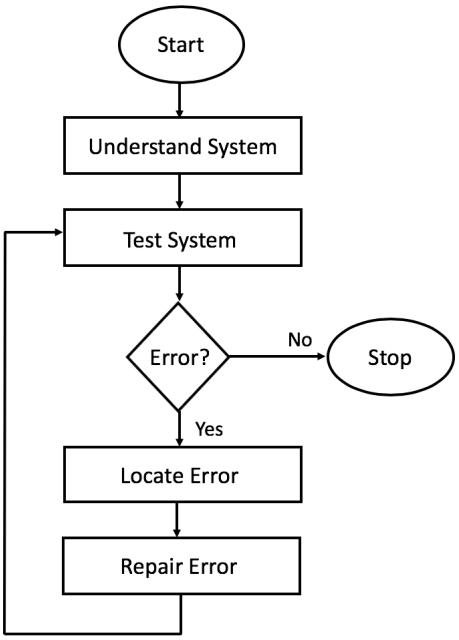

Fig. 1. Debugging Model by Katz and Anderson [5]

In addition to identifying debugging strategies requiring little to no programming knowledge, we also wanted students to recognise that negative emotions, such as frustration, can lead to poor performance [17]. To help students realise their emotions during programming [26], the manual included information on emotional awareness, defined as a person's ability to describe and understand feelings that arise [27]. Helping students recognise their negative emotions can positively influence their demeanor during the problem-solving process, which has been show to lead to higher learning [28]. A

higher degree of emotional awareness can elevate a person's Emotion Regulation (ER), "the processes by which individuals influence which emotion they have, when they have them, and how they experience and express these emotions" [29, p. 275], and positively influence their problem solving [30]. By bringing in emotional awareness into the manual, we wanted to address the negative emotions students experience during software development [31].

After identifying strategies, we created sections by collating strategies with similar goals. After the collation process, we used a debugging model [5], shown in Figure 1, to organise the sections to form the manual's presentation. Sections focusing on particular areas of the debugging process were arranged together, potentially demonstrating to students a debugging process to apply these strategies. Figure 1 shows a generalised debugging approach comprised of four states: *Understand System*, *Test System*, *Locate Error*, and *Repair Error*. *Understand System* is the initial debugging state, where the person applies strategies to better understand the program. The *Test System* state uses strategies to determine if the program is operating as expected. The *Locate Error* state involves strategies to help the person find the error in the program. The *Repair Error* state occurs when the person attempts to fix the error by altering code. Within the model are transitions showing how the person can return to prior states to reapply strategies.

B. Manual Results

The resulting manual contained six sections, made available to students as a PDF. The PDF format gave students the flexibility to download or view the document online. The information was presented in a conversational tone, removing technical jargon and verbosity that might be above the students' comprehension. The manual presents any procedural steps in bullet point, guiding the students through the strategy processes addressing their debugging concerns [32]. An appendix was also provided at the end of the manual, to provide students with more information and examples. The appendix was referenced at the end of each section to let the students know more information was available at the end of the manual.

The six sections supported three debugging states: *Understand System*, *Test System*, and *Locate Error*. No strategies supported the *Repair Error* debugging state. We describe the manual's sections with respect to the debugging states they support. Due to paper length restrictions, we do not present all the strategies within the sections, but exemplars to demonstrate the sections' goals.

1) Understand System: Two sections support the *Understand System* state, promoting a better understanding of the problem and preparing students for locating issues. The literature [33] suggests that students are more successful debugging when they comprehend the problem. The first section presents strategies that promote program comprehension, helping the student develop a good mental model of the problem [34]. A good mental model can mitigate issues during implementation [3]. Programming planning was presented in this section, where students were encouraged to plan using pseudocode. Pseudocode is a description of program code using natural and programming languages [35]. When students use pseudocode, they were more accurate in developing their solutions [36]. The second section is designed to help students develop strategies to reduce frustration that can negatively influence their problem-solving process. The section suggests that students take breaks to help them make better use of their time [37]. Students might realise managing their time can help regulate their frustration, which can change the learning situation and reduce stressors [28].

2) Test System: One section in the manual promotes testing. Boundary testing is a strategy presented in this section. The section describes how to apply boundary testing to confirm the program's output to ensure correct behaviour [17], giving students examples on correctly applying extreme inputs to ensure the program behaves as expected. This section also encourages writing tests to incorporate more complete testing into the debugging process over a small test set which might be incomplete in identifying program issues.

3) Locate Error: Three sections support the *Locate Error* state. The first section presents strategies to help with compile-time issues. This section included a couple of common errors CS1 students might encounter [33] and instructions on how to read these errors[38]. The section noted to the students that the remaining common compiler errors were available in the appendix. The second section presents strategies to help locate logic issues. A strategy presented in this section is code tracing, an effective method of isolating issues using "some form of physical annotation on a piece of paper" [39, p. 71]. The code tracing was presented with a C++ exemplar, demonstrating how code tracing can assist in fixing the logic error. The third section supports finding runtime errors. A strategy presented in this section is the use of `print` statements to observe program state [17]. The instructions describe how to place the statements in the source code and interpret the results when the program executes. The section provides tips on avoiding haphazard placement of `print` statements, which can diminish the effectiveness of the statements as a debugging tool.

V. Methodology

This section describes the pilot study, where we apply the manual as a debugging intervention in a programming activity. Section V-A presents the study's context. Section V-B describes collecting and analysing students' feedback on the manual.

A. Context

The pilot study was conducted in an Introductory Object Oriented Programming (OOP) course using C++. The course was offered at the University of Adelaide in Australia with 466 students enrolled in the July 2020 course. The course administered weekly lectures and computer lab sessions. Students were divided into lab sessions containing 45-60 students and were facilitated by 2-3 teaching assistants (TAs). During the two-hour lab sessions, students worked on programming activities to reinforce learning concepts presented in the weekly lectures.

The pilot study was conducted in week 9, where students were learning polymorphism. To reinforce polymorphism, students were given the programming activity Rock, Paper, Scissors [40] to work on during the lab session. The game-based activity was presented with the core objectives incorporated into the problem description to help students focus on the learning objectives while developing a coded solution. Students were given the manual to use as a debugging intervention for the programming activity. The programming activity also contained a design plan document that encouraged students to begin to formulate a plan prior to implementing the solution. The design plan aligned with the *Understand System* debugging state, and was designed to encourage the students apply the first debugging state into their problem-solving processes. During the lab session, TAs encouraged students to use the manual before seeking assistance elsewhere and also served as a reminder to students that the manual contained teaching and learning approaches for solving the problem. Motivation research, such as the MUSIC model [41], has shown explaining to students available teaching and learning approaches can

help them engage in these approaches, potentially promoting more students to use the manual during the lab session. At the end of the lab session, students were asked to complete a survey for optimal recall performance.

B. Data Collection and Analysis

A non-compulsory survey was administered through Google Forms, to collect students' feedback on the manual. The survey was based on other instruments that evaluate users' experiences using a product [42] and students' strategies for debugging tasks [43]. We adjusted questions from the instruments to frame the questions from the students' perspectives using the manual.

The survey contained seven questions. Three questions were Likert scale that asked students how they used the manual and the likelihood they would use it in the future. The remaining four open-text survey questions were used to gather more in-depth information on students' prior debugging strategies (*"What debugging strategies have you used in past programming problems?"*), and their suggestions for improving the manual (*"What changes would you make to the manual?"*).

After students completed the survey, the data was collected for analysis. Any students that did not use the manual were excluded from the study. The responses to three Likert scale questions were imported into Excel for quantitative analysis, using analysis of means. Analysis of means was used to identify the most frequent rating from the students' responses. The four open-text questions were imported into NVivo for qualitative analysis. Thematic content analysis [44] was used to generate common themes on students' usage of the manual and to collate their feedback on improving it. The qualitative analysis process started with a coded framework containing 12 nodes: two nodes to denote positive and negative feedback, six nodes to denote the six sections in the manual, and four nodes to denote the open-text questions that mapped the coded themes to each question.

Inter-rater reliability was used to ensure consistency in the rating system. At the start of the coding process, a co-author coded 10% of the responses, where any emerging themes were added as new nodes to the coding framework. The authors discussed the emerging nodes to ensure high inter-rater reliability by negotiating on the coding criteria performed on the partial coding. The authors agreed on combining two emerging nodes because they represented the same theme, where students recommended more examples. After the authors agreed on the coding criteria, the responses were coded by a co-author. After all the responses were coded, the authors discussed responses that were categorised in the "Unrelated" node, to ensure the responses in this category did not relate to survey questions. During the discussion, one response was moved from "Unrelated" node into the emerging strategy node labelled "Debugger". The coding author was unfamiliar with the response related to an integrated debugger, but a co-author was familiar with the terminology to place in the appropriate node. When the coding process was completed, the themes were extracted from NVivo as a matrix to present the coding frequencies.

VI. Results and Discussion

From the 466 students enrolled in the course, 165 (35.41%) students completed the programming activity and the survey. From the 165 students, 80 (17.17%) reported they did not use the manual, and were excluded from the study. The reported results are from 85 (18.24%) participants that used the manual.

The results are organised in separate sections. Section VI-A presents the strategies the participants used in past problem-solving situations. This gives us insight into any strategies that might overlap with the manual. Section VI-B presents the sections the participants applied while working on the programming activity. Section VI-C provides participants' responses on the manual's usefulness, while Section VI-D provides participants' feedback on improving the manual.

Strategy	Result
Locate Error (64.41%, n=76)	
Reading compiler errors	22.88% (n=27)
Finding logic errors through `print` statements	19.49% (n=23)
Locating runtime errors through code tracing	13.56% (n=16)
Isolating code*	4.24% (n=5)
Using Google to understand the error*	3.39% (n=4)
Using debuggers*	0.85% (n=1)
Unrelated (16.10%, n=19)	
Using Google without specifics*	12.71% (n=15)
Unrelated*	3.39% (n=4)
Test System (11.02%, n=13)	
Testing	11.02% (n=13)
Understand System (8.47%, n=10)	
Discussing solutions with peers	5.93% (n=7)
Being emotional aware	2.54% (n=3)

* Strategy was not presented in the manual.

TABLE I
Prior Strategies Used by Participants

A. Prior Strategies

The 85 participants reported using multiple strategies ($\bar{s}$=1.27) in prior debugging situations. Table I is a grouped frequency distribution table that organises the strategies within the debugging states they support, and arranges the states in descending order. The table contains an additional category, *Unrelated*, to represent responses that did not pertain to the survey question. The majority (64.41%, n=76) of the reported strategies support the *Locate Error* debugging state, which also contained three emerging strategy categories. Emerging strategies showed participants commenting out code (4.39%, n=5) and using debuggers (0.87%, n=1) to find errors in the source code. Some (3.39%, n=4) responses explained how Google was applied to locate errors in the code, for example, one participant stated they would *"look at errors, if can't understand, use Google"*.

For the remaining (12.71%, n=15) Google-related responses, we could not ascertain how students applied the search engine to the debugging process. Example responses include *"Google it"* and *"it's easier to Google what I want"*. Prior research [45] has shown students using Google to support the *Repair Error* debugging state by searching for solutions, but in this study, we could not confirm from the collected data.

B. Applied Manual Sections

Table II is another grouped frequency distribution table that shows the participants' responses to the manual sections they used during the pilot study. The table organises the sections within the three debugging states supported by the manual. The results show participants using multiple sections ($\bar{s}$=2.67). The section on testing (21.15%, n=48) was used the most, but from the participants' feedback, it is unclear why they had a preference for this section.

The emotional awareness section (9.69%, n=22) was used the least. Participants responding on this section felt it *"did not seem entirely relevant"*, yet also stated *"it might seem obvious, but if you keep reminding yourself about these topics in the manual, they actually make people less frustrated to debug and solve their own problems*

Section	Result
Locate Error (52.42%, n=119)	
Locating compile-time errors	18.06% (n=41)
Locating logic errors	17.18% (n=39)
Locating runtime errors	17.18% (n=39)
Understand System (26.43%, n=60)	
Promoting program comprehension	16.74% (n=38)
Raising emotional awareness	9.69% (n=22)
Test System (21.15%, n=38)	
Testing system	21.15% (n=48)

TABLE II
MANUAL SECTIONS USED BY PARTICIPANTS

faster". Perhaps the manual needs to provide explicit instructions on the purpose of this section, helping students to realise emotional awareness can positively influence their problem solving. For example, aspects surrounding emotional awareness could be presented as strategies related to emotional learning, where the student acquires skills to manage emotions [46].

The remaining four sections were equally accessed (16.74-17.18%) by the participants, but when the sections are examined by their supported debugging states, the results show a preference for the *Locate Error* strategies. A potential reason for these results is that half of the manual's sections relate to locating errors. Another potential reason for these results is that the study was conducted during a two-hour lab session, where students might be focused on getting the program to work and spend time later to improve the software design or create additional tests.

C. Perceived Usefulness

Three (3.53%) participants felt they were unable to provide feedback on the manual's usefulness since they *"didn't use it enough to know"*. These three responses are understandable since the participants had one opportunity to use the manual to solve the programming activity. The majority (78.82%, n=67) of the responses found the manual helpful. General positive statements include *"I didn't know much about to begin with so I learnt a lot"*, *"it taught me unique and professional ways to fix problems beyond what I had already known"*, *"provides a good starting point for debugging"*, and has *"new methods to understand to debug"*. Positive comments on the manual's presentation were also provided, for example, *"it goes into good detail about main aspects of the debugging process (how to prevent, find and fix bugs)"* and *"the explanations were clear and beginner-friendly"*.

From the responses on how the participants used the manual, we can observe the manual seems to be supporting two *Self-Regulated Learning* (SRL) strategies. SRL strategies give students active control over their learning, to ensure they achieve their learning goals [47]. There are 14 SRL categories related to student-initiated SRL strategies. The first SRL strategy, *Rehearsing and Memorising*, involves "student-initiated efforts to memorise material by overt or covert practice" [47, p. 618]. The manual presents previously used strategies by the participants, so the presence of their known strategies might support rehearsing and memorising. For example, a participant stated *"while I was familiar with some of the methods already, having the manual to refresh and get some more details was helpful"*, and *"I can come back to the document if I struggle finding a bug"*. The second SRL strategy, *Seeking Information*, involves "student-initiated efforts to secure further task information from nonsocial sources when undertaking an assignment" [47, p. 618]. The manual provides students the opportunity to learn new strategies, so reviewing the manual might introduce them to new strategies. For example, participants stated it *"gave new methods to understand how to debug"*, and it *"made me think about different approaches to debugging"*.

Fifteen (17.65%) participants felt the manual was unhelpful because they were already familiar with the strategies. Example feedback from participants include *"it was pretty much what I already do"*, and *"I already knew most of the techniques it showed"*. It is unclear whether these participants have more advanced programming knowledge and do not require the manual, or if they used the *Recall and Memorising* SRL strategy to confirm their knowledge of the presented strategies. More research is required to better understand the responses claiming the manual was unhelpful.

D. Recommendations for Improvements

Table III shows participants' responses on improving the manual. Their responses formed four themes: *No Changes*, *Content*, *Presentation*, and *Unrelated*. The majority (54.12%, n=46) of the participants did not have suggestions. Example responses included *"I think all good"*, and *"None springs to mind"*. The unrelated theme contained responses (10.59%, n=9) that did not pertain to the survey question.

For content improvements (23.53%, n=20), participants suggested examples to explain advanced problem-solving situations. Participants recommended *"more complex examples"* and *"extend the testing section to include some more examples"*. Though the appendix provided examples, it was unclear from the participants' responses whether they used the appendix. They also wanted worked examples to practice the strategies. For example, a participant suggested *"practice to print more output"*. Worked examples can provide guidance in properly applying strategies, which can reduce students' debugging times [18]. For the manual's presentation (11.76%, n=10), participants suggested presenting the information with less text and more visualisations. Example suggestions include *"make the explanations more succinct"*, *"informative videos"*, and *"more diagrams"*. Adding visualisations can support better learning for novice programmers [48] and an approach that might interest visual learners [24]. Altering the manual to include more visualisations is worth considering for future revisions.

VII. THREATS TO VALIDITY

There are limitations to this study. The study had low student participation (18.24%, n=85) and does not accurately reflect how the cohort might have used the manual as a debugging intervention. Future studies will encourage more participation from students. Another limitation is the narrow scope of the pilot study, where participants had one opportunity to use the manual. Participants recognised the limited use might have influenced their feedback. One participant stated they *"didn't use enough to particularly know, but I think it is helpful to have an available resource"*. Though the pilot study provided insightful feedback for manual revisions, future studies will give students the opportunity to use the manual over multiple programming activities, so that students can provide a more informed feedback and strengthen the findings.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented a study that evaluated a debugging manual, *Read the Debugging Manual* (RTDM), designed for CS1 students. The manual presents different strategies appropriate for CS1 students that can be applied during the debugging process. This study helped us answer the research questions. For RQ1, we observed the manual seemed to support Self-Regulated Learning strategies, such as helping students rehearse and memorise existing problem-solving strategies, along with supporting them to seek information

Feedback	Result	Example Participant Feedback
No Changes	54.12% (n=46)	*"None springs to mind"*, *"Not much, it's perfect"*, *"It's good enough maybe I just have to get use to it"*
Content	23.53% (n=20)	*"More complex examples"*, *"The emotional awareness section did not seem entirely relevant and could be replaced with something else"*, *"Practice to print more output"*
Presentation	11.76% (n=10)	*"I would prefer a video version"*, *"Make the explanations more succinct"*
Unrelated	10.59% (n=9)	*"N/A"*, *"Yes"*

TABLE III
PARTICIPANTS' FEEDBACK ON IMPROVING MANUAL

that introduces them to new strategies. Future research can focus on how the manual supports the application of SRL strategies during the debugging process. By focusing on the SRL strategies, such as *Goal-setting and Planning*, we can frame Self-Regulated Learning as a theoretical lens to analyse and view the data. For RQ2, students applied more strategies related to locating errors, but this may be due to half of the manual's sections focusing on locating errors. Future revisions of the manual can add more sections to the *Understand System* and *Test System* debugging states, providing a uniform representation of strategies across the debugging states. In addition to a uniform representation of the existing debugging states, we are interested in finding strategies to support the *Repair Error* debugging state.

Students also suggested worked examples to practice the strategies presented in the manual, and examples that explain complex debugging situations. Future revisions of the manual can include more examples and present them as exercises so that students have the opportunity to practice these strategies. Future research can evaluate the debugging exercises developed by Chimel and Loui [18] to determine their influence on students' debugging skill development.

Though the manual's presentation was influenced by a debugging model, more work is required to fully realise the influence a debugging model can have on CS1 students' applied strategies. Future revisions of the manual can emphasise the debugging workflow, making the workflow more apparent so that students might adopt the debugging process for future problem-solving situations. Future research can evaluate the influence the model has on students' application of strategies, potentially helping us to better understand their preferences to certain strategies.

REFERENCES

[1] R. Chmiel and M. C. Loui, "An integrated approach to instruction in debugging computer programs," *Proceedings of AEE/IEEE in Education*, vol. 3, pp. 1–6, Nov 2003.

[2] J. D. Wilson, "A socratic approach to helping novice programmers debug programs," in *Proceedings of the Eighteenth SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 179–182.

[3] B. Blunden, *Software Exorcism: A Handbook for Debugging and Optimizing Legacy Code*. Apress, 2003.

[4] T. Grötker, U. Holtmann, H. Keding, and M. Wloka, *The Developer's Guide to Debugging*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[5] I. Katz and J. Anderson, "Debugging: An analysis of bug-location strategies." New York, NY, USA: Taylor Francis, 1989, pp. 84–88.

[6] P. Kinnunen and L. Malmi, "Cs minors in a CS1 course," in *Proceedings of the Fourth International Workshop on Computing Education Research*, ser. ICER '08. Association for Computing Machinery, 2008, pp. 79–90.

[7] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, L. T. Beth Simon, and C. Zander, "Debugging: Finding, fixing and failing, a multi-institutional study of novice debuggers," *Computer Science Education*, vol. 18, pp. 93–116, 2008.

[8] P. Kinnunen and L. Malmi, "Why students drop out CS1 course?" in *Proceedings of the Second International Workshop on Computing Education Research*, ser. ICER '06. New York, NY, USA: Association for Computing Machinery, 2006, pp. 97–108.

[9] D. Perkins and F. Martin, "Fragile knowledge and neglected strategies in novice programmers," *Papers presented at the first workshop on empirical studies of programmers*, pp. 213–229, October 1985.

[10] I. Roll, V. Aleven, B. M. Mclaren, and K. R. Koedinger, "Designing for metacognition—applying cognitive tutor principles to the tutoring of help seeking," *Metacognition and Learning*, vol. 2, pp. 125–140, December 2007.

[11] T. J. McGill and S. E. Volet, "A conceptual framework for analyzing students' knowledge of programming," *Journal of Research on Computing in Education*, vol. 29, no. 3, pp. 276–297, 1997.

[12] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, Oct. 2017.

[13] J. L. Whalley, R. Lister, E. Thompson, T. Clear, P. Robbins, P. K. A. Kumar, and C. Prasad, "An australasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies," in *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ser. ACE '06. AUS: Australian Computer Society, Inc., 2006, pp. 243–252.

[14] J. R. Baird and J. R. Northfield, *Learning from the PEEL experience*. Melbourne: Monash University, 1992.

[15] B. Adelson, "When novices surpass experts: The difficulty of a task may increase with expertise," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 10, pp. 483–495, 1984.

[16] B. S. Alqadi and J. I. Maletic, "An empirical study of debugging patterns among novices programmers," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 15–20.

[17] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander, "Debugging: The good, the bad, and the quirky – a qualitative analysis of novices' strategies," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '08. Association for Computing Machinery, 2008, pp. 163–167.

[18] R. Chmiel and M. C. Loui, "Debugging: From novice to expert," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '04, vol. 36, no. 1. New York, NY, USA: Association for Computing Machinery, 2004, pp. 17–21.

[19] C. Li, E. Chan, P. Denny, A. Luxton-Reilly, and E. Tempero, "Towards a framework for teaching debugging," in *Proceedings of the Twenty-First Australasian Computing Education Conference*, ser. ACE '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 79–86.

[20] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Transactions on Computing Education*, vol. 13, pp. 1–64, 10 2013.

[21] J. K. Czyz and B. Jayaraman, "Declarative and visual debugging in eclipse," in *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology EXchange*, ser. eclipse '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 31–35.

[22] P. J. Guo, "Online python tutor: Embedded web-based program visualization for cs education," *SIGCSE '13*, Mar 2013.

[23] P. Denny, J. Prather, and B. A. Becker, "Error message readability and novice debugging performance," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '20. Association for Computing Machinery, 2020, pp. 480–486.

[24] B. Simon, S. Fitzgerald, R. McCauley, S. Haller, J. Hamer, B. Hanks, M. Helmick, J. E. Moström, J. Sheard, and L. Thomas, "Debugging assistance for novices: A video repository," in *Working Group Reports on ITiCSE on Innovation technology in Computer Science Education*, 2007, pp. 137–151.

[25] H. Arksey and L. O'Malley, "Scoping studies: Towards a methodological framework," *International Journal of Social Research Methodology*, vol. 8, no. 1, pp. 19–32, 2005.

[26] A. Fountaine and B. Sharif, "Emotional awareness in software development: Theory and measurement," in *Proceedings of the 2nd International Workshop on Emotion Awareness in Software Engineering*, ser. SEmotion '17, 2017, pp. 28–31.

[27] M. Boden and R. Thompson, "Facets of emotional awareness and associations with emotion regulation and depression," in *Emotion*, vol. 15, 2015, pp. 399–410.

[28] B. J. Zimmerman and A. Kitsantas, "Acquiring writing revision and self regulatory skill through observation and emulation," *Journal of Educational Psychology*, vol. 94, no. 4, pp. 660–668, 2002.

[29] J. J. Gross, "The emerging field of emotion regulation: An integrative review," vol. 2, 1998, pp. 271–299.

[30] C. Izard, E. Cavadel, K. Finlon, E. S. K. Ewing, S. Johnson, and A. Seidenfeld, "Emotion knowledge, emotion utilization, and emotion regulation," *Emotion Review*, vol. 3, no. 1, pp. 44–52, 01 2011.

[31] P. Kinnunen and B. Simon, "Experiencing programming assignments in cs1: The emotional toll," pp. 77–85, 2010.

[32] J. M. Carroll, P. L. Smith-Kerker, J. R. Ford, and S. A. Mazur-Rimetz, "The minimal manual," *Hum.-Comput. Interact.*, vol. 3, no. 2, p. 123–153, Jun. 1987.

[33] R. McCauley, S. Fitzgerald, G. Lewandowski, L. Murphy, B. Simon, L. Thomas, and C. Zander, "Debugging: A review of the literature from an educational perspective," *Computer Science Education*, vol. 18, no. 2, pp. 67–92, 2008.

[34] C. Schulte, T. Busjahn, T. Clear, J. H. Paterson, and A. Taherkhani, "An introduction to program comprehension for computer science educators," *ITiCSE '10: Proceedings of the 2010 ITiCSE Working Group*, pp. 65–86, 6 2010.

[35] S. Garner, "A program design tool to help novices learn programming," *ASCILITE '07: Proceedings of Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education*, pp. 321–324, 2007.

[36] H. A. Ramadhan, "Programming by discovery," *Journal of Computer Assisted Learning*, vol. 16, pp. 83–93, 2000.

[37] G. Lewandowski, "Using process journals to gain qualitative understanding of beginning programmers," *J. Comput. Sci. Coll.*, vol. 19, no. 1, pp. 299–310, Oct. 2003.

[38] B. A. Becker, "An effective approach to enhancing compiler error messages," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 126–131.

[39] S. Fitzgerald, B. Simon, and L. Thomas, "Strategies that students use to trace code: An analysis based in grounded theory," in *Proceedings of the First International Workshop on Computing Education Research*, ser. ICER '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 69–80.

[40] C. R. Hardnett, *Programming Like a Pro for Teens*, 1st ed. Boston, MA, USA: Course Technology Press, 2011.

[41] B. D. Jones, "Motivating students to engage in learning: The music model of academic motivation," *International Journal of Teaching and Learning in Higher Education*, vol. 21, no. 2, pp. 272–285, Nov 2009.

[42] A. P. O. S. Vermeeren, E. L.-C. Law, V. Roto, M. Obrist, J. Hoonhout, and K. Väänänen-Vainio-Mattila, "User experience evaluation methods: Current state and development needs," in *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, ser. NordiCHI '10, 2010, pp. 521–530.

[43] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, and C. Zander, "Debugging from the student perspective," *IEEE Transactions on Education*, vol. 53, no. 3, pp. 390–396, 2010.

[44] J. W. Creswell and V. L. P. Clark, *Designing and Conducting Mixed Methods Research*. SAGE Publications, 2006, ch. 4, pp. 58–88.

[45] B. Hanks and M. Brandt, "Successful and unsuccessful problem solving approaches of novice programmers," in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '09. Association for Computing Machinery, 2009, pp. 24–28.

[46] M. Arguedas, T. Daradoumis, and F. Xhafa, "Analyzing the effects of emotion management on time and self-management in computer-based learning," *Computers in Human Behavior*, vol. 63, pp. 517–529, 2016.

[47] B. J. Zimmerman and M. M. Pons, "Development of a structured interview for assessing student use of self-regulated learning strategies," *American Educational Research Journal*, vol. 23, no. 4, pp. 614–628, 1986.

[48] T. Ahoniemi and E. Lahtinen, "Visualizations in preparing for programming exercise sessions," *Electronic Notes in Theoretical Computer Science*, vol. 178, pp. 137–144, 2007.