A Debugging Method Support System Using Past Error Information in Programming Exercises with the Same Teaching Materials

Keiichi Takahashi

Humanity-Oriented Science and Engineering, Kindai University, Iizuka, Japan
Email: ktakahas@fuk.kindai.ac.jp (K.T.)
Manuscript received March 10, 2025; revised April 3, 2025; accepted May 16, 2025; published August 7, 2025

***Abstract*—This study proposes a Debugging Method Suggestion System (DMSS) using past error information to address the difficulties faced by learners when solving bugs in programming-learning environments. The DMSS automatically collects error information generated when students work on assignments, allowing instructors to add annotations describing the causes and solutions for these errors. When other students using the same programming materials encounter similar errors, the system presents them with the accumulated similar error information and advice. This system was implemented to assist students in developing a learning web application using the Ruby on Rails framework. Its effectiveness was verified by an experiment involving 53 university students. The results showed that the debugging success rate when using the DMSS for unknown errors was more than twice that of cases where DMSS was not used, with particularly significant support effects for first-time errors. However, differences in effectiveness based on error difficulty and prior knowledge were observed, suggesting that content and quality of advice are more important for complex errors.**

***Keywords*—programming education, debugging support system, error information, ruby on rails, web application development, educational technology, collaborative learning**

I. Introduction

In recent years, programming skills have become increasingly important. It is recognized as a means to develop software and as a skill that contributes to problem-solving abilities and logical thinking development [1, 2]. Programming education has been widely incorporated from elementary to higher education and even into adult education [3, 4].

Despite the expansion of programming education, the learning difficulties faced by beginners remain a significant challenge [5, 6]. A major factor is bugs (program defects) caused by input errors or logical mistakes [7]. While these bugs are unavoidable in the learning process, they can be substantial barriers for beginners [8]. In particular, learners who lack the ability to identify, diagnose, and fix errors often lose motivation and abandon programming when faced with bugs [9, 10].

Several common features explain why beginners struggle with bug fixation. First, they are unfamiliar with the syntax and basic structure of programming languages, which makes it difficult to interpret error messages correctly. These messages, designed for language designers and developers, often contain advanced technical terms and abstract expressions. Consequently, beginners frequently struggle to understand error messages and find appropriate solutions [11–13]. Second, beginners typically encounter syntax, type, and logical errors while learning. However, they often struggle to identify the specific causes of these errors, leading to time-consuming troubleshooting. In particular, when errors in one part affect other parts of the program, beginners find it difficult to identify the root cause [14–16]. Such repeated situations hinder learning progress and cause significant frustration. Third, feedback during the error correction process is generally insufficient for beginners. Traditional programming education typically involves the direct support of instructors. However, individual instruction opportunities have decreased with the spread of online education and diversification of learners. Consequently, learners are isolated when fixing errors, thus, impeding efficient learning [17, 18].

To address these challenges various technical approaches for supporting programming learning have been proposed. For instance, many online programming platforms provide features that analyze learner-written code in real time and immediately identify errors, allowing for early detection and on-the-spot correction [19, 20]. Systems that analyze learners' coding patterns and advise them on predicted errors and correction methods have also been developed [21]. In addition, research is progressing regarding tools that translate error messages into beginner-friendly explanations. Recently, systems using Large Language Models (LLMs) have been proposed to provide beginners with natural language error explanations and code completion [22, 23]. The generative capabilities of LLMs offer clear explanations of typical errors and provide appropriate correction suggestions [24, 25]. Attempts are also being made to analyze learners' progress in real time and provide feedback based on individual weaknesses [26], which are expected to help beginners better understand their errors and improve their learning efficiency.

Although many technical approaches to programming learning debugging support systems have been proposed, customization according to specific teaching materials is necessary to use these research outcomes in an educational environment [27–29]. Customization requires information on programming learning materials and potential exercise data from students who have used these materials. Preparing such information is time-consuming for users of research outcomes. Additionally, programming classes often use the same material repeatedly. Automatic collection and sharing of error information from students working with these materials can minimize the customization costs for classroom support tools.

This study proposes a debugging support system based on past error information generated when identical

programming materials are used. Specifically, it automatically collects error information generated while students work on assignments, and instructors annotate this information using error causes and debugging methods. When other students using the same programming materials encounter errors, the system presents the most similar past error information, encouraging the self-resolution of the bugs. This study used Ruby on Rails (Rails) as the programming environment [30]. It is a leading web application development framework that adopts the model-view-controller architecture and requires multiple files to implement a single function. This environment was selected to verify the effectiveness of error information debugging in a realistic application-development setting. As Rails has influenced various frameworks currently in use, we expect the findings of this research to be applicable to other frameworks.

II. Proposed System

A. Proposed System Schematic

We propose a system that automatically collects the error information encountered by students when working on programming assignments and provides debugging advice. Fig. 1 shows the configuration of the system. The Error Information Collector (EIC) automatically collects error information when Student A works on assignments based on programming materials (Fig. 1-(1)). When the EIC detects an error, it sends related information, including the user ID, IP address, error messages, and files updated by the student to the Debugging Method Suggestion System (DMSS). This information is extracted from the log files of the web application server and transmitted after serialization.

The DMSS stores the received information in a database called the Debug Database (DD). This error information is referenced and annotated with the cause of the error and solution methods (Fig. 1-(2)). When Student B encounters an error, the system extracts the most similar error from the DD and presents its advice to Student B (Fig. 1-(3)). This tool is called the Debugging Method Suggester (DMS) [31].

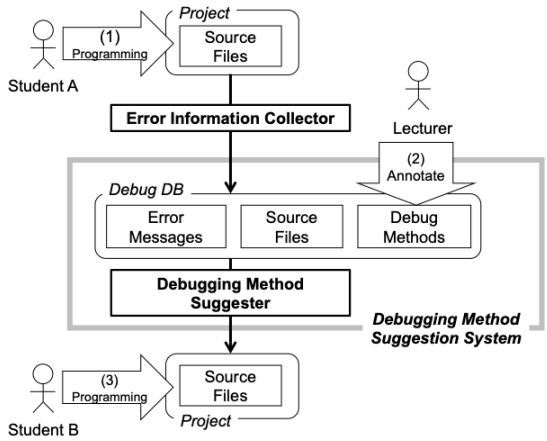

Fig. 1. Overall schematic of the DMSS.

B. Typical Use-Case

This section presents a typical DMSS use-case and explains the functions and roles of the DMSS user interface. Fig. 2 presents the use-case scenario flow, which consists of eight scenarios (S1 to S8). The operational procedures and screens of the DMSS are explained according to this scenario.

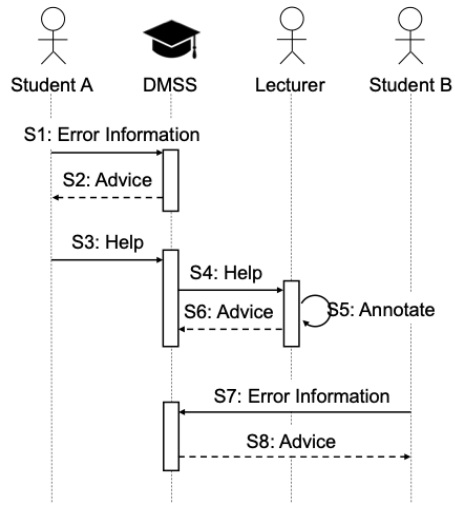

Fig. 2. Flowchart of a typical use-case scenario for the DMSS.

First, Student A creates a program for an assignment (Fig. 1-(1)). As the program is a web application, it is accessed and executed via a web browser. Fig. 3 illustrates an example of an execution screen. In this example, there is no error in the program entered by the student. However, there is an error because there is no table intended for use in the application. The cause of this is forgetting to execute the procedure to create the table.

Fig. 3. Example of an error page owing to forgotten procedure.

S1 and S2: If Student A can solve the problem independently after seeing the error message, they debug it without using the DMSS. If they cannot understand the problem, they use the DMSS (Fig. 4). Fig. 4 shows the user ID, error occurrence time, and error message collected by the EIC. The text "No advice on the error yet" indicates that the error information is not registered in the DD.

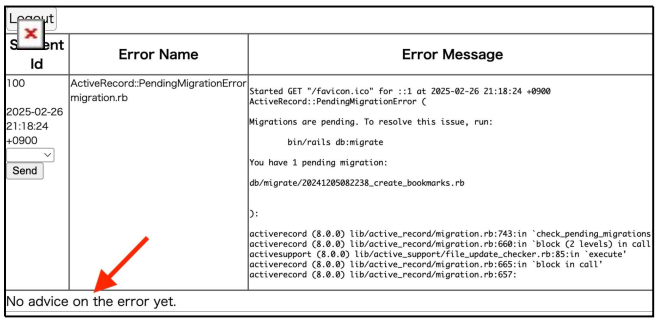

Fig. 4. Webpage where Student A checks for advice after encountering an error in the DMSS.

S3: The student informs the instructor that advice is

required to debug the error by selecting "HELP" from the menu and press the Send button (Fig. 5).

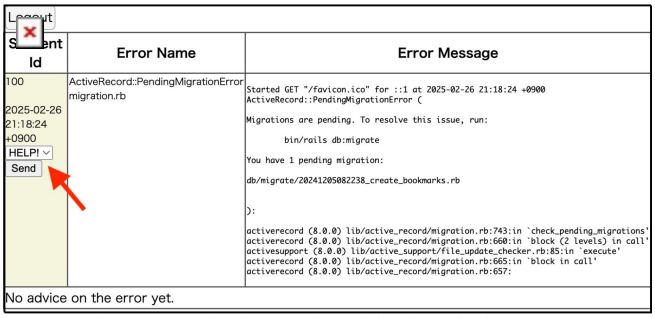

Fig. 5. Example of requesting advice from the instructor.

S4: When the instructor accesses the DMSS, a notification appears at the top of the screen, indicating that the student has requested support (Fig. 6). This notification appears in real time during class or when the instructor logs into the DMSS outside class hours. The notification does not disappear until it is clicked, thereby preventing oversight. When the instructor clicks on the student ID, the detailed error information is displayed.

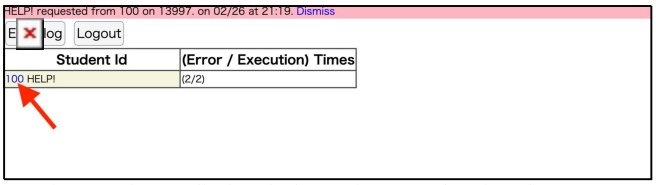

Fig. 6. Web page displayed when an instructor logs into the DMSS.

S5: Fig. 7 shows the detailed error information. This screen is similar to the screen seen by Student A (Fig. 4); however, the instructor's screen allows the deletion and edit of erroneous information. The instructor clicks on the edit icon to write advice.

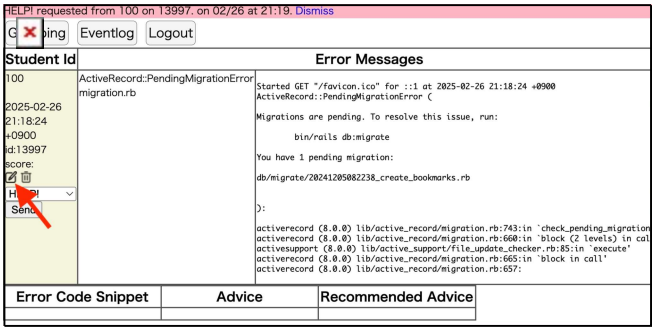

Fig. 7. Example of instructor operation when annotating advice in DMSS.

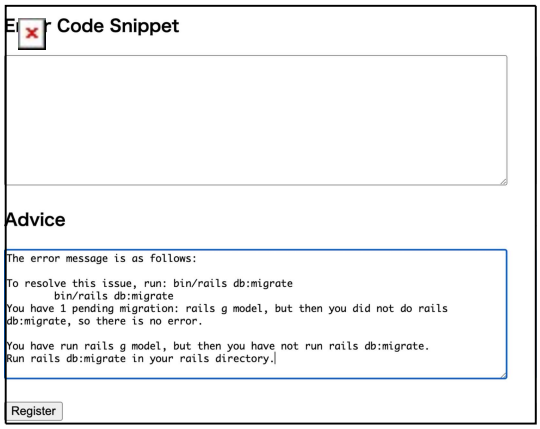

Fig. 8. Example of the instructor's advice on the web page.

Fig. 8 shows the screen on which the instructor enters the advice. They can view the related code to identify the cause of the error. We omitted this explanation here to save space. The instructor enters the cause of the error and investigation methods into the advice field. When required, the instructor can paste part of the source code into the Error Code Snippet field to explain the cause to students. In this case, there is no error in the source code; therefore, it is left blank.

S6: When the instructor finishes entering the advice, a notification appears automatically at the top of Student A's screen, allowing the student to check it (Fig. 9).

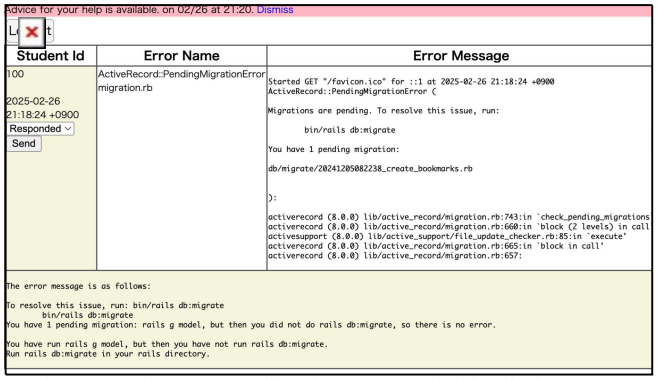

Fig. 9. Example web page of Student A when checking the instructor's response.

S7: Suppose Student B makes the same mistake as Student A. The error information encountered by Student B is automatically sent to the DMSS by the EIC.

S8: As advice regarding that error is registered in the DD, when Student B accesses the DMSS, the same advice is displayed (Fig. 10). Therefore, they can fix the program according to the correct advice.

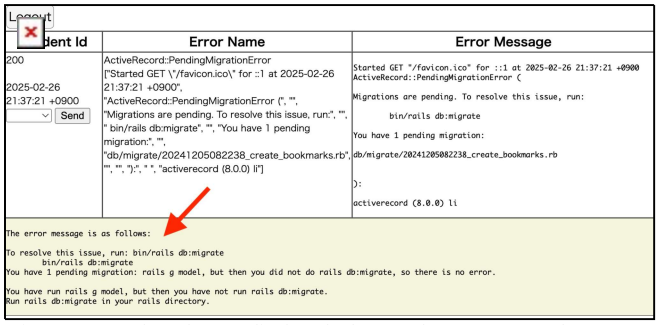

Fig. 10. Example web page displayed when Student B accesses the DMSS.

III. Experiments

A. Experimental Method

The aim of this experiment was to determine whether advice presented to students through the DMSS could effectively improve their debugging success rates. As the DMSS is intended to fulfill the role of an instructor providing individual guidance to students, the advice prepared in advance for the DMSS mirrored the natural guidance that instructors would provide during class. Additionally, as some students might not require assistance, the use of the DMSS was made optional. The experimental methodology is described below.

The Department of Information and Computer Science at Kindai University offers a course on Rails programming to third-year students. This course consists of 15 sessions, each

lasting 180 min. The present experiment was conducted during the 15th session. Therefore, the participants had acquired the basics of Rails programming before the experiment. Among the course attendees, 53 students consented to participate in this experiment.

Even when using the same teaching materials, the types and locations of programming input errors varied among the students. As studying the corrections for different errors would make it difficult to compare the results, programs containing deliberately inserted errors were prepared to compare the possibility of error correction and effectiveness of advice under identical conditions. The debugging tasks assigned to the students consisted of four problems, labeled Ex1, Ex2, Ex3, and Ex4 (Table 1). These problems were selected based on the instructor's experience with frequently asked questions from students who had previously required guidance. These errors were either "forgetting one procedure" or "typos of 2–3 characters in the program."

Table 1. Test descriptions of debugging problems used in the experiment

Experiment ID	Test Description
Ex1	After executing the "rails generate" command to create a controller and a model, verify that the top page is displayed correctly.
Ex2	After implementing the feature to list bookmarks retrieved from the database, verify that the registered bookmarks are correctly displayed on the page.
Ex3	After implementing the feature to enter a new bookmark, verify that the entry page is displayed correctly.
Ex4	After entering bookmark information on the entry page, verify that the data are successfully stored in the database and displayed on the list page.

B. Experimental Procedure

The Rails application that the participants debugged in the experiment was a small application that managed bookmark data. It saves the titles and URL of bookmarks entered by users into a database and displays a list of these bookmarks. All participants had experience creating this application two weeks before the experiment. Thus, the creation procedure was known and some participants may have already experienced errors.

For the experiment, an application containing bugs was downloaded to the participants' computers using Git, a version of the control software. Each problem (Ex1–Ex4) could be instantly switched using the Git "checkout" command. After switching to a problem, the participants had 10 min to verify the operation of each problem's test description. These were debugged by examining the displayed error messages and programs. The participants could decide whether to use the DMSS during debugging. After 10 min, they answered a questionnaire prepared in Google Forms within 5 min and then switched to the next problem to prepare for the experiment. In this experiment, the participants' use of the DMSS corresponded to operations S7 and S8, as described in Section II.B.

The following questionnaire was administered to the students after they completed debugging tasks Ex1–Ex4:

Q1. Was the debugging process successful? (Yes/No) Respondents who answered "Yes" to Q1 only:

Q2. Please provide a detailed description of the cause of the error.

Q3. Have you encountered similar errors in the past? (Yes/No)

Q4. Did you use the DMSS? (Yes/No)

Q5. Was the advice provided by the DMSS helpful? (Yes/No)

C. Instructor-Annotated Advice

From an instructor's perspective, the goal is to encourage students to understand how Rails works and help them to identify errors by following Rails' execution flow. Therefore, from an educational perspective, instructors do not directly indicate the program correction points or methods but encourage students to verify whether the execution flow is functioning correctly.

Table 2 lists the advice annotated for each problem. The underlines indicate the parts of each advice that should be noted. The advice for Ex1 shows specific procedures and executing these procedures allows for bug fixing. Contrarily, the advice for Ex2–Ex4 indicate locations to check, but do not include correction methods. This implies that bug fixing based on advice alone is difficult for such problems.

Table 2. Examples of advice for debugging problems used in the experiment

Experiment ID	Advice
Ex1	You have run rails g model, but then you have not run "rails db:migrate". *Run "rails db:migrate" in your rails directory.*
Ex2	The error message is "ActionController::RoutingError (uninitialized constant BookmarkController)." The term "Bookmark" is used in the error message, whereas it should be "Bookmarks." The system appears to be attempting to access "Bookmark" instead of "Bookmarks," which suggests a mismatch. Given that the error is a RoutingError, it is likely that there is an issue in the routing configuration (config/routes.rb) causing this problem. *Please verify whether any instance of bookmark exists where it should instead be bookmarks in the routes.rb file.*
Ex3	The error message is "ActionView::Template::Error (First argument in form cannot contain nil or be empty)." 1: <%= form_for @bookmark, url: {action: :create} do |f| %> 2: <%= f.label :title, 'title' %> 3: <%= f.text_field :title %> 4: <%= f.label :url, 'URL' %> app/views/bookmarks/new.html.erb:1. The error message said, "The first argument in form_for cannot contain nil or be empty". This means that the value of @bookmark is nil. Where do you set the value of @bookmark? *Please check the value of @bookmark. Even if it is set, if the variable name is different, it cannot be passed on and may become nil, so please check.*
Ex4	The error message is "NoMethodError (undefined method '[]' for nil:NilClass): app/controllers/bookmarks_controller.rb:11:in create." This indicates that an error occurs on line 11 of the bookmarks_controller.rb file. The relevant line of code is shown below: 011: bookmark = Bookmark.new(title: params[:bookmarks][:title], url: params[:bookmarks][:url]) The error suggests that params[:bookmarks] is nil, resulting in a failure when attempting to access [:title]. This implies that params[:bookmarks] is empty or not properly initialized. *Investigate why params[:bookmarks] is empty to identify the root cause of the issue.*

IV. Results

Fig. 11 shows the debugging success rate and DMSS usage rate for each problem. Participants were considered successful at debugging only if they answer Yes to Q1 and provided the correct reason in Q2 in the questionnaire. The debugging success rate calculated from all experimental data was 64%. In Ex1 and Ex2, more than 80% of participants successfully debugged their problems. However, the

debugging success rate for Ex3 was less than 30%. This indicates that debugging difficulty varies according to a problem.

The DMSS usage rate calculated from the experimental data was 69%. That is, the percentage of participants who answered Yes to Q4. The DMSS usage rate was approximately 70% for all problems. The participants used the DMSS regardless of the problem difficulty.

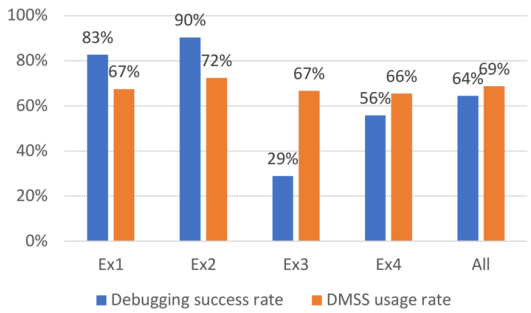

Fig. 11. Comparison of debugging success rates and DMSS usage rates across problems, showing consistent approximately 70% usage despite varying success rates.

Fig. 12 shows the percentage of debugging problems already known to the students. Problems Ex1–Ex4 were selected based on the instructor's subjective judgment. These data validate the appropriateness of subjective selection. The results indicated that Ex1 and Ex2 were errors that many students had previously experienced. In contrast, Ex3 and Ex4 were errors that most students were encountering for the first time.

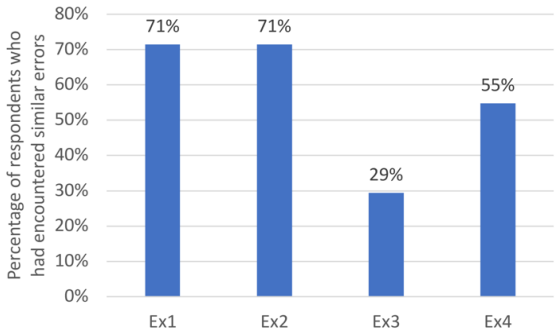

Fig. 12. Prior experience with similar errors by problem (Q3), revealing Ex1 and Ex2 were familiar to many participants while other problems were generally novel.

Table 3 shows the debugging success rates tabulated based on the responses to Q3 and Q4. When the bug was already known, the debugging success rate was very high at 96% even without using DMSS. However, when the bug was unknown, the use of the DMSS doubled the debugging success rate ("All" row, Table 3). The success rates for the individual problems are analyzed in Section V.

Table 3. Debugging success rates tabulated based on responses to Q3 and Q4 of the questionnaire

Experiment ID	Known Error		Unknown Error	
	With DMSS	*Without DMSS*	*With DMSS*	*Without DMSS*
All	81%	96%	61%	28%
Ex1	91%	92%	89%	38%
Ex2	96%	100%	91%	50%
Ex3	50%	NA[1]	29%	20%
Ex4	65%	100%	57%	27%

[1] Not Applicable

Fig. 13 presents the results to Q5. For Ex1 and Ex2, the rate was nearly 100%, indicating that the advice presented by the DMSS was helpful for debugging. For Ex4, the rate was 84%, suggesting that the advice was helpful for students who understood it. By contrast, the response rate for Ex3 was 63%. The advice recommended checking the execution flow of the program, because the initial values of the program variables were not set. However, Ex3 pertained to forgetting to input the method, which set the initial value of the variable. For students who did not understand Rails' execution flow, it was likely difficult to identify the cause, even after receiving the advice.

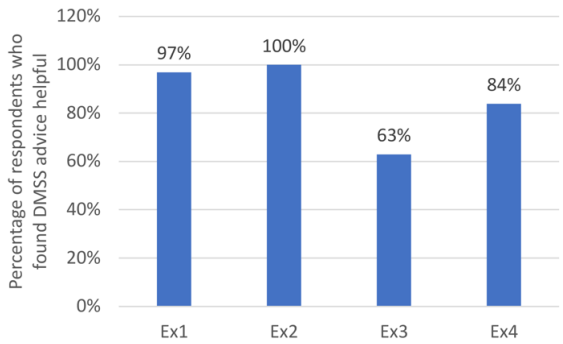

Fig. 13. Perceived helpfulness of DMSS advice by problem (Q5), with notably lower effectiveness for Ex3 compared to other problems.

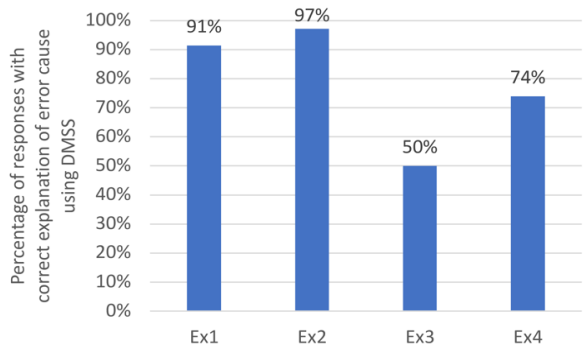

Fig. 14. Correct error cause explanation rates when using DMSS (Q2, Q4), showing lower understanding for Ex3 corresponding to its reduced perceived helpfulness in Fig. 13.

Fig. 14 displays the percentage of participants who used the DMSS and correctly explained the cause of the error. These data were obtained from questionnaire items Q2 and Q4. The results showed nearly the same trend as Fig. 13. This indicates that many participants who reported that the advice was helpful were able to use the DMSS to understand the cause of the bug properly. The results in Fig. 13 show that only 63% of the students could understand the DMSS advice for Ex3, which is considerably lower than for other problems. Consequently, the percentage of Ex3 in Fig. 14 was also low.

V. Discussions

A. Interaction Between Problem Difficulty and Various Factors

The impact of the problem difficulty on the debugging success rates was significant (Fig. 11). Ex1 and Ex2 had success rates of 83% and 90%, respectively, indicating that they were relatively simple. In contrast, Ex3 and Ex4 had lower success rates of 29% and 56%, respectively, suggesting a higher difficulty. Notably, problem difficulty had the

following effects (Table 3). For simple problems such as Ex1 and Ex2, the DMSS achieved high success rates regardless of prior knowledge. The success rates of Ex1 were 91%, 89%, and for Ex2, they were 96%, and 91%, respectively. However, for difficult problems, such as Ex3 and Ex4, the success rates dropped significantly, even with prior knowledge and DMSS. The success rates for Ex3 and Ex4 were 50% and 65%, respectively. The rates were even lower under the "unknown error with DMSS" condition, with Ex3 at 29% and Ex4 at 57%. These results suggest that a higher problem difficulty lowers success rates, making prior knowledge and quality of the DMSS advice crucial.

B. Impact of Prior Knowledge

As shown in Table 3, the debugging success rate is strongly influenced by whether the errors are known. Across all the problems, the success rate for known errors was 81% with DMSS and 96% without DMSS. In contrast, the success rates for unknown errors were significantly lower at 61% with DMSS and 28% without DMSS. This result indicates that prior experience and knowledge of specific errors significantly enhance problem-solving abilities in programming learning. Notably, students achieved a high debugging success rate of 96% for known errors, even without using the DMSS, suggesting that they developed the ability to effectively handle errors they had previously encountered. Interestingly, the success rate of known errors was higher without DMSS (96%) than with DMSS (81%). An important consideration here is that when participants responded "known" to Q3, it remained unclear whether they had previously succeeded in debugging the error. Participants who reported errors as known but did not use the DMSS likely had prior successful debugging experiences and understood Rails well. Conversely, there were participants who likely, despite recognizing the error, had no prior successful debugging experience and were able to debug successfully by referring to the DMSS advice. Their debugging success rate would naturally be lower than that of participants with a better understanding. Additionally, the debugging success rate for Ex3 was particularly low at 50% even when the error was known and the DMSS was used, which significantly reduced the overall debugging success rate for known errors with DMSS usage. This resulted in a lower debugging success rate when using the DMSS.

C. Analysis of DMSS Usage Rate

The usage rates of the DMSS varied depending on the problem, with 67% for Ex1, 72% for Ex2, 67% for Ex3, and 66% for Ex4. Notably, the DMSS usage rate for Ex3, the most difficult problem (debugging success rate: 29%), did not differ significantly from that for Ex2, the simplest problem (debugging success rate: 90%). This suggests that students referred to the advice at a consistent rate, regardless of the difficulty or knowledge of the problem. It is also noteworthy that many students consulted the advice, even for errors with which they were already familiar. This may indicate a lack of confidence in their knowledge or a need to confirm their understanding. Particularly, for challenging problems such as Ex3, where all students who identified the error as known still referred to the advice, they seemed to seek additional support when they perceived the problem as more difficult.

D. Effect of the DMSS Advice

The impact of the DMSS advice was significant, especially for unknown errors. For these errors, the debugging success rate with advice was 61%, which was more than twice that observed without advice (28%). This highlights the importance of providing appropriate guidance for errors encountered for the first time. By contrast, the effects of advice on known errors showed more complex patterns. Overall, the success rate for known errors was higher without DMSS (96%) than with DMSS (81%). However, when examined by problem, Ex1 showed nearly identical success rates under both conditions (91% without advice and 92% with advice), whereas for Ex3, the data were available only for the DMSS condition. These results suggest that the effectiveness of advice for known errors depends on the type of error and difficulty of the problem.

E. Generalizability beyond Ruby on Rails

This study used Rails as the web application framework. Other frameworks and programming languages are also available for the development of web applications. Educational institutions often provide instructions using Java or Python. This section discusses the possibility of using the DMSS in these environments. As shown in Fig. 1, the EIC detects errors and transmits the error information to the DMSS, which functions as an application server. The errors are detected based on error messages output to log files. While the EIC needs to be prepared individually for different programming languages and frameworks because it differs across environments, the DMSS can be used as is because it operates as independent software. Therefore, the applicability is determined based on whether a mechanism exists for outputting error messages to log files. We investigated commonly used frameworks other than Rails, including Flask, Django, Spring Boot, Laravel, Symfony, and Express.js. The results showed that these frameworks, had the required mechanisms for outputting error information to log files. Although it is necessary to specify the log file name in the configuration files, the DMSS proposed in this study can be used as is.

F. Limitations

This study has several limitations. First, the four problems used in the experiment did not represent all programming errors. In particular, the lack of data for the known error without advice condition in Ex3 indicates room for improvement in problem selection and data collection. Additionally, students' self-reported judgments of "known/unknown" may not accurately reflect their actual knowledge state. Furthermore, no detailed analysis has been conducted on the content, quality, or presentation method of advice. Therefore, it remains unclear which types of advice are most effective.

VI. Conclusion

This study focused on the difficulties that programming beginners face with debugging, and developed and evaluated a new debugging assistance system (DMSS). It automatically collects error information generated by students, stores solutions annotated by teachers in a database, and presents similar error information and solutions to other students

using the same programming materials.

In a learning environment for beginners in Rails, important findings regarding the effectiveness of DMSS were obtained from an experiment involving 53 university students. In particular, the success rate when using the DMSS for unknown errors was more than twice that when not using the DMSS, indicating a significant supporting effect for errors encountered for the first time. However, for known errors, the students showed high self-solving abilities, and the effect of the DMSS was limited. The experimental results revealed that problem difficulty, prior knowledge, and advice quality interact in complex ways to influence the debugging success rates. For simple problems, high success rates were achieved using the DMSS, regardless of prior knowledge, whereas for complex problems, success rates decreased even when using the DMSS, despite having prior knowledge. This suggests that as problem difficulty increases, the quality and content of advice become more important. The usage rate of the DMSS remained constant at approximately 70%, regardless of problem difficulty, indicating that students tended to refer to advice regardless of their knowledge state. In particular, the fact that many students referenced advice even for known errors suggests that they did not have complete confidence in their knowledge, or wanted to confirm their knowledge through advice.

In future research, additional experiments are necessary to validate the findings of this study. For instance, to verify the validity of the results, specific data for qualitative and quantitative analysis can be collected and analyzed, such as statistics on student feedback for different types of suggestions. Additionally, data on student interactions with the DMSS can be obtained to supplement the engagement metrics. Furthermore, there is a need to address more diverse error patterns, optimize the quality and content of advice, and develop personalized support methods based on learners' knowledge levels and learning styles. Specifically, debugging success rates vary depending on the quality and content of advice. Future experiments should examine changes in success rates when different types of advice are provided, to explore more appropriate methods of delivering advice. The DMSS developed in this study can be applied to other programming languages and frameworks, and is expected to be implemented in various programming learning environments.

Conflict of Interest

The author declares no conflict of interest.

Funding

This work was supported by JSPS KAKENHI Grant Number 21K02758, Japan.

References

[1] J. Nouri, L. Zhang, L. Mannila, and E. Norén, "Development of computational thinking, digital competence and 21st century skills when learning programming in K-9," *Education Inquiry*, vol. 11, pp. 1–7, 2019.

[2] G. Wong and H. Cheung, "Exploring children's perceptions of developing twenty-first century skills through computational thinking and programming," *Interactive Learning Environments*, vol. 28, pp. 438–450, 2018.

[3] M. Guzdial, "Computing education as a foundation for 21st century literacy," in *Proc. Technical Symposium on Computer Science Education*, Minneapolis MN, USA, 27 Feb. 2019, pp. 502–503.

[4] S. Huang and Y. Xu, "A comparative study on programming education—Based on China and America," *Journal of Education, Humanities and Social Sciences*, vol. 15, pp. 220–231, 2023.

[5] J. Figueiredo and F. J. García-Peñalvo, "Design science research applied to difficulties of teaching and learning initial programming," *Universal Access in the Information Society*, vol. 23, pp. 1151–1161, 2022.

[6] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming," *ACM Transactions on Computing Education (TOCE)*, vol. 18, pp. 1–24, 2017.

[7] D. Radaković and W. Steingartner, "Common errors in high school novice programming," *IPSI Transactions on Internet Research*, vol. 20, no. 1, pp. 47–59, 2024.

[8] C. S. Cheah, "Factors contributing to the difficulties in teaching and learning of computer programming: A literature review," *Contemporary Educational Technology*, vol. 12, ep272, 2020.

[9] H. Du, W. Xing, and Y. Zhang, "A debugging learning trajectory for text-based programming learners," in *Proc. 2021 ACM Conference*, Virtual Event Germany, June 2021, 645.

[10] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debugging among novice computer science students," in *Proc. Annual Conference on Innovation and Technology in Computer Science Education*, Caparica, Portugal, Jun. 2005.

[11] J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *Proc. Visual Languages and Human-Centric Computing (VL/HCC'06)*, Brighton, UK, Sep. 2006

[12] T. Barik *et al.*, "Do developers read compiler error messages?" in *Proc. IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, Buenos Aires, Argentina, May 2017.

[13] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proc. 28th International Conference on Software Engineering*, May 2006, pp. 492–501.

[14] A. J. Ko and B. A. Myers, "Finding causes of program output with the Java Whyline," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, Boston MA, USA, April 2009, pp. 1569–1578.

[15] S. Fitzgerald *et al.*, "Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers," *Computer Science Education*, vol. 18, pp. 93–116, 2008.

[16] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, and C. Zander, "Debugging from the student perspective," *IEEE Transactions on Education*, vol. 53, pp. 390–396, 2010.

[17] S. Marwan, A. Dombe, and T. W. Price, "Unproductive help-seeking in programming: What it is and how to address it," in *Proc. 2020 ACM Conference on Innovation and Technology in Computer Science Education*, Trondheim, Norway, June 2020, pp. 54–60.

[18] U. Ahmed, N. Srivastava, R. Sindhgatta, and A. Karkare, "Characterizing the pedagogical benefits of adaptive feedback for compilation errors by novice programmers," in *Proc. IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Seoul South, Korea, Jun. 2020, pp. 139–150.

[19] J. Kim, Y. Sun, and F. Zhang, "ReCodez: An intelligent and intuitive online coding editor using machine learning and AI," *Computer Science & Information Technology (CS & IT)*, Sydney, Australia, Oct. 2020, pp. 157–164.

[20] R. Chenartha and C. Safitri, "Web-based realtime course platform with integrated live coding interface," *IT for Society: Journal of Information Technology*, vol. 9, no. 1, 2024.

[21] A. Gupta, M. Jindal, and A. Goyal, "Identification of student programming patterns through clickstream data," in *Proc. 2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, Greater Noida, India, Feb. 2024.

[22] J. Leinonen *et al.*, "Using large language models to enhance programming error messages," in *Proc. 54th ACM Technical Symposium on Computer Science Education*, Toronto ON, Canada, Mar. 2023.

[23] F. Assiri and H. Elazhary, "Automated Java exceptions explanation using natural language generation techniques," *Computer Applications in Engineering Education*, vol. 28, pp. 626–644, 2020.

[24] A. Amburle, C. Almeida, N. Lopes, and O. Lopes, "AI based code error explainer using gemini model," in *Proc. 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, Salem, India, Jun. 2024.

[25] A. Taylor, A. Vassar, J. Renzella, and H. A. Pearce, "dcc--help: Transforming the role of the compiler by generating context-aware error explanations with large language models," in *Proc. 55th ACM*

Technical Symposium on Computer Science Education, Portland OR, USA, Mar. 2024.

[26] S. Schacht, S. Barkur, and C. Lanquillon, "Generative agents to support students learning progress," in *Proc. 5th edition of the annual International Conference on Business meets Technology*, Valencia, Spain, July 2023.

[27] E. Paikari, B. Sun, G. Ruhe, and E. Livani, "Customization support for CBR-based defect prediction," in *Proc. 7th International Conference on Predictive Models in Software Engineering*, Alberta, Canada, Sep. 2011.

[28] M. Velez, P. Jamshidi, N. Siegmund, S. Apel, and C. Kästner, "On debugging the performance of configurable software systems: Developer needs and tailored tool support," in *Proc. 44th International Conference on Software Engineering*, Pennsylvania, USA, May 2022, pp. 1571–1583.

[29] D. Oliveira *et al.*, "The untold story of code refactoring customizations in practice," in *Proc. IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, Melbourne Victoria, Australia, May 2023, pp. 108–120.

[30] Ruby on Rails. [Online]. Available: https://rubyonrails.org/

[31] K. Takahashi and N. Suzuki, "Learning status report tool for programming learning services," *Procedia Computer Science*, vol. 207, pp. 1562–1570, 2022.

Copyright © 2025 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).