Leveraging Error Information from Identical Learning Materials for Debugging Effectiveness

Keiichi Takahashi
Kindai University
11-6 Kayanomori, Iizuka, Fukuoka, Japan

ABSTRACT
As programming education has become more widespread, helping novices fix bugs remains a significant challenge. While various support methods have been studied, implementing them in educational settings requires customization of specific teaching materials, which demands time and technical resources. This study focuses on the fact that teaching materials are used repeatedly in educational settings. If error information from previous uses of the same teaching materials can help debug similar errors that occur later, it enables debugging support tailored to specific materials by simply conducting classes and collecting error data. To test this hypothesis, a system is proposed that automatically collects error information, searches for similar errors, and suggests appropriate solutions. Experiments were conducted using this system by collecting error data from programming practice courses at Kindai University in 2021 and 2022. The results showed that past error information was able to effectively support debugging for approximately 30% of future errors, confirming the potential of this approach as a new method to support novice programming learning.

General Terms
Algorithms, Design, Experimentation, Human Factors, Measurement, Education

Keywords
Programming Education, Debugging Support, Error Information Sharing, Similar Error Retrieval, Web Application Development

1. INTRODUCTION
Programming skills have become increasingly important. Programming is now recognized not only as a means of software development but also as a skill that develops problem-solving abilities and logical thinking [1], [2]. Programming education has been widely adopted, from elementary to higher education and professional training [3], [4].

However, as programming education expands, novices continue to face significant learning challenges [5], [6]. One major factor is bugs (program errors) caused by input mistakes and logical errors [7]. While these bugs are an inevitable part of the learning process, they can become substantial barriers for novices [8]. Many learners who have not yet developed the ability to find, identify, and fix errors may lose motivation and give up programming when faced with bugs [9], [10].

Several common factors make bug fixing difficult for novices. First, they struggle to correctly interpret error messages because they are unfamiliar with programming language syntax and basic structures. Error messages designed for language developers often contain advanced technical terms and abstract expression. As a result, novices frequently find it difficult to understand error messages and determine the appropriate fixes [11], [12], [13]. Second, while novices typically encounter syntax, type, and logic errors during learning, they have not yet developed the ability to identify the specific causes of these errors, often leading to time-consuming solutions. This is particularly challenging when errors in one part of the program affect other parts, making it difficult for novices to identify the root cause [14], [15], [16]. These recurring situations can not only hinder learning progress but also cause significant frustration. Third, novices may not receive sufficient feedback during the error correction. While traditional programming education typically involves direct support from instructors, the rise of online education and learner diversity has reduced the opportunities for individual instruction. This can leave learners isolated when fixing errors, thereby hampering efficient learning [17], [18].

Various technical approaches have been proposed to address these issues. For example, many online programming platforms now provide features that analyze learners' codes in real time and immediately identify errors, allowing early detection and correction [19], [20]. Systems that analyze coding patterns and suggest potential errors and fixes have also been developed [21]. Research on tools that translate error messages into explanations suitable for novices is also progressing. Recently, systems using Large Language Models (LLMs) have been proposed to provide novices with natural language error explanations and code completion [22], [23]. LLMs' generative capabilities offer clear explanations of typical errors and provide appropriate correction suggestions [24], [25]. There have also been attempts to analyze learners' progress in real time and provide feedback based on their individual weaknesses [26].

While many technical approaches have been proposed for programming learning debug support systems, using these research outcomes in one's own educational environment requires customization of specific teaching materials [27], [28], [29]. Customization requires at least information about programming learning materials, and sometimes data from students' practice results using these materials. It is technically time-consuming for users of research outputs to prepare such information. However, programming classes often repeatedly use the same programming materials. If the error information generated when students work with these materials can be used to debug others using the same materials, error information from each class can be automatically collected and accumulated, minimizing the cost of customizing classroom support tools.

Therefore, this study examines whether error information collected from using specific programming materials can help fix errors that occur when using the same programming materials. This study uses Rails (Ruby on Rails) as the programming environment [30]. Rails is one of the major web application development frameworks. Since Rails adopts the

MVC (Model-View-Controller) architecture, implementing a single feature requires combining multiple files. We chose this development environment to verify the effectiveness of debugging support using error information in a programming environment that assumes realistic application development. As Rails currently influence various frameworks in use, the findings obtained are expected to be applicable to other frameworks as well.

2. PROPOSED SYSTEM

2.1 System Architecture

We propose a system that automatically collects error information from students working on programming assignments and provides debugging advice. Figure 1 shows the system's structure. A tool that automatically collects error information when students encounter errors in programming assignments is called the Error Information Collector (EIC). When EIC detects an error, it sends the related information to Debugging Method Suggestion System (DMSS). DMSS stores the received information in Debug Database (DD). Instructors can review this error information and annotate it with causes and solutions. When another student encounters an error, similar errors are extracted from DD and presented to the student via a tool called Debugging Method Suggester (DMS). The following sections explain EIC and DMS in detail.

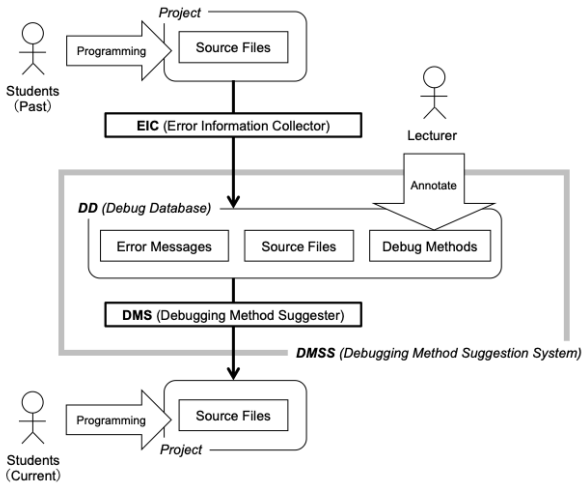

Fig 1: System Architecture of Debugging Method Suggestion System (DMSS)

2.2 Error Information Collector (EIC)

This tool automatically sends relevant files to DMSS when a student encounters an error. EIC detects an error when the string "Error" appears in a log file. Most modern web application frameworks include web servers that output execution logs to facilitate debugging. For example, in Rails, this log file is called development.log. This tool can be adapted for other web application frameworks by configuring the log file name and the error string. The information sent to DMSS includes the user ID, IP address, error message, and updated files. The extracted information is then serialized and sent to DMSS.

2.3 Algorithm for Determining the Most Similar Error Information (MSEI)

DMS compares files from a student's current error with files registered in DD to extract error information with the highest similarity. This extracted information is referred to as the Most Similar Error Information (MSEI). This error information is assumed to have debugging methods annotated by instructors. A key feature of our method is that it does not perform syntax analysis, which is expected to minimize the customization required for programming materials. Modern web development frameworks such as Rails manage multiple files in directories according to their roles. There are approximately 35 directories in Rails. To determine the similarity between multiple files containing errors, our method calculates the similarity between files with the same name in the same directory. The algorithm for determining MSEI from DD is described below [31], [32], [33].

First, we extract error information by matching error messages from DD. If only a single instance of error information is extracted, it is designated as MSEI. If multiple instances are extracted, we determine MSEI as follows.

Two metrics are introduced: D_{diff} and F_{diff}. D_{diff} is the number of files that exist in only one of the two directories. F_{diff} is the number of different lines between files with the same name, and F_{diff} is calculated using the Longest Common Subsequence (LCS). For example, when D_{diff} is zero, both directories contain files with the same names. When F_{diff} is zero, the contents of the two files are identical. After extracting the error information with matching error messages from DD, we determine MSEI from these candidates using the following rules:

Rule 1: If there is only one error information with a minimum, it becomes MSEI.

Rule 2: If there is only one piece of error information with both the minimum D_{diff} and minimum F_{diff}, it becomes MSEI.

Rule 3: If MSEI cannot be determined by the above rules, the oldest error information among the candidates becomes MSEI.

3. EXPERIMENTS

3.1 Research Method

The 12th session of a web application programming course was conducted using Ruby, which was offered to third-year students at Kindai University. Each session lasted for 180 min. Students had varying programming experience levels: 45% were beginners, 35% had intermediate skills, and 20% were advanced learners. In this course, students learned Ruby basics and programming using Rails. In the 12th session, students developed a web application using Rails to upload image files. This study examined error information collected from students during this class in both 2021 and 2022. The class had 67 students in 2021 and 50 students in 2022. The study evaluated whether error information from students in 2021 could be useful for debugging errors encountered by students in 2022.

3.2 Programming Materials

All steps for developing a web application are described in the programming materials, which consist of a total of 24 steps. Since the same programming materials were used in both the 2021 and 2022 classes, the programming steps were identical. As Rails adopts the MVC architecture, programs need to be written in various MVC files. In total, programs need to be written in six files, including main files such as the routing file, which links URLs to program calls, the controller file, which retrieves data from the database based on the requested content, and the view file, which contains a mixture of HTML and

Ruby. In such a development environment, compared to single-file programs, programming input errors are more likely to occur, and debugging tends to be more challenging.

3.3 Method for Evaluating the Usefulness of Error Information for Debugging

The error information from 2021 was loaded into the DD. Using the algorithm described in Section 2.3, the MSEI was determined for the errors that occurred in 2022. The debugging advice was obtained associated with this error information. The course instructors evaluated whether this advice was appropriate for errors that occurred in 2022.

Figure 2 illustrates an example of verifying the applicability of this advice. The figure shows a case in which a student inputs an Error Code in the view file and receives an ActionView::Template::Error when executing the web application. It also shows the MSEI and its associated advice. The error in the MSEI program occurred because, similar to the Error Code, there was a mismatch between Ruby's block parameter and the variable name used within the block. It was judged that debugging would be possible because students could be expected to notice their mistakes by reading their advice. Conversely, if the MSEI was caused by a different issue, the advice was judged as not applicable.

Error message	ActionView::Template::Error-V
Error code	<% @Images.each do |image| %> <%= image.title %> <%= image_tag "/get_image/#{image.id}" %> <%= link_to 'Delete', "/bookmarks/#{book.id}", method: :delete %> <%= link_to 'Edit', "/bookmarks/#{book.id}/edit" %> <% end %> <%= link_to 'Add Item', '/images/new' %>
MSEI	<% @image.each do |book| %> <%= link_to image.title %> <%= image_tag "/get_image/#{image.id}" %> <% end %> <%= link_to 'Add Item', '/images/new' %>
Advice	The block argument book needs to match the variable image used within the block. In the following example, it is better to change |book| to |image|.

Fig 2: An example of debugging feasibility verification

4. RESULTS

4.1 Frequency Distribution of Error Messages

Before examining the availability of debuggable information, an analysis of the content of error information is presented. There were 248 errors in 2022 and 236 errors in 2021. Figure 3 shows a graph of the seven most frequent errors categorized by error message. The vertical axis shows the relative frequency, indicating the proportion of each error message relative to the total number of errors. The labels on the horizontal axis include "V" or "C", indicating the functionality of the file in which the error occurred. "V" indicates a view function file, and "C" indicates a controller function file. In both 2022 and 2021, the most frequent error was ActionView::TemplateError, which occurred in the view function files. The line graph shows the cumulative relative frequencies. The 2022 graph shows that the top seven error messages accounted for 90% of all the errors. A similar trend was observed in 2021 as well. The seven most frequent error messages are analyzed in detail below.

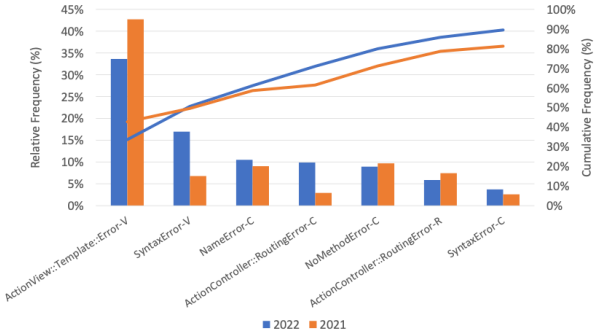

Fig 3: Frequency distribution of error messages

4.2 Feasibility of Debugging for Each Error Message

For the seven most frequent errors shown in Figure 3, MSEI was determined using the algorithm described in Section 2.3. The results of evaluating debugging possibilities with the method described in Section 3.3 are shown in Table 1. The "Errors" column shows the number of occurrences for each error. The "Debugged" column shows the number of cases judged as debuggable, with the percentage shown in parentheses. Of the 222 total occurrences of the seven error messages, 76 (34%) were judged as debuggable. Among the seven types, ActionView::TemplateError-V was the most frequent, with 88 occurrences, of which 49 cases (56%) were debuggable. SyntaxError-V was the second most frequent, with 33 occurrences, of which 13 (39%) were debuggable. This indicates that view function files tend to be more prone to errors but are more debuggable than controller or routing files. Generally, view function files contain more lines of code than other files because they generate HTML, making them more susceptible to input errors.

Table 1. Number of debuggable errors per error message

No	Error Messages	Errors	Debugged
1	ActionView::Template::Error-V	88	49 (56%)
2	SyntaxError-V	33	13 (39%)
3	NameError-C	24	9 (38%)
4	ActionController::RoutingError-C	32	1 (3%)
5	NoMethodError-C	21	2 (10%)
6	ActionController::RoutingError-R	19	0 (0%)
7	SyntaxError-C	5	2 (40%)
	Total	222	76(34%)

5. DISCUSSION

5.1 Example of ActionView::TemplateError-V

Of the ActionView::TemplateError-V errors, 49 (56% of all errors) were judged as debuggable (Table 1). Among these, 44 had programs similar to those shown in Figure 4. For easier analysis, we highlighted the differences between the error code and MSEI. By comparing the two programs, it was observed that MSEI contained an error where "book" was mistakenly entered instead of "image." Nevertheless, this was determined to be MSEI because the overall program structure was similar

to those of other errors types, and debugging was considered successful because the error factors were of the same type as those in other cases.

Error message	ActionView::Template::Error-V
Error code	<% @images.each do |image| %> <%= image.title %> <%= image_tag "/get_image/#{e.id}" %> <%= link_to 'Delete', "/images/#{image.id}",method: :delete %> <%= link_to 'Edit', "/images/#{image.id}/edit" %> <% end %> <%= link_to 'Add Item', '/images/new' %>
MSEI	<% @image.each do |book| %> <%= link_to image.title %> <%= image_tag "/get_image/#{image.id}" %> <%= link_to 'Delete', "/images/#{book.id}", method: :delete %> <%= link_to 'Show', "/images/#{book.id}" %> <%= link_to 'Edit', "/images/#{book.id}/edit" %> <% end %> <%= link_to 'Add Item', '/images/new' %>

Fig 4: Successful debugging example of ActionView::TemplateError-V

Figure 5 shows an example of an error in which debugging was judged to be unsuccessful for ActionView::TemplateError-V. This error occurred during the process of generating HTML from a template file that contains Ruby code. Compared with the program in Figure 4, the program in Figure 5 appears more similar, differing by just one line. However, upon examining MSEI program, no syntax errors were found. The error occurred because the @images variable was empty due to an error in the controller program upstream in the data flow. Therefore, debugging was judged to be unsuccessful because MSEI addressed a different issue from that indicated by the error code. As shown in Section 2.3, since similarity is calculated as the total number of different lines across several files, this demonstrates that debugging can fail even when the file where an error occurs has a high similarity score.

Error message	ActionView::Template::Error-V
Error code	<% @images.each do |image| %> <%= image.title %> <%= link_to 'Delete', "/bookmarks/#{book.id}", method: :delete %> <% end %> <%= link_to 'Add Item', '/images/new' %>
MSEI	<% @images.each do |image| %> <%= image.title %> <% end %> <%= link_to 'Add Item', '/images/new' %>

Fig 5: Failed debugging example of ActionView::TemplateError-V

5.2 Example of SyntaxError-V

Next, we analyze SyntaxError-V, the second most frequent error message. As the name suggests, this error occurs when the program violates Ruby's syntax rules. Figure 6 shows representative examples of 13 debuggable error cases. A common mistake in both the error code and MSEI is highlighted in a red box. Both cases show a missing closing tag "%>" that should correspond to "<%= link_to." Although the missing closing tag is only two characters long, this program was classified as MSEI because the rest of the program was similar and had the characteristic of a missing closing tag.

Error message	SyntaxError-V
Error code	<% @images.each do |image| %> <%= image.title %> <%= link_to 'Delete', "/images/#{image.id}", method: <% end %> <%= link_to 'Add Item', '/images/new' %>
MSEI	<% @images.each do |image| %> <%= image.title %> <%= link_to 'Delete', "/images/#{image.id}", <% end %> <%= link_to 'Add Item', '/images/new' %>

Fig 6: Successful debugging example of SyntaxError-V

However, Figure 7 shows an example in which the system failed to provide appropriate MSEI despite having the same error cause (red arrow). In this case, the suggested program failed to debug because the error was caused by incorrect quotation marks, rather than a missing closing tag. This demonstrates that even when programs appear similar, the appropriate MSEIs cannot always be extracted.

Error message	SyntaxError-V
Error code	<% @images.each do |image| %> <%= link_to image.title %> <%= link_to 'Delete', "/images/#{image.id}", method: :delete %> <%= link_to 'Edit', "/images/#{image.id}/edit" <% end %> <%= link_to 'Add Item', '/images/new' %>
MSEI	<% @images.each do |image| %> <%= link_to image.title %> <%= image_tag "/get_image/#{image.id}" %> <%= link_to 'Delete', "/images/#{image.id}", method: :delete %> <%= link_to 'Show', "/images/#{image.id}" %> <%= link_to 'Edit', "/images/#{image.id}/edit" %> <% end %> <%= link_to 'Add Item', '/images/new' %>

Fig 7: Successful debugging example of SyntaxError-V

5.3 Example of NameError-C

Figure 8 shows examples of the third most frequent debuggable error cases. This error occurs when an undefined local variable or a constant is used. Looking at the error code, we can see that the error occurred because @image.update method used the local variable "file" before it was assigned an initial value. Similarly, the MSEI program showed an error in which the variable "title" was used in the Image.new method before being initialized. MSEI was determined to be appropriate because debugging would be possible by checking whether values were set before using the method arguments.

Error message	NameError-C
Error code	# ... def update @image = Image.find(params[:id]) title = params[:image][:title] @image.update(title: title, file: file) file = params[:image][:file].read redirect_to '/' end # ...
MSEI	# ... def create file = params[:image][:file].read @image = Image.new(title: title , file: file) image.save redirect_to '/' end # ...

Fig 8: Successful debugging example of NameError-C

In contrast, Figure 9 shows an example in which the error code sets data to the variable "@file", but an error occurs because @image.update method argument specifies the variable "file" without the "@" symbol, resulting in the use of an uninitialized variable. In the MSEI, the error is due to using the variable "title" without setting its value. Since these are different causes, it was determined that debugging would be difficult because the MSEI would not be helpful.

Error message	NameError-C

Error code	# ...
	def update
	  @image = Image.find(params[:id])
	  title = params[:image][:title]
	  @file = params[:image][:file].read
	  @image.update(title: title , file: file)
	  redirect_to '/'
	end
	# ...

MSEI	# ...
	def create
	  file = params[:image][:file].read
	  @image = Image.new(title: title , file: file)
	  image.save
	  redirect_to '/'
	end
	# ...

Fig 9: Failed debugging example of NameError-C

5.4 Challenges in Extracting Debuggable Error Information

In this section, we examine situations in which debugging was determined to be difficult by referring to specific programs. Based on this analysis, two conditions necessary for successful debugging were identified using our method of referencing past error information and advice.

Condition 1: Similar error information exists in DD.

Condition 2: Error information with the same error factors can be extracted from DD.

In this study, Condition 1 was satisfied because we focused on frequently occurring errors. Error information is likely to improve as more data are accumulated through repeated programming classes using the same teaching materials. As shown in Sections 5.1 to 5.3, even when programs had similar formal structures, their error causes could differ. As shown in Table 1, only 30% of the error information met Condition 2. How should this 30% result be evaluated? Many studies have been conducted to measure program similarity, such as code-clone research. Recently, the accuracy of clone code detection has been improved through the use of LLMs [34]. Research is also progressing toward developing code search engines to promote code reuse [35]. The effectiveness of search queries in major code search engines has been reported to be between 25% and 60%. It is important to note that the programs targeted in these studies did not contain bugs. As shown in our experiment, even when the same programming materials are used, errors can occur in various ways. It is generally known that searching for code containing bugs is challenging. Given this context, while a 30% success rate for all error information may not be sufficient for practical use, it could be considered an reasonable initial result. Through this investigation, areas for improvement in the method were identified. MSEI was determined using the algorithm described in Section 2.3. To extract similar error information, it compares all the error information that occurs with the same error message across files. As shown in sections 5.1 to 5.3, incorrect error information sometimes became MSEI because of the high similarity with files other than those where the error occurred. As a potential solution to this problem, the algorithm from Section 2.3 could be applied only to files that are upstream of the error-occurring file in the data flow, potentially leading to more appropriate error information becoming MSEI.

6. CONCLUSIONS

This study addressed a key challenge faced by novices in programming education: the difficulty of debugging. This study investigated how past error information, gathered from the same teaching materials, could help address this problem. For this investigation, three tools were proposed: Error Information Collector (EIC), Debugging Method Suggestion System (DMSS), and Debugging Method Suggester (DMS). In DMS, we introduced an algorithm to determine the Most Similar Error Information (MSEI). Error information was collected using the proposed tools during programming practice courses at Kindai University in 2021 and 2022. An analysis of the collected data showed that 30% of the errors in 2022 were considered debuggable using MSEI extracted from the 2021 data. In code search research, search queries for defect-free datasets have been reported to be effective 25-60% of the time. Considering that our target data consisted only of programs with bugs, these results were reasonable. However, a 30% success rate for debugging support is insufficient for practical use in classrooms. Going forward, several promising directions will be explored: (1) integrating machine learning algorithms to improve error similarity calculations, (2) developing cross-language error pattern recognition, (3) implementing real-time adaptive feedback mechanisms, and (4) creating personalized debugging assistance based on individual learning patterns. Future work will also investigate the integration of Large Language Models for natural language explanation generation and the development of a comprehensive multi-institutional database for error pattern sharing. The long-term vision is to create an intelligent, scalable debugging support ecosystem that can adapt to diverse programming environments and educational contexts, ultimately fostering independent problem-solving skills among novice programmers across different institutions and programming domains.

7. 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, no. 1, pp. 1–17, 2019. https://doi.org/10.1080/20004508.2019.1627844

[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, no. 4, pp. 438–450, 2018. https://doi.org/10.1080/10494820.2018.1534245

[3] M. Guzdial, “Computing Education as a Foundation for 21st Century Literacy,” in Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE 2019), Minneapolis, MN, USA, Feb. 27–Mar. 2, 2019. https://doi.org/10.1145/3287324.3290953

[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. https://doi.org/10.54097/ehss.v15i.9272

[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. https://doi.org/10.1007/s10209-022-00941-4

[6] Y. Qian and J. Lehman, “Students’ misconceptions and other difficulties in introductory programming,” ACM Transactions on Computing Education, vol. 18, no. 1, pp. 1–24, 2017. https://doi.org/10.1145/3077618

[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. https://doi.org/10.58245/ipsi.tir.2401.05

[8] C. S. Cheah, "Factors contributing to the difficulties in teaching and learning of computer programming: A literature review," Contemporary Educational Technology, vol. 12, no. 3, Article ep272, 2020. https://doi.org/10.30935/CEDTECH/8247

[9] H. Du, W. Xing, and Y. Zhang, "A Debugging Learning Trajectory for Text-Based Programming Learners," In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education, Virtual Event, Germany, Jun. 26–Jul. 1, 2021. https://doi.org/10.1145/3456565.3460049

[10] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debugging among novice computer science students," in Proceedings of the 10th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2005), Caparica, Portugal, Jun. 27–29, 2005. https://doi.org/10.1145/1067445.1067472

[11] J. Ko, B. A. Myers, and D. H. Chau, "A Linguistic Analysis of How People Describe Software Problems," in Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Brighton, UK, Sept. 4–8, 2006. https://doi.org/10.1109/VLHCC.2006.3

[12] T. Barik et al., "Do Developers Read Compiler Error Messages?" in Proceedings of the 39th International Conference on Software Engineering (ICSE 2017), Buenos Aires, Argentina, May 20–28, 2017. https://doi.org/10.1109/ICSE.2017.59

[13] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20–28, 2006. https://doi.org/10.1145/1134285.1134355

[14] A. J. Ko and B. A. Myers, "Finding causes of program output with the Java Whyline," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2009), Boston, MA, USA, Apr. 4–9, 2009. https://doi.org/10.1145/1518701.1518942

[15] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas, and C. Zander, "Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers," Computer Science Education, vol. 18, no. 2, pp. 93–116, 2008. https://doi.org/10.1080/08993400802114508.

[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, no. 3, pp. 390–396, 2010. https://doi.org/10.1109/TE.2009.2025266

[17] S. Marwan, A. Dombe, and T. W. Price, "Unproductive help-seeking in programming: What it is and how to address it," in Proceedings of the 25th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2020), Trondheim, Norway, Jun. 15–19, 2020. https://doi.org/10.1145/3341525.3387394

[18] U. Ahmed, N. Srivastava, R. Sindhgatta, and A. Karkare, "Characterizing the Pedagogical Benefits of Adaptive Feedback for Compilation Errors by Novice Programmers," in Proceedings of the 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET 2020), Seoul, South Korea, Jun. 27–Jul. 19, 2020. https://doi.org/10.1145/3377814.3381703

[19] J. Kim, Y. Sun, and F. Zhang, "ReCodez: An Intelligent and Intuitive Online Coding Editor using Machine Learning and AI," in Proceedings of the 12th International Conference on Computer Science and Information Technology (CSIT 2020), Sydney, Australia, Oct. 24–25, 2020. https://doi.org/10.5121/csit.2020.101216

[20] R. Chenartha and C. Safitri, "Web-based realtime course platform with integrated live coding interface," IT Society Journal of Information Technology, vol. 9, pp. 22–26, 2024. https://doi.org/10.33021/itfs.v9i1.5077

[21] A. Gupta, M. Jindal, and A. Goyal, "Identification of Student Programming Patterns through Clickstream Data," in Proceedings of the 2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT 2024), Greater Noida, India, Feb. 9–10, 2024. https://doi.org/10.1109/IC2PCT60090.2024.10486775

[22] J. Leinonen et al., "Using Large Language Models to Enhance Programming Error Messages," in Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE 2023), Toronto, ON, Canada, Mar. 15–18, 2023. https://doi.org/10.1145/3545945.3569770

[23] F. Assiri and H. Elazhary, "Automated Java exceptions explanation using natural language generation techniques," Computer Applications in Engineering Education, vol. 28, no. 3, pp. 626–644, 2020. https://doi.org/10.1002/cae.22232

[24] A. Amburle, C. Almeida, N. Lopes, and O. Lopes, "AI based Code Error Explainer using Gemini Model," in Proceedings of the 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC 2024), Salem, India, Jun. 5–7, 2024. https://doi.org/10.1109/ICAAIC60222.2024.10574931

[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 Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024), Portland, OR, USA, Mar. 20–23, 2024. https://doi.org/10.1145/3626252.3630822

[26] S. Schacht, S. Barkur, and C. Lanquillon, "Generative Agents to Support Students' Learning Progress," in Proceedings of the 5th International Conference on Business Meets Technology (BmT 2023), Valencia, Spain, Jul. 13–15, 2023. https://doi.org/10.4995/bmt2023.2023.16750

[27] E. Paikari, B. Sun, G. Ruhe, and E. Livani, "Customization support for CBR-based defect prediction," in Proceedings of the 7th International Conference on Predictive Models in Software Engineering (PROMISE 2011), Alberta, Canada, Sept. 20–21, 2011. https://doi.org/10.1145/2020390.2020406

[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 Proceedings of the 44th International Conference on Software Engineering (ICSE 2022), Pennsylvania, USA, May 21–29, 2022. https://doi.org/10.1145/3510003.3510043

[29] D. Oliveira et al., "The untold story of code refactoring

customizations in practice," in Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE 2023), Melbourne, Victoria, Australia, May 14–20, 2023. https://doi.org/10.1109/ICSE48619.2023.00021

[30] Ruby on Rails. [Online]. Available: https://rubyonrails.org/ [Accessed: Apr. 12, 2025].

[31] K. Takahashi, "Analyze the possibility of advice based on historical error information for debugging during programming exercises," in Proceedings of the Hinokuni Information Symposium 2023, Kagoshima, Japan, Mar. 13–14, 2023.

[32] K. Takahashi and N. Suzuki, "Learning Status Report Tool for Programming Learning Services," Procedia Computer Science, vol. 207, pp. 1562–1570, 2022. https://doi.org/10.1016/j.procs.2022.09.213

[33] K. Takahashi, "A tool for estimating the learning progress of web application framework," in Proceedings of the 28th Foundation of Software Engineering Workshop (FOSE 2021), Fukushima, Japan, Nov. 11–13, 2021. https://doi.org/10.11309/fose.28.0_97

[34] M. Khajezade, J. J. Wu, F. H. Fard, G. Rodríguez-Pérez, and M. S. Shehata, "Investigating the efficacy of large language models for code clone detection," in Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC 2024), Lisbon, Portugal, Apr. 15–16, 2024. https://doi.org/10.1145/3643916.3645030

[35] L. Di Grazia and M. Pradel, "Code search: A survey of techniques for finding code," ACM Computing Surveys, vol. 55, pp. 1–31, 2023. https://doi.org/10.1145/3565971