ADROID Conference Series: Engineering and Technology
Volume 2, Issue 3, pp.74-82, 2026

Voice-Assisted Multimodal Debugging and Comparative Analysis of Python Applications in Visual Studio Code and PyCharm

Pavithra L, Rakshitha S, Madhumita P, Akash RS, Namee Jain

Department of Computational Intelligence, SRM Institute of Science and Technology, Kattangulathur – 603203, India

pavithrl3@srmist.edu.in, rs0512@srmist.edu.in, mp9075@srmist.edu.in, ar1256@srmist.edu.in, nj3716@srmist.edu.in

Abstract. Software debugging is a crucial but lengthy process in software development that may require developers to decode complicated error messages and stack traces. The growing complexity of Python applications today underscores the need for more intuitive and effective debugging methods. This paper focuses on enhancing debugging effectiveness by presenting a voice-assisted multimodal debugging system. This work is valuable because it helps reduce cognitive load and improve accessibility, particularly for developers who have difficulty with traditional text-based debugging techniques. Current debugging tools in integrated development environments like Visual Studio Code and PyCharm are mostly visual, which can slow error understanding and increase cognitive load. These solutions are not multimodal and are not able to accommodate the needs of different users. The aim of the study is to develop and test a system that combines voice feedback and visual debugging to enhance understanding of errors and overall productivity. The suggested system will present a voice-assisted debugging system that records the runtime exceptions and provides audio and graphical feedback, as well as smart recommendations. Experimental evidence shows that the system is also faster at identifying errors and gives a more user-friendly debugging experience in both development environments. Also, the approach offers benefits such as lower cognitive load, greater accessibility, and increased resource awareness through comparisons of IDE performance.

Keywords: Voice-Assisted Debugging, Multimodal Systems, Python Debugging, Visual Studio Code, PyCharm, Software Debugging, Cognitive Load Reduction, Error Analysis, Integrated Development Environments, Performance Evaluation.

I. INTRODUCTION

Modern programming languages and integrated development environments have improved the software development process but one of the most difficult and time-intensive activities of software development is debugging. Many developers are busy finding and correcting bugs, which directly affects productivity and software quality [6]. The process of debugging, analyzing program behavior and understanding the root cause of errors involve a lot of cognitive effort and focus.

The fact that software systems are becoming more complex and that the traditional methods of debugging are limited by the complexity of the software systems makes it necessary to find better methods of debugging. Most debugging instruments are based on textual descriptions of stack traces and console logs. Although the methods are detailed, they tend to flood developers, particularly beginners, and slow down the understanding of errors and the overall mental burden [13]. Moreover, developers often alternate between tools and documentation, which also disrupts their workflow and makes it less efficient.

The significance of better debugging mechanisms consists in the fact that they help to increase the productivity of the developer, to decrease the cognitive load and to make the programming accessible. Studies have indicated that there are cognitive factors including mental workload and attention that are critical in debugging performance [12]. Additionally, research that has been done on the behavior of developers has shown that ineffective debugging behavior may result in more frustration and errors in the development of software [7]. Thus, it is highly desirable to have systems that can render debugging information in a more accessible and easier to understand way.

The current debugging systems of popular integrated development environments like Visual Studio Code and PyCharm have strong capabilities, including breakpoints, step execution, and variable inspection. But these systems are mainly visual and the error messages are manually interpreted by the developers. Although they have sophisticated features, they do not support multimodal interaction and do not actively help to alleviate cognitive load when performing the debugging activity [3]. In addition, the existing tools are not effective to accommodate users who might need other types of interaction, including auditory feedback.

The use of visual processing is one of the most significant drawbacks of current methods as it may slow down the process of error detection and predispose to misinterpretation. Moreover, the systems lack simplified explanations and smart suggestions based on types of errors. Consequently, developers tend to use the external resources to learn and fix the problems and this interferes with their workflow and adds more time to the debugging process.

This study aims to create and deploy a voice-assisted multimodal debugging system that will improve the comprehension of errors and efficiency of debugging. The system will be designed as a way to combine auditory feedback with visual display, which will enable the developers to process information in multiple channels at the same time. Moreover, the research aims to compare the performance of the suggested system in two popular development tools, Visual Studio Code and PyCharm, to learn how they contribute to the efficiency of the debugging process and the use of resources.

To overcome these difficulties, a voice-assisted debugging system is suggested, which records exceptions at the run-time and offers real-time notification in the form of speech synthesis and graphical display. The system also has a suggestion engine to steer the developers to possible solutions, thus saving time to fix errors. The suggested method is based on the integration of various communication mechanisms, which is compatible with human cognition and enhances the efficiency of the overall debugging.

The outcomes of the current research prove that the suggested system can improve the speed of error detection and improve the debugging experience. As experimental analysis reveals, there are some significant differences in the execution time and resource consumption of the Visual Studio Code and PyCharm that should be considered when choosing the right development environment to perform a certain task. Voice feedback also helps increase the rate of understanding and decrease the cognitive load.

The benefits of the suggested system are the increased accessibility, decreased cognitive load, increased efficiency in debugging, and an improved user experience. This study will help in the creation of more intelligent and user-friendly software engineering tools by adding multimodal interaction to the debugging processes.

II. LITERATURE REVIEW

The problem of debugging has received a lot of research attention as an important process in software development, and researchers have been interested in the behavior of developers and enhancing the debugging process. Li and Coblenz [1] investigated the actual practices of debugging in the real-world setting through a grounded theory framework and discovered typical patterns and inefficiencies in the professional setting, yet their research was specific to the industrial setting. Likewise, Stolp et al. [2] examined cognitive load during software development on physiological measures and found that debugging tasks indeed have a greater mental load, but their methodology needs specialized hardware, which is less practical in daily life.

Several studies have been conducted on the comparison of debugging techniques and performance. Tajimalela et al. [3] performed a comparative study of the debugging techniques and concluded that there is no universal technique that is effective, but their research is not implemented in the real development setting. Gao and Hew [4] introduced a systematic method of debugging to simplify the cognitive load, which showed better performance in the educational context, but the model cannot be directly applied to the situation of professional development. Sun et al. [5] also confirmed the benefit of structured debugging by conducting a meta-analysis which demonstrated better learning results, although their context is mostly academic.

Previous background research by McCauley et al. [6] has also discussed the significance of debugging as a fundamental computer science education skill, and the insufficient support tools. A long-term industrial observation of developer behavior by Lanza et al. [7] has identified that a lot of development time is consumed in debugging, however their study does not suggest any technical solution. Experiments by Khan et al. [8] have indicated that the emotional aspects like mood can affect debugging performance, and experiments by Shrivastava et al. [9] have revealed that personality traits can also have an effect on debugging strategies, but these are subjective and context dependent.

Mature methods have also been researched to learn program comprehension in the process of debugging. The eye-tracking method employed by Mohammed et al. [10] to examine the attention of developers to error-prone regions of code has a limitation of scalability due to the need to have specialized equipment. Storey et al. [11] gave a more general view of program understanding, as an important variable in debugging performance, but their results are generalized. Mohanani et al. [12] investigated the phenomenon of cognitive biases in software engineering and identified that cognitive bias may adversely affect the process of debugging, whereas Henley and Fleming [13] empirically established that debugging tasks impose a strong cognitive load, which needs to be addressed by more intuitive tools.

The issues of debugging have also been highlighted by human-centered views. Myers et al. [14] talked about the complexity of debugging processes and the need to enhance support systems and Pressman [15] reiterated the significance of debugging in the software development lifecycle. Fritz et al. [16] evaluated the difficulty of the tasks with the help of psycho-physiological measures, which proves that debugging is one of the most challenging processes in software development. Ko and Myers [17] proposed a more interactive technique of debugging whereby the method is founded on questioning techniques that enhance understanding but demand more effort by the user. Murphy-Hill et al. [18] have examined the correlation between refactoring and debugging, revealing that code modification can affect the efficiency of debugging, but they only consider the reorganization of the code. Brooks [19] pointed to wider software engineering issues, the complexity of the processes of development.

To overcome these limitations, the notion of multimodal debugging has been proposed more recently. The foundational research [20] suggested a voice-assisted Python debugging system that uses a combination of audio and visual feedback as a way of enhancing the understanding of errors and their accessibility. Although the system showed great gains in debugging efficiency, it failed to test the performance in various development environments. Such a restriction is what makes it necessary to conduct a comparative analysis of such systems in popular integrated development environments, which is the core of the proposed study.

III. SYSTEM DESIGN & IMPLEMENTATION

The proposed system will improve the debugging process and combine voice-assisted feedback with conventional visual debugging methods. It aims at enhancing the understanding of errors, lessening cognitive burden, and facilitating efficient debugging with a multimodal method. The methodology integrates structured error management, intelligent suggestion generation and performance monitoring into a single structure. The general strategy is presented in subsections below to give a clear picture of the system design and how it will be implemented in terms of system architecture, data set up and working framework.

A. System Architecture

The suggested system is a voice-assisted multimodal debugging system that can improve the error detection and interpretation process of Python applications. Fig. 1 shows the architecture of the system.

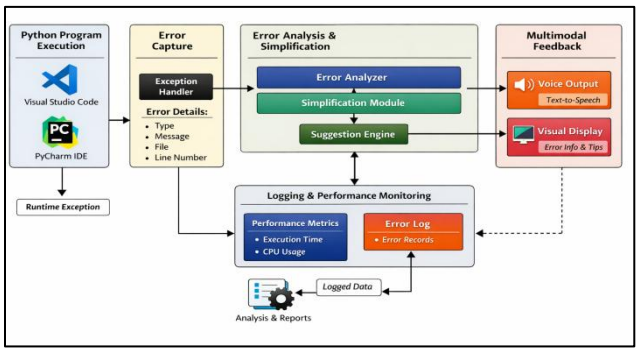

Fig. 1: System Architecture Diagram

The architecture is modular in design and comprises interconnected modules that capture, analyze, generate feedback, and evaluate performance. The cycle starts with the running of a Python program in the development environment. In the case of a runtime exception, the exception is handled with a specialized exception handling mechanism that is built into the system. This mechanism puts the detailed information about the error such as its type, message, file location, and line number.

When the error is captured, it is sent to the analysis module that manipulates the raw error information to determine the type and cause of the exception. An error message is further simplified to a more comprehensible form by a simplification layer, minimizing the technical description complexity. This allows developers to understand the problem without having to spend a lot of time interpreting stack traces.

The information that is processed is then relayed to the output modules, which produce auditory as well as visual feedback. The voice module translates the error information into speech, where developers can get immediate feedback without necessarily having to look at the information. At the same time, the graphical interface shows organized error information and recommended resolutions, which is a complete debugging experience.

Besides generating feedback, the system also has a logging and performance assessment module that logs the time and CPU use when a debugging process is in progress. These measures are employed to assess the efficiency of the system and help make comparative analysis in various development settings.

B. Data Source and Test Environment.

A set of predefined Python test cases are used to evaluate the system, which are meant to represent common runtime errors, including division by zero, out-of-range index access, key errors, and type mismatches. These test cases serve as the input data to the system and are run in both development environments to make sure that there is consistency in the evaluation. The experiments are carried out on the two popular integrated development environments, Visual Studio Code and PyCharm. The test cases are run in both environments with the same conditions and it is possible to compare the performance metrics of execution time and CPU utilization fairly. The data obtained is organized in a structured form to be further analyzed and visualized.

C. Proposed Framework

The suggested framework incorporates various functional elements in a cohesive mechanism to enhance the efficiency of debugging. It starts by initializing the debugging environment, in which a custom exception handler is registered to detect runtime exceptions. When the program is executed, an exception will be raised that will call the handler and retrieve the necessary debugging information. The framework then uses a simplification process to convert complex error messages to user friendly descriptions. This is an important step towards lessening the mental load and facilitating quicker understanding. Subsequently, a suggestion engine is created to offer context-related suggestions on the basis of the detected error type, helping developers to resolve problems more efficiently.

The system has multimodal feedback in form of voice and graphical feedback. The voice module involves a text-to-speech engine to relay the error information orally, whereas the graphical interface displays the error information and suggestions in detail. This mixture of modalities improves user interaction and facilitates a variety of preferences in debugging. Moreover, the framework has a performance monitoring mechanism that captures the time and CPU usage of execution at every debugging session. These measurements are applied to measure the performance of the system and provide comparative analysis among development environments. The framework is designed in a modular way, thus making it flexible and easy to integrate with other IDEs.

D. Voice Assisted Debugging System Algorithm.

The overall working of the proposed system is defined through the following algorithm:

1. Start the voice-assisted debugging system.
2. Hook in the custom exception handler via system-level hooks.
3. Start performance measurement (time and CPU usage).
4. Execute the Python program in the selected development environment.
5. Identify runtime exceptions in execution.
6. Log error information such as error type, message, file name and line number.
7. Reduce the error message to simple user-friendly format.
8. Make context-based recommendations with the suggestion engine.
9. Translate the error message into a speech with the voice module.
10. Present information, detailed error and recommendations with the help of the graphical interface.
11. Terminate performance measurement and compute execution time and CPU consumption.
12. Record the error details and performance measurements in structured files.
13. Repeat the procedure with various test cases and environments.
14. Compare data collected in terms of performance.
15. Terminate the debugging.

IV. RESULTS AND DISCUSSION

The effectiveness and performance of the proposed voice-assisted debugging system was tested by a set of controlled experiments performed on a set of test cases. The performance indicators that are evaluated include the execution time and CPU consumption, the analysis of the graphical interface of the system and its usability. To gain an idea about their effect on the efficiency of debugging, a comparative analysis was conducted with two popular development environments, Visual Studio Code and PyCharm. The findings of these experiments are discussed and given in the following subsections.

A. Experimental Setup

The suggested voice-assisted debugging system was tested with several Python test cases that covered typical runtime errors, such as division by zero, out-of-range index access, key errors, and type errors. All the test cases were run in two development environments, Visual Studio Code, and PyCharm, in the same conditions to guarantee consistency in assessment.The system noted the performance measures like time of execution and CPU consumption of each test case. These measures were kept in a tabular form and compared later. Besides the performance evaluation, the graphical user interface created by the system was analyzed to determine the usability and effectiveness in displaying debugging information.

B. Analysis of Time of execution.

Each test case was timed to determine the execution time of the debugging system in both development environments. The findings reveal that there were observable differences in the performance of the two environments.

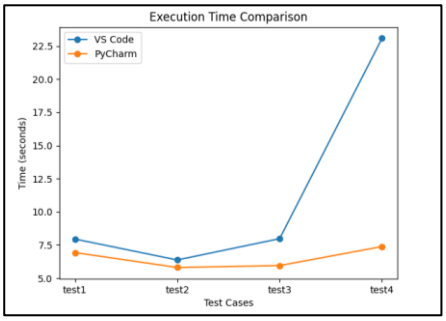

Fig. 2. Execution Time Comparison of VS Code and PyCharm

Fig. 2 shows the execution time of each test case. As can be seen, PyCharm is generally shown to have less execution time than the Visual Studio Code in most situations. Visual Studio Code, however, is more variate, especially in complicated situations, like test case 4, where the execution time grows considerably. This difference can be explained by the differences in internal processing and environment processing.

C. CPU Usage Analysis.

The CPU usage of the system when debugging was also assessed to comprehend resource usage in both environments.

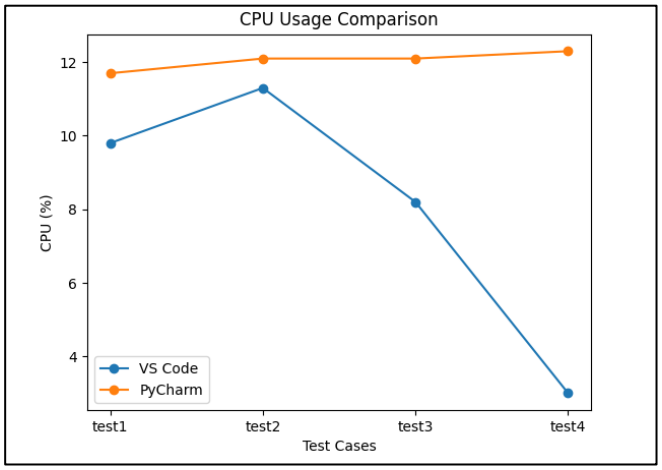

Fig. 3. CPU Usage Comparison of VS Code and PyCharm

PyCharm has a relative higher CPU usage in all the test cases as indicated in Fig. 3, which means that it takes more system resources to run its programs. However, Visual Studio Code has proven to be less resource-intensive and thus, less CPU-intensive. This performance-resource consumption trade-off shows the disparities in the design and execution behavior of both environments.

D. Comparison of Performance on average.

To have a better view of the overall system behavior, the average execution time and the CPU usage were computed in all test cases.

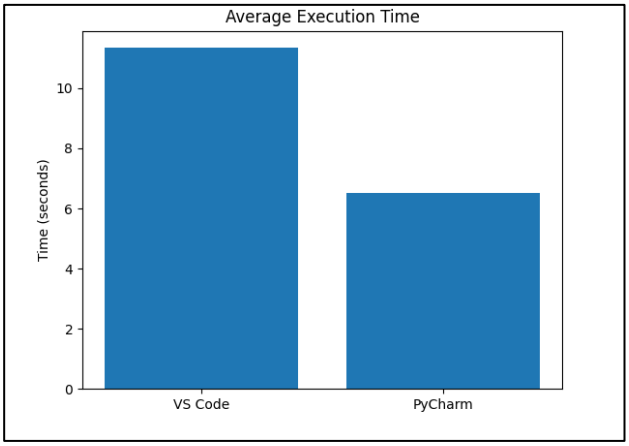

Fig. 4. Average Execution Time Comparison

The average execution time of both environments is shown in Fig. 4. The findings validate that on average PyCharm is faster in execution, which means it is faster in terms of speed.

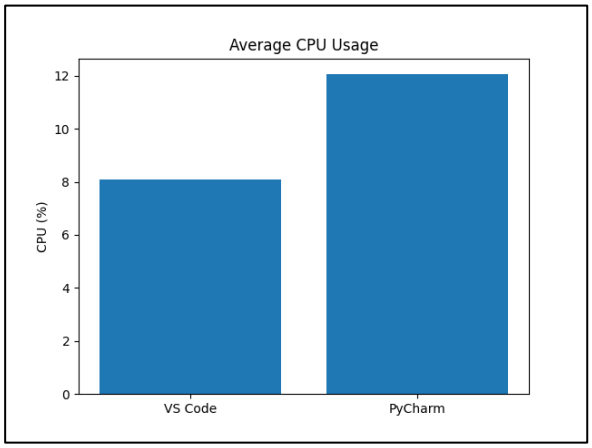

Fig. 5. Average CPU Usage Comparison

The average CPU usage is presented in Fig. 5, with PyCharm registering higher values than Visual Studio Code. This supports the fact that enhanced performance in PyCharm is associated with the high cost of consuming more resources.

E. Graphical User Interface Evaluation.

Besides performance metrics, the usability of the system was measured with regard to the graphical interface, where the details of the error and the recommendations are displayed in a structured format.

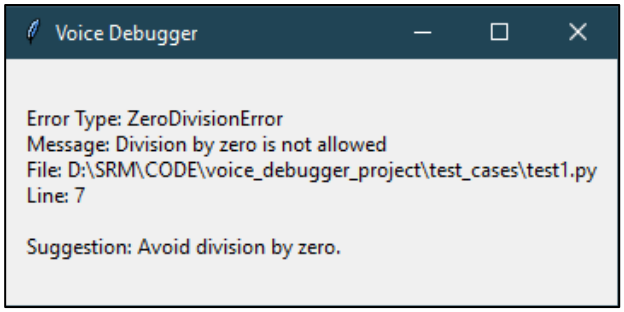

Fig. 6. GUI Output for Test Case 1 (ZeroDivisionError)

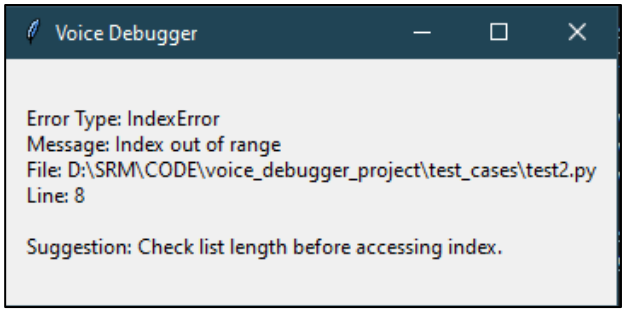

Fig. 7. GUI Output for Test Case 2 (IndexError)

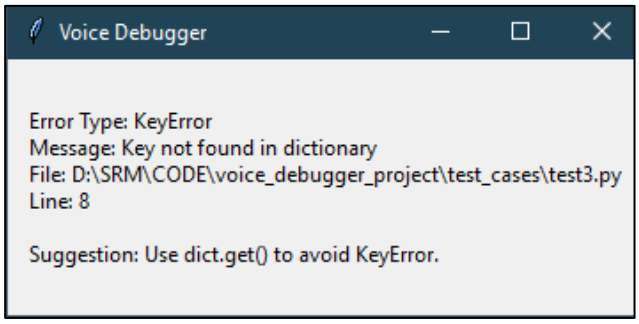

Fig. 8. GUI Output for Test Case 3 (KeyError)

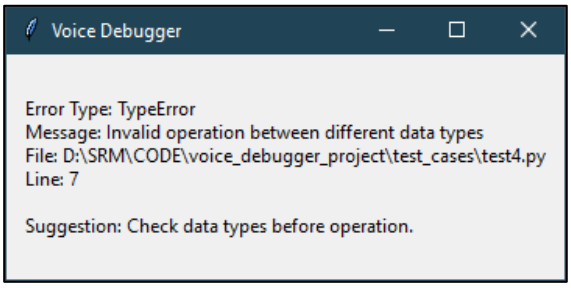

Fig. 9. GUI Output for Test Case 4 (TypeError)

The graphical results depicted in Fig. 6 to Fig. 9 indicate that the system can present debugging information in an easy to understand and read way. The error type, message, file location, line number and recommended solution is shown in each interface. This hierarchical display minimizes the amount of work needed to understand the errors and enhances the overall debugging experience.

F. Discussion

The experimental findings indicate that the suggested voice-assisted debugging system is effective in improving the understanding of errors and efficiency of debugging. The combination of auditory feedback enables developers to obtain real-time information without having to inspect only the visual content, which will lighten cognitive load.

Performance wise, the PyCharm has a faster execution time, and it is therefore appropriate in the situation where speed is a factor. Nonetheless, it utilizes a larger amount of CPU resources, which can affect performance in resources-limited environments. Visual Studio Code, on the other hand, uses fewer resources in terms of CPU usage, and can thus be considered a more lightweight and efficient package, though in some instances this can have increased execution time.

The multimodal feedback and performance assessment allow us to gain lots of information about the trade-offs among the various development settings. The findings are quite evident that the two environments have different performance characteristics that are useful with the proposed system, whereby developers can select the most appropriate platform depending on their needs.

V.DISCUSSION

The outcomes of the experimental assessment give valuable information on the efficiency of the developed voice-assisted debugging system. Multimodal feedback, especially the combination of auditory and visual feedback, can greatly enhance the interaction of the developers with the error information. The system simplifies the cognitive load needed to interpret and fix problems by simplifying complex error messages into simplified speech and structured graphical displays.

The analysis of the performance indicates a definite difference between the two development environments. PyCharm shows a low execution time in most test cases, which implies its effectiveness in dealing with debugging processes. This is due to its optimized and built in debugging framework. Nevertheless, this better performance is at the expense of increased CPU usage, which implies increased consumption of resources during execution.

On the other hand, Visual Studio Code has a reduced CPU usage, which is why it is a more lightweight and resource-efficient environment. Although this is an advantage, it is more variable in the execution time especially in more complex situations. This action demonstrates the trade-off between speed and resource efficiency in choosing a development environment to debug tasks.

The other notable observation is that the graphical user interface has improved its usability. The systematized display of the error information along with the appropriate recommendations allow quicker comprehension and less reliance on the external sources. Voice feedback also makes it more accessible, as developers do not need to monitor the screen at all times to get real-time information.

In general, the discussion points out that the proposed system not only enhances efficiency in debugging, but also offers useful information about performance of various development environments. The results reinforce the notion that multimodal interaction may be important in contemporary debugging tools.

VI. CONCLUSION

This study introduced a voice-based multimodal debugging system that aimed to better understand errors and increase the efficiency of debugging Python programs. The system combines exception handling, speech synthesis, graphical visualization, and performance monitoring into a single system, making the debugging experience more intuitive and user-friendly.

The experimental analysis showed that the suggested system is efficient in decreasing the complexity of interpreting errors by integrating auditory and visual feedback. The findings also revealed that Visual Studio Code and PyCharm had some differences in performance with the latter being faster and the former being more efficient in terms of resource consumption. These results underline the need to choose the right development environments depending on the particular requirements.

The offered solution can contribute to the sphere of software engineering as it leads to the introduction of a convenient and effective debugging tool that corresponds to human cognition. The system is more productive to the developers by minimizing cognitive load and facilitating interaction.

The next step in work may be to implement more sophisticated features to the system like machine learning-based error prediction, real-time debugging support, and other programming language support. The usefulness of the proposed system can also be reinforced by further assessment using bigger data sets and practical implementations.

Conflict of Interest

The authors declare no conflicts of interest regarding the current research.

References

[1] H. Li and M. Coblenz, “A Grounded Theory of Debugging in Professional Software Engineering Practice,” arXiv preprint arXiv:2602.11435, 2026.

[2] F. Stolp et al., “Using CognitIDE to Capture Developers' Cognitive Load via Physiological Activity During Everyday Software Development Tasks,” arXiv preprint arXiv:2503.03537, 2025.

[3] A. C. Tajimalela et al., “Comparison Towards Different Methods of Software Debugging,” Journal of Software Engineering, Information and Communication Technology, 2024. :contentReference[oaicite:0]{index=0}

[4] X. Gao and K. F. Hew, “A Flipped Systematic Debugging Approach to Enhance Program Debugging Performance and Optimize Cognitive Load,” Educational Technology Research and Development, 2023. :contentReference[oaicite:1]{index=1}

[5] C. Sun, S. Yang, and B. Becker, “Debugging in Computational Thinking: A Meta-analysis on the Effects of Interventions on Debugging Skills,” Journal of Educational Computing Research, 2024. :contentReference[oaicite:2]{index=2}
[6] R. McCauley et al., “Debugging: A Review of the Literature from an Educational Perspective,” Computer Science Education, vol. 18, no. 2, pp. 67–92, 2008. :contentReference[oaicite:3]{index=3}
[7] M. Lanza, R. Robbes, and M. Lungu, “Patterns of Developers’ Behaviour: A 1000-Hour Industrial Study,” Journal of Systems and Software, 2017.
[8] I. A. Khan, W. P. Brinkman, and R. M. Hierons, “Do Moods Affect Programmers’ Debug Performance?” Cognition, Technology & Work, vol. 13, no. 4, pp. 245–258, 2011. :contentReference[oaicite:4]{index=4}
[9] S. Shrivastava, S. Maurya, and A. Kumar, “Debugging Behavior in Software Engineers Based on Personality Dimensions,” TPM – Testing, Psychometrics, Methodology in Applied Psychology, 2025. :contentReference[oaicite:5]{index=5}
[10] A. Mohammed et al., “An Empirical Eye-Tracking Study of Program Comprehension and Debugging,” Information and Software Technology, 2026. :contentReference[oaicite:6]{index=6}
[11] M. A. Storey et al., “How Do Programmers Understand Code? A Survey of Program Comprehension,” IEEE Transactions on Software Engineering, 2005.
[12] R. Mohanani et al., “Cognitive Biases in Software Engineering: A Systematic Mapping Study,” IEEE Transactions on Software Engineering, 2017.
[13] A. Z. Henley and S. Fleming, “Cognitive Load While Debugging: An Empirical Study,” IEEE Symposium on Visual Languages and Human-Centric Computing, 2016.
[14] B. A. Myers et al., “The Debugging Process and Its Challenges: A Human-Centered Perspective,” Communications of the ACM, 2016.
[15] R. S. Pressman, “Software Engineering: A Practitioner’s Approach,” McGraw-Hill, 2014.
[16] T. Fritz et al., “Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development,” ICSE, 2014.
[17] A. J. Ko and B. A. Myers, “Debugging Reinvented: Asking and Answering Why and Why Not Questions,” ICSE, 2008.
[18] G. Murphy-Hill et al., “How We Refactor, and How We Know It,” IEEE Transactions on Software Engineering, 2012.
[19] F. P. Brooks, “The Mythical Man-Month: Essays on Software Engineering,” Addison-Wesley, 1995.
[20] “Hear Your Code Fail: Voice-Assisted Debugging for Python,” 2025.