Clarity: An AI-Powered Visual Studio Code Extension for Code Insights and Smart Comments

Richmond Lavadia, Christian Remoh Mappatao, Niño Israel Pajarillo, Rey Christian Sumeran

Information Technology Education Program
School of Engineering, Architecture, and Information Technology Education, University of Saint Louis
Tuguegarao City, Cagayan

***Abstract*— Clarity, an AI-powered Visual Studio Code extension, presents an innovative approach to enhancing code quality and fostering maintainable software development. Leveraging the capabilities of large language models (LLMs), Clarity provides context-aware suggestions for code improvements, including variable naming, function naming, and code comments. The results revealed a significant improvement in the quality of the code produced by the participants, as measured by readability, maintainability, and reduced bug count. Clarity stands out as a valuable tool for software developers seeking to enhance the clarity and maintainability of their code. Its AI-powered capabilities empower developers to write more efficient, bug-free, and readable code, ultimately leading to improved software quality and reduced development time.**

***Keywords*— AI-powered coding assistant; code insights; smart comments; Visual Studio Code extension; software development**

I. Introduction

Adhering to coding conventions is crucial for achieving efficient and high-quality software. By following these conventions, programmers can easily comprehend shared code, communicate effectively, and maintain the source code at minimal expenses. Despite this, many developers encounter difficulties in producing code that is clear, concise, and easily maintainable.

Recent studies reveal a nuanced perspective on software development and maintenance. Studies have shown that the quality of documentation significantly impacts how well developers understand a program. Poor documentation makes it harder for developers to grasp the program's functionality, leading to increased effort and time spent on maintaining the software [1]. Building on this point, research suggests that thorough code reviews are hampered by the significant time it takes for reviewers to fully grasp the code's purpose and any changes made [2]. Understanding methods, variables, and changes can be overwhelming for reviewers. Simplifying these concepts can help reduce the strain on their cognitive load. These time constraints in acquiring contextual understanding compromise software quality, as thorough analysis is crucial for effective reviews.

Recent research highlights that both how easy code is to understand (readability) and how intricate it is (complexity) are critical factors in determining the overall quality of software. These factors significantly influence how easily the code can be reused and maintained in the future [3]. The research also emphasizes that during the maintenance phase, a high percentage of the software lifecycle cost is allocated. Adding to these points, researchers have identified challenges in accurately measuring code quality. One key difficulty is the absence of a single, universally agreed-upon set of metrics for evaluation[[4]. Their study shows that developers consistently prioritize structure, readability, and documentation as crucial for creating code that is easy to understand and maintain, pointing to the need for a broader understanding of factors contributing to software excellence.

To enhance code comprehension and foster a seamless development experience, researchers and developers have explored various tools and techniques to assist programmers throughout the coding process. Recent research explored the use of powerful language models in analyzing faulty code. The study showed that these models, similar to ChatGPT, can generate explanations for errors, potentially offering valuable assistance to developers [5]. A study explored using a powerful AI tool (like OpenAI's Codex) to generate programming exercises and explanations. The results showed the tool's potential to create new and practical learning materials, even providing some exercises that could be used directly [6]. While these studies showcased the capabilities of language models for programming assistance, none of them streamlined the process for timely and seamless usage.

Github's Copilot, which also leverages Codex, introduced a cloud-based and extension tool for generative programming, studies have shown that developers can struggle to understand, modify, and fix errors in the code Copilot suggests. This can actually hinder their ability to solve tasks effectively [7]. Furthermore, several VS Code extensions in the market such as the Cody and GPTutor have utilized OpenAI’s GPT 3.5 LLM to generate code explanations, However the results generated are not always contextualized or sometimes not closely aligned with the code's unique context.

To address these challenges, the researchers developed Clarity, an AI-powered Visual Studio Code (VS Code) extension for generation of contextualized code insights and smart comments. This software extension seamlessly integrates with the IDE, streamlining the coding and learning processes for developers. Through Clarity, developers can ensure well-documented code by generating smart comments, complementing other generative programming tools. By providing a deeper understanding of codes and aiding in error debugging, Clarity aims to enhance the overall coding experience for developers and improve software quality.

II. Methods

A. *Software Architecture*

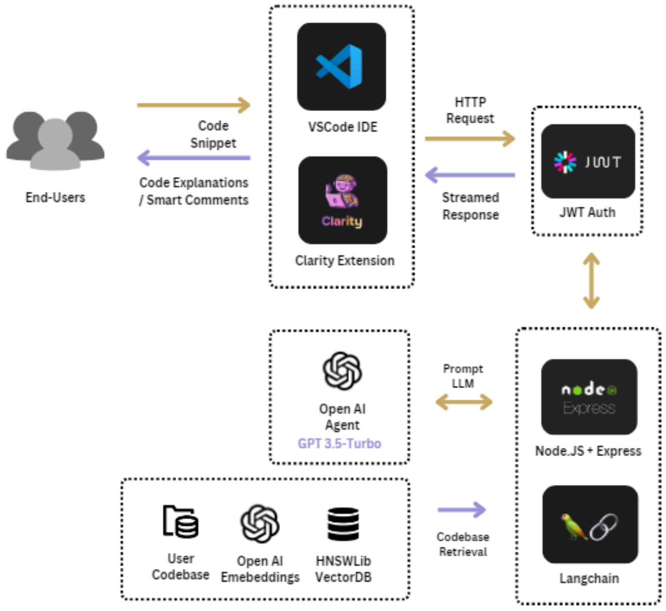


Fig. 1. High-Level Software Architecture

Clarity seamlessly integrates into the workflow, offering an intuitive user experience and effortless code analysis. It prioritizes security with a robust authentication layer using JSON Web Tokens, ensuring secure communication when interacting with the extension. The powerful backend built with Node.js and Express efficiently handles code snippets, while the LangChain library delves deep into their semantics for accurate insights. Clarity supports a wide range of programming languages, allowing developers comfortable in various languages to leverage its functionalities. It utilizes embeddings to retrieve relevant files from the codebase, ensuring the analysis considers the broader context for even richer insights. With Clarity, users gain a deeper understanding and enhanced workflow for coding endeavors. At the heart of Clarity lies the OpenAI Agent, leveraging the cutting-edge capabilities of GPT-3.5 Turbo. This advanced language model analyzes code semantically, not just syntactically, for deeper understanding.

B. *Software Flowchart*

Figure 2 presents the system flowchart that was followed in this study. This flowchart outlines the process that was used by the extension in generating code insights and smart comments for the end-users.

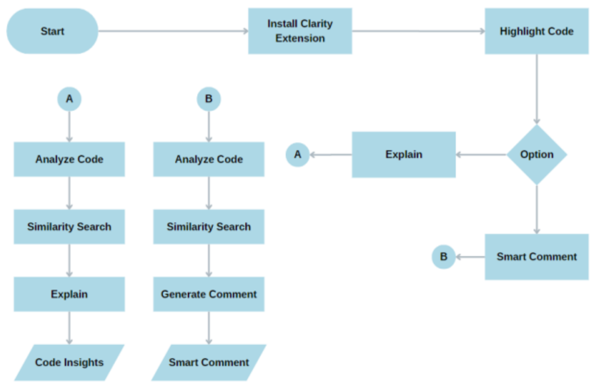


Fig. 2. Clarity Process Flowchart

C. *Extension Development*

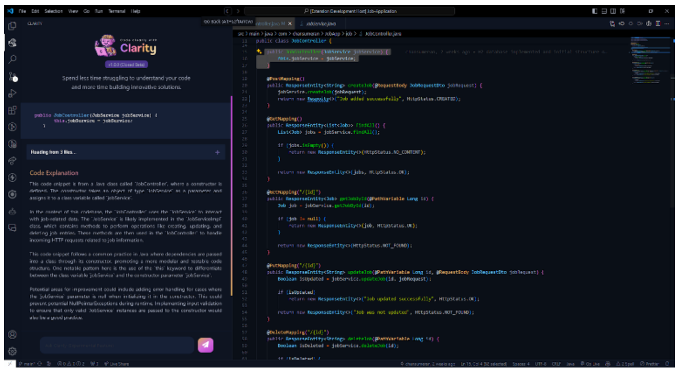
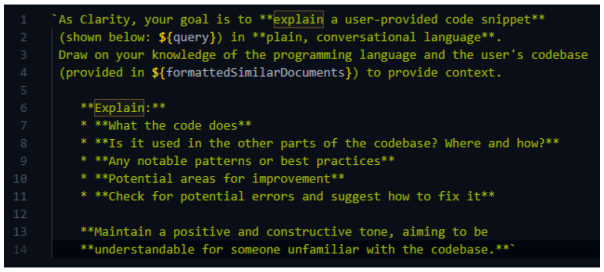


Fig. 3. Clarity Extension



```
`As Clarity, your goal is to **explain a user-provided code snippet**
 (shown below: ${query}) in **plain, conversational language**.
 Draw on your knowledge of the programming language and the user's codebase
 (provided in ${formattedSimilarDocuments}) to provide context.

    **Explain:**
    * **What the code does**
    * **Is it used in the other parts of the codebase? Where and how?**
    * **Any notable patterns or best practices**
    * **Potential areas for improvement**
    * **Check for potential errors and suggest how to fix it**

    **Maintain a positive and constructive tone, aiming to be
    **understandable for someone unfamiliar with the codebase.**`
```

Fig. 4. Prompt used for generating code insight

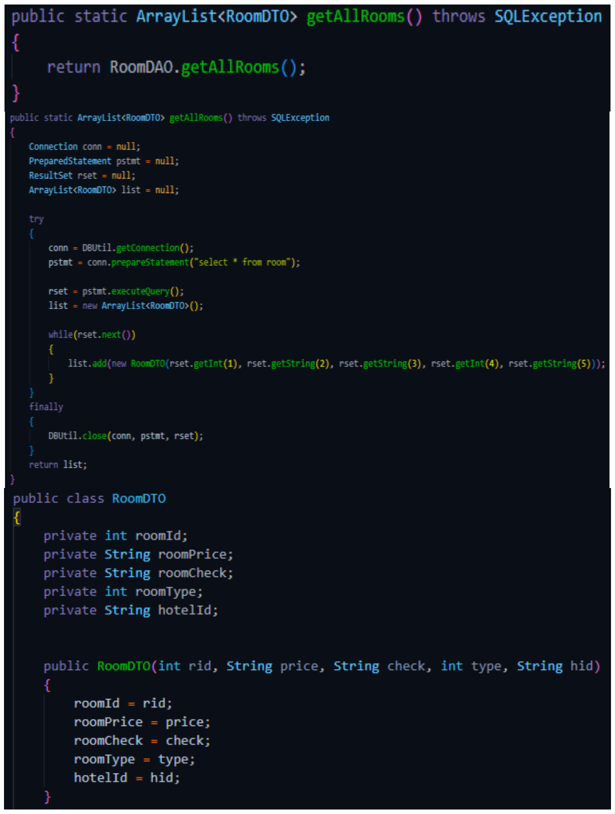
The Clarity extension was developed within the Visual Studio Code environment, leveraging its rich API and TypeScript as the primary programming language. This choice enabled seamless integration with existing developer workflows, ensuring a familiar and intuitive user experience.

The extension communicates with a robust backend powered by Node.js and the Express framework, providing a reliable and efficient foundation for handling code analysis requests. This architecture ensures a responsive and scalable platform for the Clarity service.

To generate insightful code comments and explanations, Clarity leverages a large language model (LLM) through a carefully constructed prompting mechanism. This approach guides the LLM's attention towards specific code elements and desired analysis tasks, ensuring contextually relevant and informative output. The prompts used are shown in Figure 4.

III. Results and Discussion

A comparative analysis of explanations provided by GPTutor shown in Figure 6, Github Copilot shown in Figure 7, and Clarity shown in Figure 8 for the RoomDAO.getAllRooms() method presented in Figure 5 reveals distinct characteristics despite offering a fundamental understanding.



```
public static ArrayList<RoomDTO> getAllRooms() throws SQLException
{
    return RoomDAO.getAllRooms();
}
public static ArrayList<RoomDTO> getAllRooms() throws SQLException
{
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rset = null;
    ArrayList<RoomDTO> list = null;

    try
    {
        conn = DBUtil.getConnection();
        pstmt = conn.prepareStatement("select * from room");

        rset = pstmt.executeQuery();
        list = new ArrayList<RoomDTO>();

        while(rset.next())
        {
            list.add(new RoomDTO(rset.getInt(1), rset.getString(2), rset.getString(3), rset.getInt(4), rset.getString(5)));
        }
    }
    finally
    {
        DBUtil.close(conn, pstmt, rset);
    }
    return list;
}
public class RoomDTO
{
    private int roomId;
    private String roomPrice;
    private String roomCheck;
    private int roomType;
    private String hotelId;


    public RoomDTO(int rid, String price, String check, int type, String hid)
    {
        roomId = rid;
        roomPrice = price;
        roomCheck = check;
        roomType = type;
        hotelId = hid;
    }
```

Fig. 5. RoomDAO.getAllRooms() method

Legends	***Correct, Irrelevant / Vague, Unique***
GPTutor v1.2.3	RoomDAO.getAllRooms(); is a method call that retrieves and returns a list of RoomDTO objects representing all the rooms from the database. This method is expected to interact with the data access layer to fetch the room data from the underlying database storage. The RoomDAO class likely contains the necessary database querying logic to fetch the room information and transform it into a list of RoomDTO objects, which are then returned for further processing or display.

Fig. 6. Response of GPTutor

Legends	***Correct, Irrelevant / Vague, Unique***
Github Copilot v0.14.2024030604	The selected code is a method in Java, specifically from a service class in a typical Java application that follows the DAO (Data Access Object) design pattern. The method getAllRooms() is declared as public and static, meaning it can be accessed from anywhere in your application without needing an instance of the class it's in, and it returns an ArrayList of RoomDTO objects. RoomDTO (Data Transfer Object) is a simple object that carries data between processes, in this case, probably between the database and the application. The getAllRooms() method is designed to interact with the RoomDAO class, which is likely responsible for the actual database operations. The getAllRooms() method in RoomDAO is expected to return an ArrayList of RoomDTO objects, representing all the rooms in the database. The throws SQLException clause in the method declaration indicates that this method may throw an SQLException. This is an exception that provides information on database access errors or other errors. By declaring this in the method signature, it's made clear that whoever calls this method will need to handle this potential exception.

Fig. 7. Response of Github Co-pilot

Legends	***Correct, Irrelevant / Vague, Unique***
Clarity v1.0.0	This code snippet is part of a Java application for managing hotel rooms. It's designed to fetch all rooms from the database and return them as a list of RoomDTO objects. Here's a breakdown of what's happening: Method Explanation: The provided code defines a method named getAllRooms() within the RoomDAO class. This method has a return type of ArrayList<RoomDTO>, indicating that it will return a list of RoomDTO objects. Database Interaction: Inside the getAllRooms() method, there's interaction with the database: It establishes a database connection using the DBUtil.getConnection() method. It prepares a SQL query to select all records from the room table using conn.prepareStatement("select * from room"). It executes the query using pstmt.executeQuery() and obtains the result set. It iterates through the result set, creating RoomDTO objects for each row fetched from the database and adding them to an ArrayList. RoomDTO Usage: The RoomDTO class represents the data structure for a room. It has attributes like roomId, roomPrice, roomCheck, roomType, and hotelId, along with getter and setter methods to access and modify these attributes. In summary, the getAllRooms() method fetches all room data from the database and returns it as a list of RoomDTO objects.

Fig. 8. Response of Clarity

GPTutor's explanations miss details and provide a vague response. While Github Copilot accurately describes the code's functionality, though a minor vague response is provided. It may also overlook broader context and best practices. Clarity distinguishes itself by providing comprehensive and thematic explanations, including database interaction steps and usage of the RoomDTO class. It emphasizes best practices, offers simplified summaries potentially beneficial for beginners, and uniquely accesses external files. This comprehensive and context-aware approach positions Clarity as a potentially valuable tool for developers seeking deeper code comprehension.

IV. Conclusion

This research successfully developed and evaluated Clarity, an AI-powered Visual Studio Code extension designed to empower developers with deeper code comprehension and a more streamlined development experience. The comparative analysis revealed Clarity's distinct advantage in offering comprehensive and context-aware explanations. Its ability to delve into database interactions, external file usage, and best practices makes it a valuable tool for developers of all experience levels.

This research suggests several promising directions for future work. First, future iterations of Clarity can address the usability aspects identified by IT professionals. By incorporating user feedback and conducting usability testing, the extension's interface and functionalities can be further refined to enhance user experience. Additionally, exploring alternative prompting techniques for the LLM could potentially yield even more comprehensive and informative code explanations. Furthermore, research into the long-term impact of AI-powered code comprehension tools on developer skill development is warranted. It would be valuable to investigate how tools like Clarity influence a developer's ability to independently understand and write code. Finally, the ethical implications of AI-assisted coding, such as potential biases within the LLM or over-reliance on automated explanations, should be explored to ensure responsible development and use of these technologies.

References

[1] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope (2018). “The effect of poor source code lexicon and readability on developers’ cognitive load,” Proceedings of the 26th Conference on Program Comprehension. ACM. Retrieved from https://doi.org/10.1145/3196321.3196347

[2] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli (2018). “Information Needs in Contemporary Code Review,” Proceedings of the ACM on Human-Computer Interaction, vol. 2, no. CSCW. Association for Computing Machinery (ACM), pp. 1–27. Retrieved from https://doi.org/10.1145/3274404

[3] Y. Tashtoush, N. Abu-El-Rub, O. Darwish, S. Al-Eidi, D. Darweesh, and O. Karajeh (2023). “A Notional Understanding of the Relationship between Code Readability and Software Complexity,” Information, vol. 14, no. 2. MDPI AG, p. 81. Retrieved from https://doi.org/10.3390/info14020081

[4] J. Börstler et al., “Developers talking about code quality (2023)” Empirical Software Engineering, vol. 28, no. 6. Springer Science and Business Media LLC, https://doi.org/10.1007/s10664-023-10381-0

[5] Tian, H., Lu, W., Li, T. O., Tang, X., Cheung, S.-C., Klein, J., & Bissyandé, T. F. (2023). Is ChatGPT the Ultimate Programming Assistant -- How far is it? (Version 1). arXiv. Retrieved from https://doi.org/10.48550/ARXIV.2304.11938

[6] Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations using Large Language Models. arXiv. Retrieved from https://doi.org/10.48550/ARXIV.2206.11861

[7] Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In CHI Conference on Human Factors in Computing Systems Extended Abstracts. CHI ’22: CHI Conference on Human Factors in Computing Systems. ACM. Retrieved from https://doi.org/10.1145/3491101.3519665