ICGHIT

International Conference on
Green and Human Information Technology 2025

ICGHIT 2025

Jan.15 – 17, 2025
Nha Trang, Vietnam

Proceedings

5, 2024
am

GHIT IEIE SsangYong Information & Communications Corp.

http://icghit.org/

ISSN : 2466-121X

AI-Driven Code Generation Mechanism

Yejin Jin *, Chaeyun Seo †, Kidu Kim ‡, R. Young Chul Kim §
*†§ *Software Engineering Laboratory, Hongik University, Sejong, Korea*
‡ *Telecommunications Technology Association, City, Korea*
Emails: *yejin_jin@g.hongik.ac.kr, †chaeyun@hongik.ac.kr, ‡kdkim@tta.or.kr, §bob@hongik.ac.kr

***Abstract*— Recently, generative AI has become a tool in code generation that allows developers to generate code through simple queries easily. But it isn't still clear that the generative AI tools generate code with input prompts. This means why we don't know inside the process of code generation as a "black box." As we are software engineers, we raise concerns about the reliability and verifiability of generated code. There are limited existing methods for verifying the results of generative AI. We need a more systematic approach. We propose a step-by-step approach based on the software development life cycle (SDLC) to uncover the black box of AI-based code generation. We aim to increase the understanding and reliability of generative AI in software development by breaking down the process into individual steps. We also explore how this approach can be applied to multiple stages of the development process to make AI-based code generation more systematic and reliable.**

Keywords—LLMs, Natural Language Requirements, Code Generation, UML Diagram

I. INTRODUCTION

Recently, generative AI technology has been actively utilized in the field of code generation as well as development. Many users can easily generate code with simple queries by utilizing generative AI. However, the specific mechanism by which generative AI generates results from prompt input is not revealed, and this process is often expressed as a "black box." This black box characteristic raises issues regarding the reliability and verifiability of the results. Existing methods for verifying the black box of generative AI are limited, and a systematic approach is needed to solve this. Based on the flow of the software development life cycle (SDLC), the method of performing verification the use of generative AI can track the step-by-step process of code generation. This will not only allow a more intuitive understanding of the process of generating code from prompts but also increase the reliability of the results by applying existing verification technologies.

We propose a step-by-step approach based on the software development life cycle (SDLC) to reveal the black box of the generative AI-based code generation mechanism. We examine how generative AI can be utilized in the software development stage, and propose methods applied to code generation using generative AI at each stage. Through this, we aim to provide a foundation for designing the utilization of generative AI more systematically and reliably, thereby increasing the convenience of code generation. In Chapter 2, this paper discusses related research and existing code generation methods, and in Chapter 3, we propose a generative AI code generation mechanism based on a step-by-step approach. Finally, in Chapter 4, we discuss the conclusions and limitations of the study, and suggest future research directions.

II. RELATED WORKS

A. 3D Object Extraction from Natural Language

AI has also been introduced into the 3D modeling field, and Text to 3D technology has emerged. Text to 3D AI can quickly generate 3D through natural language without complex modeling technology. However, generative AI has difficulty in ensuring consistency and quality of results. This previous study proposes a mechanism for generating 3D objects from unstructured requirements in the cartoon field from a software perspective[1]. It effectively analyzes the meaning of natural language requirements by integrating requirements engineering and linguistics theory. Figure 1 shows the process of this study.

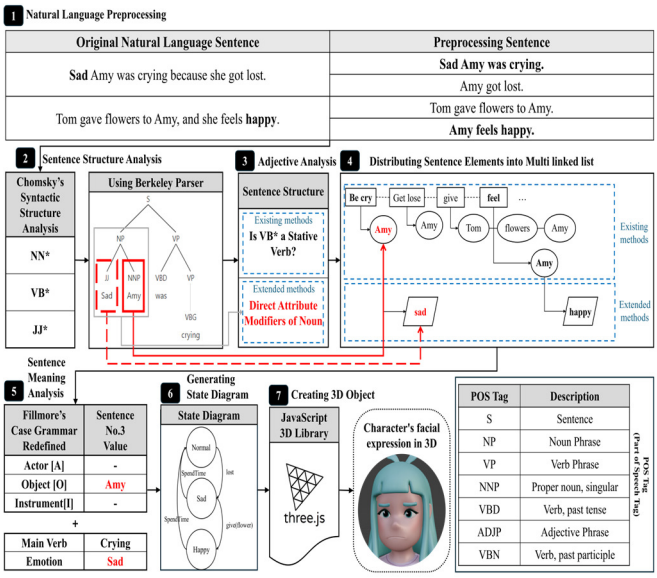

Fig. 1. 3D Object Extraction Process from Natural Language

We intend to apply software development with the step-by-step mechanism

B. Code Generation Research

Several approaches have been explored to automate and streamline the code development process. The three main approaches to generating code are based on the Software Development Life Cycle (SDLC), automated code generation tools, and generative AI technologies. Each approach has the following characteristics:

1) SDLC-based Code Generation

The Software Development Life Cycle (SDLC) is a structured methodology that divides the software development life cycle into various phases, including requirements analysis, design, implementation, testing, deployment, and maintenance[2]. This method follows a step-by-step approach, so each phase is addressed systematically. One of the main

advantages of SDLC-based code generation is the structured approach that ensures thorough development at each phase. This method allows for clear traceability, transparency, and accountability, allowing verification of the entire process. SDLC also facilitates detailed documentation and verification procedures that improve the reliability of the generated code. However, this method has limitations in that the process requires significant human intervention at each step, which is time-consuming.

2) Automated Code Generation Tools

Automated code generation tools can significantly reduce the amount of manual work required to write code for common or repetitive tasks[3]. They help improve development speed and maintain consistency in code structure when using predefined templates. However, while these tools can increase productivity, they often struggle to handle complex tasks because they rely on predefined templates or rules. Additionally, the generated code may not be optimal in some cases.

3) Generative AI for Code Generation

The rise of generative AI technologies, such as OpenAI's GPT model, has introduced a new approach to code generation. Generative AI can quickly generate code from simple user input in the form of prompts, using natural language processing (NLP), saving significant time and effort[4]. It can also generate code in multiple programming languages, increasing flexibility and usability. However, generative AI for code generation also has some notable limitations. AI-generated code can be difficult to understand for what it was generated, making debugging and maintenance more difficult. Furthermore, generative AI tools can sometimes generate incomplete or buggy code, requiring manual intervention to fix errors.

We use SDLC and generative AI technology-based approaches to complement each other's limitations and suggest ways to maximize their strengths.

III. Generating Code From Natural Language

We propose a mechanism that applies the GPT API and generates code from natural language requirements through a step-by-step process. UML design is generated through natural language requirement analysis, and code is generated from UML design using metamodeling techniques. Figure 2 shows the mechanism for the process of systematically analyzing natural language.

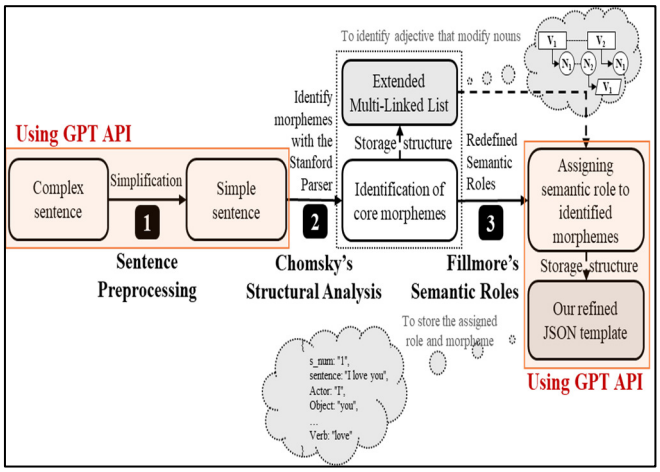

Fig. 2. Natural Language Analysis Procedure

A. Natural Language Requirements Analysis

To analyze natural language requirements, we go through three steps. First, we preprocess natural language sentences to make them easy to analyze. Next, we apply Chomsky's theory for grammatical analysis. Then, we apply Fillmore's linguistic theory for semantic analysis and provide the analysis results in JSON format. Although we cannot directly access the underlying model, we can automate the process through prompting. We use the Lang-chain technique with the goal of optimal natural language analysis. We illustrate our mechanism with a simple door lock system as an example, as shown below[5].

R1: The members can open the door through the door lock device.
R2: When the door is closed, the door lock device is locked.
...
R7: The unlock function can be unlocked using a password, IC card, or iris.
R8: When the user goes to the door, it must recognize the iris information of the user and unlock it.

The preprocessing process is performed to transform them into simple sentences. In this process, any missing subjects are replaced with the subjects of existing sentences, and pronouns are substituted with nouns that represent them. R8 is made up of both compound and complex sentences. If the R8 requirement is preprocessed according to this procedure, it is divided into three sentences as follows.

R8-1: The user goes to the door.
R8-2: The door must recognize the iris information of the user.
R8-3: The door must unlock the door.

B. Morphological-based requirements analysis

We perform grammatical analysis on the preprocessed requirement sentences. We analyze morphemes based on Chomsky's Syntactic Structural Analysis theory[6]. For this, we use Stanford Parser, which can identify the structure of sentences based on Chomsky's theory. It can distinguish the main verb, related nouns, and adjectives of the sentence. Requirement R8-1 ***'The user goes to the door.'*** is parsed by Stanford parser as ***'The(DT) user(NN) goes(VBZ) to(IN) the(DT) door(NN).'*** We treat parts of speech starting with NN as nouns and parts of speech starting with VB as verbs.

C. Semantic-based requirements analysis

Based on the analyzed morphemes, we use Fillmore's Semantic Roles theory to understand the meaning of sentences[7]. The theory assigns roles to nouns based on the main verb. Fillmore's theory has been studied and evolved by many linguists. We redefine it to be suitable for UML generation as shown in Table 1.

Table 1. Redefined Fillmore's Semantic Roles for UML

Roles	Definition
Actor	The entity that is the main subject of the event.
Element	The entity that indicates a property of the actor
Object	The entity affected by the event.

Source	The entity performing the action.
Target	The entity receiving the action.
Instrument	The tool used to perform the action.

If we analyze requirement sentences with the redefinition of roles as shown in Table 1, it is as follows:

R8-1: The user(Actor, Source) goes(Verb) to the door(Target).
R8-2: The door(Actor) must recognize(Verb) the iris information(Instrument) of the user(Object).
R8-3: The door(Actor) must unlock(Verb) the door(Object).

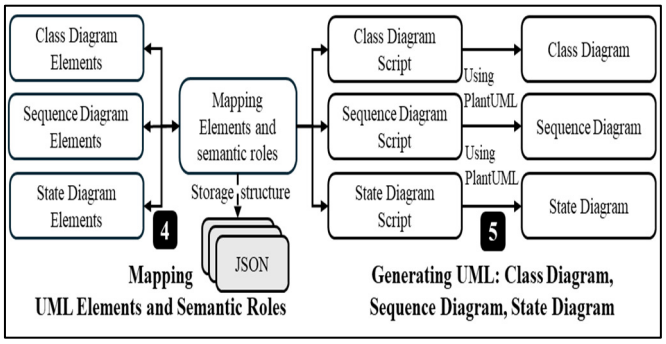

Fig. 3. Generating UML Diagram

A use case diagram is completed through cases analyzed from natural language [8]. Based on the use case, the results of natural language analysis and UML diagram elements are mapped to create a class diagram, sequence diagram, and state diagram.

D. Generating Code Template

Metamodeling is used to generate code based on the generated UML diagram. The items in the metamodel are mapped to each element of the class diagram, sequence diagram, and state diagram. As shown in Figure 4, a metamodel is generated for the elements of the diagram and the elements required for development, so that code can be generated through transformation rules.

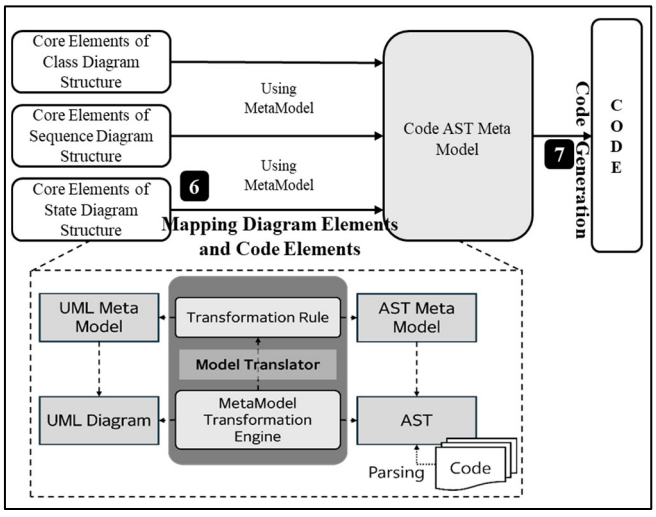

Fig. 4. Generating Code from UML Diagram

The diagram and code generated as natural language requirements are mapped, so that code that reflects the requirements is ultimately generated.

IV. Conclusion

In conclusion, We propose a step-by-step approach for understanding the black box of generative AI-based code generation mechanisms using an SDLC-based approach. By utilizing this approach, generative AI can be more effectively integrated into software development processes. This will bridge the gap between prompts and generated code, and further lay the groundwork for enhancing the reliability of generative AI in software engineering. However, while simple examples can be addressed, applying this method to complex scenarios still has limitations, necessitating further research. Future studies could expand this approach to more complicated scenarios and explore additional methods to improve the interpretability of generative AI systems.

Acknowledgment

This research was supported by Korea Creative Content Agency (KOCCA) grant funded by the Ministry of Culture, Sports and Tourism (MCST) in 2024 (Project Name: Artificial Intelligence-based User Interactive Storytelling 3D Scene Authoring Technology Development, Project Number: RS-2023-00227917307820870502O1) and National Research Foundation (NRF), Korea, under project BK21 Four.

References

[1] Y.J. Jin, C.Y. Seo, J.H. Kong and R.Y.C. Kim. (2024). 3D Object State Extraction Through Adjective Analysis from Informal Requirements Specs. KIPS Transactions on Software and Data Engineering, vol. 13, no. 10, pp. 529-536. https://doi.org/10.3745/TKIPS.2024.13.10.529.

[2] N. B. Ruparelia. (2010). Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, vol. 35, no. 3, pp. 8-13. https://doi.org/10.1145/1764810.1764814.

[3] J. Shin, J. Nam (2021). A survey of automatic code generation from natural language. Journal of Information Processing Systems, vol. 17, no. 3, pp. 537-555. https://doi.org/10.3745/JIPS.04.0216

[4] B. Idrisov, T. Schlippe (2024). Program Code Generation with Generative AIs. Algorithms, vol. 17, no. 2, pp. 62. https://doi.org/10.3390/a17020062.

[5] C.Y. Seo, J.H. Kim, R.Y.C. Kim. (2021). Applied Practices on Codification Through Mapping Design Thinking Mechanism with Software Development Process, KIPS Transactions on Computer and Communication Systems. Vol.10, No.4, pp.107-116. https://doi.org/10.3745/KTCCS.2021.10.4.107.

[6] N. Chomsky, Syntactic structures, USA: Mouton de Gruyter, 2002.

[7] C.J. Fillmore. The Case for Case. Universals in Linguistic Theory, ed. by Emmon Bach and Robert T. Harms, 1-90. Holt, Rinehart & Winston: New York, 1968.

[8] B.K. Park and R.Y.C Kim. (2020). Effort estimation approach through extracting use cases via informal requirement specifications. Applied Sciences vol. 10, no. 9: 3044. https://doi.org/10.3390/app10093044.