ISSN 2093-0542




KOREAN SOCIETY FOR INTERNET INFORMATION

The 20th Asia Pacific International Conference on Information Science and Technology

(APIC-IST 2025)

July 06-09, 2025, SAii Laguna Resorts, Phuket, Thailand

http://www.apicist.org

Proceedings of APIC-IST 2025

| Organized by |

Korean Society for Internet Information (KSII)

Contents

An Applied Practice on Software Quality Measurement Mechanism based on Non-Caching Iteration-Augmented Generation

Jinmo Yang[1], Chansol Park[2], and R. Young Chul Kim[3*]
[1,3*] SE Lab, Hongik University, Sejong, South Korea
[2] Wisenut Inc., Sungnam, South Korea
[e-mail: [1]yjmd2222@g.hongik.ac.kr, [2]chansol53@wisenut.co.kr, [3*]bob@hongik.ac.kr]
[*]Corresponding author: R. Young Chul Kim

Abstract

In the era of information and data technology, it is crucial to ensure that software products meet quality standards and to be considered "good" software. Nowadays, it is almost impossible to find domains where AI applications are not being researched. However, as AI-based software inherently contains uncertainties due to the nature of AI, rigorous verification and validation are critical. We propose a novel method of measuring code quality by providing information on quality measurement for comprehensive evaluation, using non-caching iteration-augmented generation (NCIAG) method. Whereas traditional retrieval-augmented generation (RAG) searches for similarity match on the user query and the information to be provided, we design the workflow to iteratively provide relevant information, without caching, thereby increasing AI's focus on the current code metric measurement. For future work, we aim to build a software visualization tool that runs our code analyzer.

Keywords: software quality, metrics, abstract syntax tree, generative AI, retrieval-augmented generation

1. Introduction

Software quality is the degree to which a software product meets the specified requirements and user expectations [1]. This includes various characteristics, such as functional suitability, reliability, usability, etc. As the current and future world revolves around AI, software of different varieties and capacities are produced with generative AI. Generative AI is normally trained with large corpus of data, which may contain biased, incorrect, or outdated information [2]. The resulting AI inherently generates outputs with less than 100 % confidence, indicating that the need for verifying and validating the software is of utmost importance.

Traditionally, measurements on software quality were collected with manual calculations and rule-based analyzers implemented by developers. Nowadays, AI measures software quality from the input, by learning the patterns within the software development process as input and understanding high-level instruction—prompts. Users do not design algorithms but provide data for the AI to capture the important features for making decisions. As software analyzers are software themselves, the advancement of these tools follows the shift of

This research was conducted with the support of the Korea Creative Content Agency (Project Name: Artificial Intelligence-Based Interactive Multimodal Interactive Storytelling 3D Scene Authoring Technology Development, Project Number: RS-2023-00227917, Contribution Rate: 100%) and the Korea Research Foundation's four, Brain Korea 21 (Project Name: Ultra-Distributed Autonomous Computing Service Technology Research Team, Project Number: 202003520005).

software paradigms [3]. In Software 1.0, humans write the software from the development process [3]. In Software 2.0, parts of the software are trained with and captured by AI, optimized for predictions [4]. In Software 3.0, requirements for the software are given as input to the AI and the source code is produced [5].

In our earlier work, we trained the CodeBERT model with the transfer learning method to detect Common Weakness Enumeration (CWE), lack of cohesion in methods (LCOM), and response for a class (RFC) of programs written in the java language or similar languages, or both [6,7]. This method is from the Software 2.0 paradigm, which requires fine-tuning of the pretrained CodeBERT model for specific code metric.

To progress further and leverage recent AI mechanisms for generalization, interoperability, and better performance, we propose the software quality measurement mechanism with non-caching iteration-augmented generation (NCIAG). Our novel NCIAG is based on retrieval-augmented generation (RAG) [8], where the retriever is replaced with iteration. From the knowledge base of embedded documents that contain the explanation and measurement algorithms of the quality metrics of our interest, NCIAG can focus on the specific quality metric in each iteration to measure the quality with high performance.

Interestingly, the analysis mechanisms that we mention are AI systems themselves. It is ironic that we use AI to verify AI, where AI possesses uncertainty. This is an important note to the paradox of the current AI analysis grounds. AI can be a source of both power and potential errors, which further promotes the development of AI techniques to improve AI usage and decrease errors.

The rest of the paper is as follows. Section 2 reviews the related works. Section 3 presents software quality measurement with our NCIAG. Section 4 shows the applied practice of the proposed mechanism. Finally, Section 5 mentions the conclusion.

2. Related Works

2.1 Rule-based Code Analysis

The traditional method for measuring software quality is for humans or algorithms designed by humans to measure the quality of the software [9, 10]. Specifically for code metrics, developers extract abstract syntax tree (AST) of the code and implement analyzers that use this parsed information for the assessment. Gorchakov *et al.* [11] proposed an analyzer system that computes educational complexity of Python code from the Python AST that can be used in programming courses for automatic grading.

Initially for this context, we built a toolchain that extracts AST of various languages and stores the parsed information into database [12,13,14]. With rule-based code metric measurement methods, our tool extracts metric scores from the lookup tables. Because the system is modularized, more programming languages can be added for analysis, given that the measurement methods are defined.

Rule-based methods can be trusted with 100 % confidence if they are completely implemented. However, implementation often requires deep understanding of the syntax and the measurement procedure. Moreover, AST can vary greatly across significantly different programming languages, requiring a substantial rewrite to the metric measurement method.

2.2 AI Models as Analysis Algorithms

AI is technology that allows computers to perform tasks that require human cognition [15]. Practically all phenomena can be represented with AI, including software quality assessment. Khan *et al.* [16] conducted a comprehensive analysis on software quality test with multiple machine learning algorithms. The authors used various machine- and deep-learning models to accurately predict software defects and determine the most significant deciding features in the source code.

Advancing from rule-based approach, we used the CodeBERT model for static code analysis [6,7,17]. CodeBERT is an encoder-only transformer model that specialized in understanding code at the time of release [18]. Setting the model to the classification and regression mode [19], we trained the model from the input code and ground truth quality metrics to give quality measurements as the output. Since the transformer foundation model can recognize similar patterns of different text strings, the downstream CodeBERT model can train from the patterns of code in one language and analyze the quality of the code written in a similar, yet

different language [7]. However, as AI performance increases exponentially over time [20], the CodeBERT model can now be considered outdated. Additionally, methods to complement the AI reasoning for higher capacity are actively explored; engineering prompts to direct the AI to think about the measurement process and to calculate by small steps greatly enhances the output quality [21].

2.3 AI Enhancement Techniques

Nowadays, techniques to enhance AI experience are announced on a daily basis. It is widely known that entering a concise prompt containing the directions for the AI to follow reduces hallucination and improves the quality of the output [21].

RAG systems can further increase the correctness of the output. These systems use a prebuilt knowledge base that provides information related to the user's query, concatenates the query and the information, and enters the resulting data into the AI for generating output that is closely bound to the context, thus reducing the likelihood that the AI generates false information from its internal weights only [8].

Better advancement is that AI systems act as agents to complete complex user requests e.g. from summarizing the meeting and launching spreadsheets to record items discussed and the costs to creating the prototype of the business item and communicating with specialized AI to generate an advertising video [22]. This is a step forward from the previous paradigm where the user must actively utilize the AI outputs to complete the task.

To achieve this, major AI development companies are announcing protocols, such as model context protocol (MCP) and Agent2Agent (A2A), for AI models and tools to have consistent interface and provide high interoperability [23,24]. Whereas it is important that AI quickly fulfills users' needs, the correctness of the AI's task completion—including the output and the process—needs assessment. Currently, there is not a unified software quality assessment method for these protocols [25,26]. Also, current software quality metrics were built on extensive theoretical and empirical grounds [27], which AI must follow to properly assess the software [28]. Therefore, we propose a software quality measurement method that uses predefined knowledge base of metric definitions and measurement processes to assess the software. Our main contribution is the improvement of software quality measurement methods from our previous traditional training of the legacy CodeBERT model to use enhancement techniques to reduce hallucinations.

3. Software Quality Measurement with Non-Caching Iteration -Augmented Generation (NCIAG)

The objective is to provide an enhanced AI method that reliably measures code quality. Code analysis is regularly conducted to measure software quality from constantly updating source

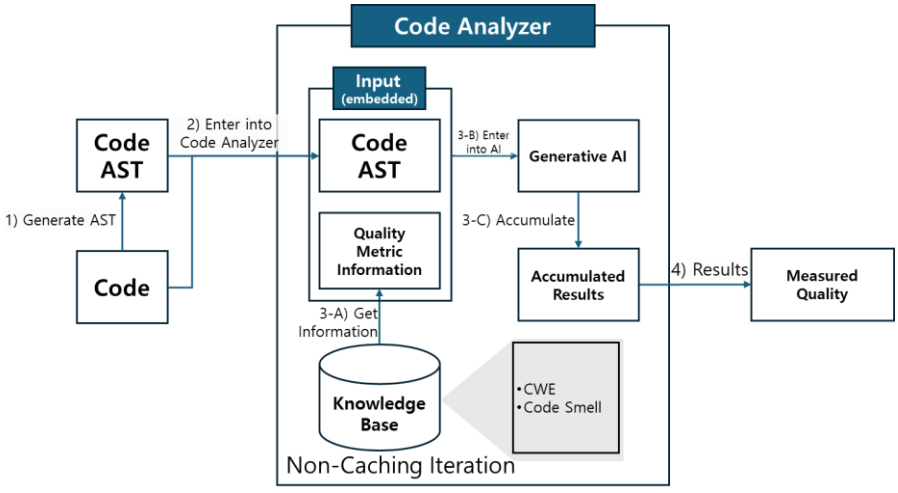

Fig. 1. Design of AI code analyzer with non-caching iteration-augmented generation

code [29]. This can be automatically executed with a continuous integration (CI) tool, which builds source code, checks code quality, runs tests, and detects defects when the source code is updated or at a fixed interval (nightly) [30]. If AI measures the software quality with acceptable performance, it is wise that we provide methods and information to conduct code analysis.

To achieve this, we propose and use NCIAG design, as shown in **Fig. 1**. NCIAG provides information for code assessment from the knowledge base with source code as input. Each document in the knowledge base contains information on how to calculate a single code quality metric. While the supply of information is the same as the traditional RAG mechanism, the similarity checks for retrieving relevant information are omitted. Instead, our novel approach uses code analysis information from the knowledge base until all documents are iterated. However, the prompt from the previous iteration is not cached in memory. This is an important design for the AI to focus on the current measurement only. As with are exhausted. on increasing the AI performance with augmentation for software quality measurement.

Below are the steps and the explanation of the code analysis process with NCIAG.

1) The AST of the source code is generated. This is to help the AI focus on the structure of the code.
2) The AST and the code is entered into the code analyzer. As it is used throughout the whole iteration, it is embedded with the same embedding model as the knowledge base.
3) Iteration of the code analyzer.
   A. The document corresponding to the current iteration index is selected and concatenated to the embedded code AST. This input block holds source code and current metric measurement information.
   B. The concatenated input is entered into the generative AI for analysis on the current code metric. The AI will read the source code AST and measure its quality as explained in the current quality metric information document.
   C. The AI results are accumulated. The accumulation style is not specified, but JSON is preferred for compatibility.
4) The accumulated results are returned for a complete quality measurement report.

The CI tool integration and dashboard connection are out of the scope of this paper. We plan on developing a code visualization tool incorporating the CI tool, code analysis, and the dashboard in our future work.

4. A Case Study for Our Proposed Mechanism

For the applied practice, we ran a single iteration of long method of the code smell [31] on a sample Python code generated with Gemini 2.5 Flash [32]. We used the Qwen3 8B model [33] for the generative AI and the BGE-M3 [34] for the embedding model. **Fig. 2** shows the code

```python
class OrderProcessor:
    def create_new_order(self, customer_id, items, currency="USD"):
        order = {}
        order['order_id'] = f"ORD-{random.randint(100000, 999999)}"
        order['customer_id'] = customer_id
        order['created_at'] = datetime.datetime.now().isoformat()
        order['items'] = items
        total_amount = 0.0
        product_prices = {
            "PROD001": 15.99, "PROD002": 25.50, "PROD003": 5.00,
            "PROD004": 120.00, "PROD005": 3.75
        }
        for item in items:
            product_id = item.get('product_id')
            quantity = item.get('quantity', 0)
            if product_id in product_prices and quantity > 0:
                total_amount += product_prices[product_id] * quantity
            else:
                print(f"""Warning: Invalid item or quantity \
                    for {product_id}. Skipping.""")
        order['total_amount'] = round(total_amount, 2)
        order['currency'] = currency
        order['status'] = "PENDING"
        order['shipping_address'] = "123 Main St, Anytown, USA"
        order['payment_method'] = "Credit Card"
        if random.random() < 0.1:
            discount = round(total_amount * random.uniform(
                0.01, 0.05), 2)
            order['total_amount'] -= discount
            order['discount_applied'] = discount
        else:
            order['discount_applied'] = 0.0
        order['estimated_delivery_days'] = random.randint(3, 10)
        order['fulfillment_steps'] = \
            ["Packed", "Shipped", "Delivered"]
        order['customer_notes'] = ""
        priority_levels = ["Low", "Medium", "High"]
        order['priority'] = random.choice(priority_levels)
        order['express_shipping'] = random.choice([True, False])
        order['reference_code'] = ''.join(random.choices(
            "ABCDEFGHJKLMNPQRSTUVWXYZ23456789", k=8))
        return order
```

Fig. 2. Python sample code (Gemini 2.5 Flash)

```json
[
    {
        "type": "code",
        "name": "long_method",
        "description": "Method 'create_new_order' has 33 lines",
        "module/class": "OrderProcessor",
        "line number": 1,
        "severity": "low"
    }
]
```

Fig. 3. Single iteration of code analysis

excerpt, and **Fig. 3** shows the code analysis result. From the result, it can be seen that the NCIAG approach for software quality measurement is valid and promising.

5. Conclusions

We have proposed the software quality measurement method based on non-caching iteration-augmented generation. This method uses RAG-based AI enhancement technique to correctly assess code quality by iterating through documents that hold metric calculation information with non-caching mechanism for the AI to focus on current calculation. This paper is an applied practice on our proposed mechanism. Therefore, we plan on developing a complete module for integration with a visualization tool and conduct a comparison study with mature rule-based and our previous training-based code analyzers.

References

[1] Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model, ISO/IEC 250100:2023, International Organization for Standardization/International Electrotechnical Commission, Geneva, Switzerland, 2023.

[2] Y. Bang, S. Cahyawijaya, N. Lee et al., "A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, pp.1-45, 2023.

[3] M. Carbin, "Overparameterization: A connection between software 1.0 and software 2.0," in *Proc. of 3rd Summit on Advances in Programming Languages (SNAPL 2019)*, pp.1-13, 2019.

[4] M. Dilhara, A. Ketkar, and D. Dig, "Understanding software-2.0: A study of machine learning library usage and evolution," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.30, no.4, pp.1-42, 2021.

[5] J. Park, "Software 3.0: A new programming paradigm opened by LLMs," *The Journal of the Korean Institute of Communication Sciences*, vol.41, no.1, pp.86-94, 2023.

[6] C. Park and R. Kim, "Detecting common weakness enumeration through training the core building blocks of similar languages base don the CodeBERT model," in *Proc. of 30th Asia-Pacific Software Engineering Conference (APSEC)*, pp.641-642, 2023.

[7] C. Park, J. Yang, J. Kong, and R. Kim, "Measuring software complexity of other similar structured softwares through learning the characteristics of a single high level language," in *Proc. of 11th International Symposium on Advanced and Applied Convergence (ISAAC)*, vol.AACL22, pp.96-100, 2023.

[8] P. Lewis, E. Perez, A. Piktus et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. of 34th Conference on Neural Information Processing Systems (NeurIPS)*, pp.1-16, 2020.

[9] J. Yahaya, Z. Deraman, and A. Hamdan, "Software quality form behavioral and human perspectives," *International Journal of Computer Science and Network Security (IJCSNS)*, vol.8, no.8, pp.53-63, 2008.

[10] T. Sharma and D. Spinellis, "Do we need improved code quality metrics," *arXiv preprint arXiv:2012.12324*, pp.1-11, 2020.

[11] A. Gorchakov, L. Demidova, and P. Sovietov, "Analysis of program representations based on abstract syntax trees and higher-order Markov chains for source code classification task," *Future Internet*, vol.15, no.9, pp.1-28, 2023.

[12] C. Park, B. Jeon, and R. Kim, "Effective code static analysis and visualization based on normalization of internal code information," in *Proc. of Annual Conference of KIPS (ACK2022)*, 2022.

[13] J. Kim, C. Park, S. Moon et al., "Blockchain code and quality visualization," *The Magazine of the IEIE*, vol.49, no.463, pp.66-74, 2022.

[14] C. Park, S. Moon, and R. Kim, "Quality visualization of quality metric indicators based on table normalization of static code building information," *KIPS Trans. Softw. And Data Eng.*, vol.12, no.5, pp.199-206, 2023.

[15] H. Abbass, "Editorial: What is artificial intelligence?," *IEEE Transactions on Artificial Intelligence*, vol.2, no.2, pp.94-95, 2021.

[16] A. Khan, R. Mekuria, R. Isaev, "Applying machine learning analysis for software quality test," *arXiv preprint arXiv:2305.09695*, pp.1-16, 2023.
[17] C. Park, S. Moon, and R. Kim, "Detecting common weakness enumeration (CWE) based on the transfer learning of CodeBERT model," *KIPS Trans. Softw. And Data Eng.*, vol.12, no.10, pp.431-436, 2023.
[18] Z. Feng, D. Guo, D. Tang, et al., "CodeBERT: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, pp.1-12, 2020.
[19] T. Wolf, L. Debut, V. Sanh, et al., "Transformers: State-of-the-art natural language processing," in *Proc. of 2020 Conference on Empirical Methods in Natural Language Processing*, pp.38-45, 2020.
[20] Vaibhavs10, 2024-ai-timeline, Hugging Face Spaces. [Online]. Available: https://huggingface.co/spaces/reach-vb/2024-ai-timeline. Accessed May 22, 2025.
[21] Y. Kim, J. Kim, L. Kim, and S. Kim, Quickpass AI: Prompt application strategies, AI Prompt Application Strategies Laboratory, 2024.
[22] D. Acharya, K. Kuppan, and B. Divya, "Agentic AI: Autonomous intelligence for complex goals – A comprehensive survey," *IEEE Access*, vol.13, pp.18912-18936, 2025.
[23] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (MCP): Landscape, security threats, and future research directions," *arXiv preprint arXiv:2503.23278*, pp.1-20, 2025.
[24] Google, Agent2Agent (A2A) Protocol, Google. [Online]. Available: https://google-a2a.github.io/A2A/. Accessed May 21, 2025.
[25] Z. Luo, X. Shi, X. Lin, and J. Gao, "Evaluation report on MCP servers," *arXiv preprint arXiv:2504.11094*, pp.1-16, 2025.
[26] Twillo Emerging Technology & Innovation Team, MCP-TE benchmark: Evaluating model context protocol task efficiency, GitHub. [Online]. Available: https://github.com/nmogil-tw/mcp-te-benchmark. Accessed May 21, 2025.
[27] N. Fenton and J. Bieman, Software Metrics: A Rigorous and Practical Approach, 3ed, CRC Press, 2014.
[28] S. Ali, V. Naganathan, D. Bork, "Establishing traceability between natural language requirements and software artifacts by combining RAG and LLMs," *Conceptual Modeling*, pp.295-314, Springer, 2024.
[29] M. Shahin, M. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol.5, pp.3909-3943, 2017.
[30] S. Bobrovskis and A. Jurenoks, "A survey of continuous integration, continuous delivery and continuous deployment," in *Proc. of 2018 BIR workshops*, pp.314-322, 2018.
[31] K. Shivashankar and A. Martini, "PyExamine A Comprehensive, UnOpinionated Smell Detection Tool for Python," *arXiv preprint arXiv:2501.18327*, pp.1-12, 2025.
[32] Google, Gemini 2.5 Flash/ [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash. Accessed Jun. 11, 2025.
[33] A. Yang, A. Li, B. Yang, et al., "Qwen3 Technical Report," *arXiv preprint arXiv:2505.09388*, pp.1-35, 2025.
[34] J. Chen, S. Xiao, P. Zhang, et al., "BGE M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," *arXiv preprint arXiv:2402.03216*, pp.1-18, 2024.