Bridging the Gap in Code Reviews: A Large Language Model-based Proposal for Global Development Teams

Gustavo Martins[1], **Emiliandro Firmino**[1]

[1]Sidia Instituto de Ciência e Tecnologia
Amazonas – AM – Brazil

Abstract. *Code reviews are crucial for enhancing software quality and adherence to design principles. Yet, they're challenged by issues like time zone and language differences in global teams. This paper introduces an automated review bot using Large Language Models, specifically GPT-4, to enforce SOLID principles in real-time reviews. It addresses communication hurdles and offers educational insights, especially for novice developers. We explore the bot's design, its ChatGPT (GPT-4) integration, and potential impacts, comparing it with existing solutions to underline its benefits and possible constraints.*

Resumo. *Revisões de código são cruciais para melhorar a qualidade do software e a aderência a princípios de design. Contudo, são desafiadas por questões como diferenças de fuso horário e idioma em equipes globais. Este artigo apresenta um bot de revisão automatizado que usa Modelos de Linguagem de Grande Escala, especificamente o GPT-4, para impor os princípios SOLID em revisões, agilizando o processo, buscando assim minimizar obstáculos de comunicação e oferece insights educacionais, especialmente para desenvolvedores novatos. Exploramos o design do bot, sua integração com o ChatGPT (GPT-4) e impactos potenciais, comparando-o com soluções existentes destacando seus benefícios e possíveis limitações.*

1. Introduction

Contemporary software development frequently relies on the collaboration of geographically dispersed teams. This particular practice poses distinct challenges attributable to factors such as disparate time zones, linguistic obstacles, and varying levels of expertise. Automation, specifically in the realm of code reviews, plays a vital role in enforcing code quality and compliance with established design principles. It presents a promising approach for addressing these difficulties. The SOLID principles are widely acknowledged as fundamental design principles in the field of Object-Oriented Programming (OOP). These resources offer a robust basis for developing code that is both easily maintainable and scalable. However, the process of guaranteeing adherence to these principles during code reviews can be a labor-intensive task that necessitates a comprehensive comprehension of these principles.

This study proposes an automated code review bot utilizing Large Language Models, like OpenAI's GPT-4, to enforce SOLID principles in real-time during code reviews, aiding global teams with quick feedback and language barriers, while also serving as an educational tool for emerging developers. The paper details the bot's design, ChatGPT (GPT-4) integration, and usage implications, contrasting it with existing solutions to highlight its potential benefits and constraints. This comprehensive analysis aims to spur

discussion on automated SOLID principle enforcement in code reviews and broader LLM applications in software development, following an in-depth exploration of the need for such innovation and a review of current literature.

2. Background

In this section, we provide an extensive overview of the fundamental concepts upon which our research is founded. We begin by analyzing the significance of code reviews in software development and their associated benefits and difficulties. Next, we examine the SOLID principles and stress their importance in object-oriented programming. Then, we examine the Language Model (LLM) technology, discussing its current applications and potential uses in software development, namely code review. Finally, we discuss the use of bots in code development and review processes, laying the groundwork for a discussion regarding the integration of LLMs with these bots to improve code review.

2.1. Code reviews

Code reviews are an essential aspect of software development teams, whether they use waterfall models or agile methodologies. They serve as a checkpoint to ensure that the code conforms to the established standards, is well-structured, readable, and error-free. In addition to enhancing code quality, code reviews promote learning and knowledge sharing, allowing developers to gain insights from their peers [Sadowski et al. 2018]. However, human-driven code reviews have disadvantages, including time-intensiveness, inconsistent feedback due to varying levels of expertise, and the possibility of bias.

2.2. SOLID Principles

The SOLID principles, introduced by Robert C. Martin [Martin 2003], are a set of five design principles intended to make software designs more comprehensible, flexible, and maintainable. When adhered to, these principles reduce code odors, improve readability, and facilitate maintenance and scalability. The fundamentals, in summary, are:

- The Single Responsibility Principle (SRP): states that a class should have only one reason to change.
- Open-Closed Principle (OCP): Entities in software should be open to extension but closed to modification.
- Liskov Substitution Principle (LSP): Subtypes must be interchangeable with their base types without compromising the program's correctness.
- Interface Segregation Principle (ISP): Clients should not be required to rely on interfaces that they do not employ.
- Dependency Inversion Principle (DIP): High-level modules should not rely on low-level modules; instead, both should rely on abstractions.

2.3. Language Model (LLM) Technology

In recent years, Language Model (LLM) technology has seen significant advancements [Wei et al. 2022]. An LLM is, at its core, a form of artificial intelligence that comprehends, generates, and evaluates text. It has been trained on a large corpus of text data and can make context-based predictions. When applied to software development, particularly code reviews, LLMs hold great promise. They can evaluate code submissions

automatically, identify deviations from norms, and provide useful feedback. Using LLMs effectively in code review processes does not come without obstacles, such as understanding programming nuances, context, and adapting to various coding styles and standards [Kojima et al. 2022].

2.4. Role of Bots in Code Development and Review

The use of automated bots in software development and code review processes has become widespread. They assist with repetitive tasks, execute tests, verify style compliance, and even identify potential performance issues [Wyrich and Bogner 2019]. While their current application offers a number of advantages, such as time savings and fewer manual errors, there is room for improvement and evolution. The incorporation of LLMs can provide bots with a deeper understanding of context and improved feedback delivery, thereby enhancing their performance in the code review process.

2.5. Benefits for Large Global Development Teams

The utility of the proposed LLM-based code review bot for large global development teams is one of its significant benefits. In multinational corporations, teams frequently span multiple time zones and consist of members from diverse cultural backgrounds who speak different languages [McDonough et al. 1999]. In such circumstances, the asynchronous nature of code review can be especially advantageous.

With the proposed bot, developers can submit pull requests at any time without coordinating with a reviewer in a different timezone, thereby reducing bottlenecks and increasing productivity. In addition, the bot's ability to provide feedback in multiple languages (a feature inherent to advanced LLMs) can help international teams overcome language barriers. By enforcing SOLID principles, it provides a shared quality standard to which all developers, regardless of location, can adhere to.

3. Related Works

The incorporation of Language Model (LLM) technology in code development and review processes has been the focus of various recent studies. In this section, we present some related works done in this regard. A summary of the related work can be found in Table 1.

Reference	Main Focus	Methodology	Findings
[Xu et al. 2022]	Evaluation of LLMs	Benchmarks	• LLM size is not the sole performance factor. • Combined training on code and natural language is beneficial.
[Wessel et al. 2020]	Code review bots	Data analysis	• Increased merged pull requests. • Possible maintenance challenges due to increased contributions.
[Balachandran 2013]	Review Bot tool	User study	• Improved code review quality. • Effective reviewer recommendation.
[Sridhara et al. 2023]	ChatGPT for SE tasks	Experiment	• Excelled in log summarization, code clone detection. • Average performance in code review generation. • Struggled with tasks requiring deeper code comprehension.

Table 1. Comparison of the main aspects of each related work

3.1. A Systematic Evaluation of Large Language Models of Code

[Xu et al. 2022] provides a comprehensive evaluation of the performance of various large language models (LLMs) when applied to code-related tasks. The authors benchmark and compare models such as Codex, GPT-J, GPT-Neo, GPT-NeoX-20B, and their own model, PolyCoder, on a variety of tasks, including code completion, code synthesis from natural language descriptions, and code perplexity in numerous programming languages.

The paper demonstrates that the size of the model is not the only significant factor for these tasks. A smaller model (Codex 300M) outperformed all other models in the HumanEval benchmark, indicating that there is significant room for improvement in open-source models using methods other than simply increasing model size. In fact, the authors suggest that the superior performance of GPT-Neo over PolyCoder in certain languages suggests that training on natural language text and code can be advantageous for code modeling.

The paper provides insights regarding the use of LLMs for a bot reviewer. LLMs such as GPT-4 may be a better option for a review bot because they can potentially be trained on both natural language and code, resulting in a more comprehensive understanding of the code and any associated documentation or comments. In addition, because these models are capable of code synthesis from natural language descriptions, they could be useful during code reviews for providing recommendations.

3.2. Effects of Adopting Code Review Bots on Pull Requests to OSS Projects

[Wessel et al. 2020] examines the effects of adopting code review bots into the pull request workflow of open source software (OSS) projects. The researchers analyze data from 1194 OSS projects hosted on GitHub, focusing on the differences between merged and non-merged pull requests prior to and after the implementation of a code review bot.

The study indicates that implementing such a code review bot may lead to increased merged pull requests, suggesting faster and better feedback. While not directly addressing LLMs or SOLID principles in code review, the study implies the potential of LLMs in enhancing bot capabilities, such as identifying SOLID principle violations, possibly influencing the uptick in merges.

3.3. Reducing Human Effort and Improving Quality in Peer Code Reviews using Automatic Static Analysis and Reviewer Recommendation

[Balachandran 2013] proposes a tool called Review Bot to assist in reducing human effort in peer code reviews. Automatic static analysis is incorporated into the code review procedure by the tool. It uses output from multiple static analysis tools to automatically publish reviews. Through a user study, this paper demonstrates that integrating static analysis tools into the code review process can enhance the quality of code reviews. The Review Bot also includes an algorithm for recommending reviewers, which aims to simplify the process of identifying suitable reviewers for a large project.

The application of Large Language Models (LLMs) in code review is consistent with the standard procedure of utilizing automated tools for this purpose. Similar to how the Review Bot identifies code issues through static analysis, an LLM-based bot could be developed to detect violations of the SOLID principles. Furthermore, the integration

of the Review Bot's approach to recommending reviewers on the basis of previous line changes into an LLM-based system could facilitate the identification of reviewers who possess expertise or prior interactions with the particular code being evaluated.

3.4. ChatGPT: A Study of its Utility for Common Software Engineering Tasks

[Sridhara et al. 2023] examines the potential applications of ChatGPT (Chat Generative Pre-trained Transformer) for common software engineering tasks. The authors conducted an experiment with fifteen pervasive software development tasks and evaluated ChatGPT's utility for these tasks. They computed the accuracy of ChatGPT for each task by comparing its output to that of human experts and/or cutting-edge tools.

The paper demonstrates that ChatGPT excels at log summarization, anaphora resolution, code summarization (method name generation), and code clone detection. However, its performance for tasks such as commit message generation, code review generation, natural language code search, and merge conflict resolution was only average. The paper further suggests the model's performance on code review tasks was only average. This may suggest that, although LLMs can provide some assistance with code review, their effectiveness may be limited. The SOLID principles are not explicitly mentioned in the paper, but given ChatGPT's performance on the tasks evaluated, it may be reasonable to anticipate similar difficulties when applying LLMs to SOLID principle checks.

3.5. Insights and Implications for LLM-Based Code Review

The review of related works reveals a number of insightful observations regarding the application of automated tools and LLMs in code review. [Xu et al. 2022] demonstrate that although LLMs perform well in a variety of tasks, their efficacy is not universally superior, emphasizing the need for task-specific evaluations. Our proposal to investigate the use of an LLM-based bot for enforcing SOLID principles relies heavily on this finding.

[Wessel et al. 2020] and [Balachandran 2013] provide empirical evidence regarding the influence of automated review tools on code review workflows, demonstrating a generally positive effect. These findings encourage the investigation of LLMs' potential to further improve the quality of code review. Nevertheless, [Sridhara et al. 2023] indicates that our proposal may face obstacles regarding the efficacy of LLMs for basic coding tasks.

4. Proposed Concept

The core concept of the proposed bot is to act as a guide during the PR process, assisting developers by providing real-time feedback on the SOLID compliance of their code. The bot primarily operates on Pull Requests (PRs). When a pull request is submitted, a webhook triggers the bot to analyze the code changes.

The bot is designed to leverage the capabilities of LLM technology, with ChatGPT (GPT-4) serving as the primary API for code analysis. The GPT-4 model was trained on a wide variety of internet text, but also refined with a focus on code, thefero, it can provide the foundation for identifying common violations of the SOLID principles and recommending improvements. The bot parse the code, separating it into understandable segments, and then feed each segment to the ChatGPT API for evaluation. The results, consisting of potential violations and recommendations, are aggregated, processed, and

added to the PR as comments, making the bot's feedback easily accessible and immediately actionable for developers. The objective here is not to replace human reviewers, but rather to supplement them by automating the initial stages of the review process, ensuring a focus on design principles, and allowing human reviewers to concentrate more on the logic and other crucial aspects of the code.

4.1. Proposed Architecture and Integration

The bot is based on a modular architecture with three primary components, each with a distinct function.

- Integration of GitHub: This element connects the bot to the GitHub environment. It serves two purposes. On the one hand, it monitors GitHub for events (such as the submission of a new PR), initiates the code analysis process, and notifies the Code Analysis Layer of code changes. In contrast, it posts the processed results from the Code Analysis Layer as comments on the PR. It is designed to communicate seamlessly with the GitHub API, allowing the bot to operate effectively within the GitHub ecosystem.
- ChatGPT API: is an indispensable component of the bot. It provides the bot with ChatGPT's natural language processing capabilities. It receives the parsed code segments from the Code Analysis Layer, analyzes them, and returns natural language descriptions of the code's semantics as well as any potential SOLID violation. The information returned by the ChatGPT API is used as the foundation for the bot's comments on PRs.
- Code Analysis: This is where the bulk of the bot's analysis occurs. It takes the code changes from the GitHub Integration Layer, parses them into segments that can be fed into the ChatGPT API, and manages the ChatGPT API requests. After obtaining the results from the ChatGPT API, they are processed and analyzed to identify potential violations of the SOLID principle. The results are then arranged in an understandable format and sent back to the GitHub Integration Layer in order to be published on GitHub.

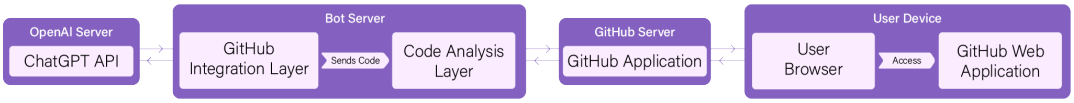

Figure 1. Component Diagram of the Proposed GitHub Bot for Code Review Automation.

The Figure 1 depicts the components of the proposed system for automating code reviews. When a new pull request (PR) is created or an existing PR is modified on GitHub, a cyclical process is initiated. The Code Analysis Layer receives the extracted changes from the GitHub Integration Layer, which is responsible for interacting with the GitHub platform. This layer converts code changes into a format that can be analyzed by the ChatGPT API. Once the analysis is complete, the API returns results that include potential violations of the SOLID principle and suggestions for improvement. The Code Analysis Layer transforms these results into a developer-friendly, human-readable format. The results are then sent back to the GitHub Integration Layer, which adds the comments to the corresponding PR. Any changes to the PR will cause the bot to re-evaluate the code and update the comments as required. This flow is shown in Figure 2.

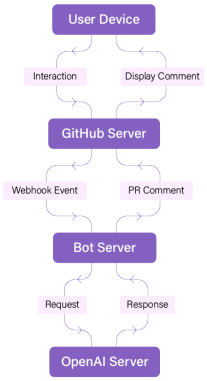

Figure 2. Simplified Flow Diagram of the Proposed GitHub Bot, a Continuous Analysis Cycle.

4.2. Usage of the Proposed Bot

The bot is intended to be a seamless addition to the developer's workflow. Once integrated into a GitHub repository, the bot is automatically activated whenever a pull request (PR) is created. There are no additional steps required for developers to initiate the bot's review.

Once the automated system has concluded its analysis of the PR, it proceeds to publish its comments directly on the PR. The following observations highlight possible deviations from the SOLID principles in the modified code and propose recommendations for improvements. The feedback provided can be utilized by the developers to enhance the code's quality prior to a human review of the pull request. This practice not only guarantees that the code follows essential design principles, but also functions as an educational resource to assist developers in gaining a deeper understanding of the SOLID principles and their practical implementation in code.

5. Impact Analysis

Our proposed code review bot is fundamentally distinct from existing solutions in its approach to code review and emphasis on SOLID principles enforcement as we analyse in this section.

5.1. Comparison with Existing Solutions

Unlike the work done by Xu et al. [Xu et al. 2022], our proposed bot takes advantage of LLMs' ability to comprehend both natural language and code. In addition, the evaluation of adherence to SOLID principles necessitates a deeper comprehension of the code's structure and design, which may prove difficult for even the most advanced LLMs. However, this is a niche that requires additional research, and our work aims to contribute to this field.

The paper by Wessel et al. [Wessel et al. 2020] highlights the positive effect of code review bots on the efficiency of OSS projects. In addition to increasing productivity, our bot aims to improve code quality by highlighting places where SOLID principles are violated. It is hoped that by addressing these violations, we can avoid a portion of the maintenance burden that code review bots may introduce.

Meanwhile, Balachandran [Balachandran 2013] illustrates the advantages of automated code review with static analysis. Our bot shares the goal of reducing human effort, but aims to supplement static analysis with LLMs' comprehension abilities. This dual approach could result in recommendations that are more precise and sensitive to context.

And finally, we try to minimize the issues identified by Sridhara et al. [Sridhara et al. 2023] by concentrating on a specific area we hope to enhance its performance in this task, building upon the average performance of ChatGPT in code reviews.

5.2. Potential Benefits

This proposal suggests a GitHub bot that uses Large Language Models, specifically GPT-4, to improve the code review process significantly. The bot automates many aspects of code reviews, providing quick feedback and freeing up human reviewers to focus on more complex code. It applies SOLID principles, which are fundamental in object-oriented programming, to prevent design issues and improve code durability and strength.

Table 2. Key benefits of the proposed bot

Benefit	Description
Enhanced Code Quality	By enforcing SOLID principles, we aim to improve the overall quality of code in the projects.
Time Efficiency	The bot can assist in identifying SOLID principles violations, potentially saving review time.
Educational	The bot could serve as an educational tool for developers unfamiliar with SOLID principles.

Furthermore, the bot is an effective learning tool for developers, particularly beginners, providing immediate and consistent feedback on SOLID principles, thereby accelerating learning and maintaining high coding standards. It ensures consistent review criteria, which contributes to a fair development environment, and it can handle multiple reviews at the same time, providing thorough feedback regardless of the number of submissions. This feature makes it ideal for large or geographically dispersed development teams. Table 2 summarizes the main benefits of such review bot.

5.3. Challenges and Limitations

Despite its potential advantages, the implementation of our proposed bot presents a number of obstacles and constraints. Achieving a comprehensive understanding of code context and higher-level design principles such as SOLID is one of the greatest obstacles. Due to the complexity and nuance of these principles, current LLMs may not always be able to fully grasp them, as they frequently require a human-like understanding of code. This incomplete understanding can result in false positives or negatives. The bot may identify nonexistent issues (false positives) or miss genuine issues (false negatives), which could impact the bot's dependability and efficiency, resulting in unnecessary work or overlooked issues.

LLMs, such as ChatGPT in its GPT-4 implementation, have demonstrated impressive capabilities, but they have inherent limitations. These models generate outputs based on training-learned patterns and lack the capacity to reason about the world like humans. This could hinder their ability to detect violations of the SOLID principles, which frequently require a nuanced understanding of the code.

6. Conclusion

In this paper, we presented a proposal for a GitHub bot that enforces SOLID principles during the code review process by leveraging the capabilities of Large Language Models (LLMs). By focusing on a specific aspect of code quality, our bot could potentially enhance the code review process.

It is evident that existing code review bots have had a positive impact on the software development workflow, thereby improving the productivity and quality of open-source software projects. However, current solutions may overlook certain best practices such as the enforcement of SOLID principles and may not utilize the capabilities of LLMs fully. The proposed bot aims to address these deficiencies by leveraging the capabilities of LLMs, specifically GPT-4, to evaluate code against SOLID principles and provide meaningful and real-time feedback. Through this method, we expect benefits including enhanced code quality, enhanced time efficiency, and educational value for developers.

While our concept is currently a theoretical proposal, we anticipate its practical implementation and evaluation, as we believe it could significantly contribute to the automation of code review processes and enhancement of software quality. We hope that this work will serve as a foundation for future research and development in this area, especially concerning the use of LLMs to enforce specific code quality principles.

References

Balachandran, V. (2013). Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 931–940. IEEE.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

McDonough, E. F., Kahn, K. B., and Griffin, A. (1999). Managing communication in global product development teams. *IEEE Transactions on Engineering Management*, 46(4):375–386.

Sadowski, C., Söderberg, E., Church, L., Sipko, M., and Bacchelli, A. (2018). Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*, pages 181–190.

Sridhara, G., Mazumdar, S., et al. (2023). Chatgpt: A study on its utility for ubiquitous software engineering tasks. *arXiv preprint arXiv:2305.16837*.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Wessel, M., Serebrenik, A., Wiese, I., Steinmacher, I., and Gerosa, M. A. (2020). Effects of adopting code review bots on pull requests to oss projects. In *2020 IEEE international conference on software maintenance and evolution (ICSME)*, pages 1–11. IEEE.

Wyrich, M. and Bogner, J. (2019). Towards an autonomous bot for automatic source code refactoring. In *2019 IEEE/ACM 1st international workshop on bots in software engineering (BotSE)*, pages 24–28. IEEE.

Xu, F. F., Alon, U., Neubig, G., and Hellendoorn, V. J. (2022). A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10.