Conceptual Model of a Hybrid Automated AI-Based Code Review System

Tea Todua

Candidate of Technical Sciences (equivalent to a PhD degree), Associate Professor, Georgian Technical University (GTU), Tbilisi, Georgia.

***Email**: tea_todua@gtu.ge*

Giorgi Tsitlidze

PhD student, Georgian Technical University (GTU), Tbilisi, Georgia.

***Email:** tsitlidze.giorgi25@gtu.ge*

Abstract

Ensuring code quality has become one of the important challenges in modern software development processes. As software systems grow, code complexity, architectural complexity, security risks, and the likelihood of technical debt accumulation also increase. Traditional code review processes, which mainly rely on human expertise and manual analysis, do not always ensure fast and consistent evaluation.

This paper presents the concept of a Hybrid Automated AI-Based Code Review System, which combines static analysis, rule-based validation, project context processing, and the capabilities of Large Language Models (LLMs). The presented approach considers code review not merely as a process of generating comments, but as a context-aware multi-stage process that includes code change assessment, risk identification, problem localization, and generation of explainable recommendations.

The research methodology is based on the analysis of existing AI-based code review approaches and the development of a hybrid architectural model. The proposed model utilizes multi-source context, including code changes, project structure, code dependencies, static analysis results, testing information, and Pull Request descriptions.

The main contribution of the research lies in the development of a conceptual model of a hybrid AI-based code review system that integrates traditional software analysis methods with AI-based contextual reasoning. This approach aims to improve the accuracy, explainability, and practical applicability of code review systems while supporting, rather than replacing, human reviewers.

Keywords: Hybrid AI, Automated Code Review, Context-Aware Analysis, Large Language Models, Static Code Analysis, Software Quality

Introduction

In the software quality control process, code review has long been considered one of the fundamental practices. Its purpose is not only to identify software defects but also to evaluate code readability, maintainability, security, and architectural compliance. Traditional code review is mainly performed by developers who analyze code changes, discuss potential issues, and provide recommendations (Fagan, 1976).

Traditional code review primarily relies on developers' experience and manual analysis. This approach plays an important role in software quality assurance; however, it has several limitations. In particular, the review process often depends on the knowledge and experience of individual reviewers (Bosu & Carver, 2013), may become time-consuming when dealing with large-scale changes, and carries the risk of missing recurring defects, security issues, or architectural inconsistencies.

In modern Agile and DevOps practices, software changes are implemented at a high frequency, which increases the need for automated and intelligent code review systems. In particular, it is important to develop mechanisms that are capable of not only detecting syntax and style-related issues but also performing deeper analysis of code changes by evaluating their purpose, impact, and potential risks.

In recent years, the development of artificial intelligence, particularly Large Language Models (LLMs), has created new opportunities in the field of software engineering. AI-based systems are capable of analyzing code fragments, processing natural language descriptions, identifying potential issues, and providing recommendations to developers. In the code review process, such systems can be applied to Pull Request analysis, where code changes are examined and their potential impact on the software system is assessed before integration.

Despite the progress achieved in this area, AI-based code review systems still face significant challenges. One of the major challenges is the ability of the model to correctly understand the full context of a software project. Analyzing only the modified code fragments is often insufficient for assessing the actual impact of a change on the system. Other important challenges include reducing false-positive and false-negative recommendations, adapting to project-specific standards, accurately evaluating critical security issues, and enabling effective collaboration between AI-based code review systems and human reviewers.

Therefore, the development of a conceptual model of a Hybrid Automated AI-Based Code Review System is proposed to integrate traditional software analysis methods with modern artificial intelligence capabilities. Such an approach involves the integration of static analysis, rule-based checks, project context processing, and Large Language Model (LLM)-based reasoning capabilities.

The aim of the research is to develop a conceptual model and functional algorithms for a Hybrid Automated AI-Based Code Review System that enables automated analysis of software code changes, detection of potential defects, security risks, and code quality issues, as well as the generation of explainable and technically justified recommendations for developers.

The development of effective Hybrid Automated AI-Based Code Review Systems raises several important research questions related to context understanding, recommendation reliability, integration of multiple analysis techniques, and human–AI collaboration:

1. How can an AI-based code review system understand the full context of a Pull Request rather than only the modified code fragments?
2. How can false positive and false negative recommendations generated by AI-based code review systems be reduced?
3. How can static analysis results and Large Language Model (LLM)-based reasoning capabilities be effectively integrated into a unified code review process?
4. How can the quality, accuracy, and reliability of AI-generated code review comments be evaluated?
5. How can human reviewer feedback be incorporated to improve the reliability and effectiveness of AI-based code review systems?

Existing Approaches and Challenges in AI-Based Code Review systems

As software systems have grown in size and complexity, various automated methods for software quality analysis have been developed. One of the important directions in this area is static analysis, which enables software code to be examined without executing it. Static analysis tools use predefined rules, program structure analysis, and various algorithms to detect issues such as syntax errors, code style violations, known security vulnerabilities, and undesirable programming practices (Bessey et al., 2010).

A significant advantage of static analysis tools is their high accuracy in detecting rule-based issues, consistency of results, and fast execution. They are particularly effective in cases where problems can be identified based on formal rules. However, one of the main limitations of static analysis is the difficulty of understanding the broader context of a software system. In particular, such tools are often unable to determine whether a specific change is consistent with the system's business logic, architectural decisions, or what impact the change may have on related components (Johnson et al., 2013).

In addition to static analysis, rule-based analysis is another widely used approach in software quality control. Unlike static analysis, which primarily focuses on automatically detecting predefined code-level issues, rule-based analysis is based on project- or organization-specific rules, coding standards, and development policies. For example, it can be used to verify compliance with naming conventions, restrict the use of prohibited libraries, ensure adherence to security requirements, or validate architectural constraints. Such approaches are particularly important in large software teams, where consistent enforcement of code quality standards is required. However, their capabilities depend on the completeness of predefined rules, and they cannot identify problems that are not explicitly covered by existing rules or that require deeper semantic analysis.

The development of Large Language Models (LLMs) has significantly influenced the automation of software engineering tasks. LLM-based approaches have demonstrated the ability to process both natural language and source code, as well as related technical information (Brown et al., 2020). In the field of software engineering, these capabilities are used to support code analysis, code generation, and code review processes. In the context of code review, LLMs can not only identify potential issues but also explain their causes, suggest possible improvements, and generate understandable recommendations for developers (Chen et al., 2021).

The use of LLM-based capabilities has contributed to the development of modern AI-based code review tools. Research studies have presented various approaches that utilize structural and semantic information of source code to improve automated code review (Yin et al., 2023). In practice, this direction has been reflected in tools such as GitHub Copilot Code Review, which leverages Large Language Models to analyze Pull Requests and generate recommendations on code changes. Amazon CodeGuru Reviewer combines machine learning techniques with software analysis methods to identify potential code quality and security issues. Sourcery focuses on providing code improvement recommendations and supporting developers, while Qodo (formerly CodiumAI) uses AI technologies to enhance code analysis and testing within the software development process.

The aforementioned systems represent significant progress in automated code review and demonstrate the potential of AI in software quality control. Nevertheless, achieving comprehensive AI-based code review remains a challenging task. Existing approaches differ in terms of the methods they employ and the scope of context they consider. Some systems primarily focus on analyzing modified code fragments or Pull Request content, whereas more comprehensive evaluation often requires additional information, such as software architecture, code dependencies, development history, testing results, and organizational rules.

Furthermore, when using Large Language Models, the accuracy, technical justification, consistency, and reliability of generated recommendations remain significant challenges. Although LLMs demonstrate strong capabilities in semantic code understanding, they may generate recommendations that are linguistically convincing but insufficiently justified for a specific software environment. This issue is particularly important when evaluating critical security problems and architectural decisions.

These challenges indicate that effective AI-based code review systems should not rely on a single analysis approach. A hybrid methodology that combines deterministic software analysis methods, such as static analysis and rule-based checks, project context processing, and context-aware reasoning capabilities of Large Language Models represents a promising research direction. Such integration can improve the accuracy of issue detection, the explainability of recommendations, and the practical applicability of AI systems in software development processes. This perspective is also consistent with recent studies on AI-assisted programming, which highlight the importance of contextual information and effective human–AI collaboration for improving the practical use of AI in software development (Tabatadze, 2026).

Research Methodology

This research adopts a conceptual system design approach to develop the architectural model and define the functional principles of the Hybrid Automated AI-Based Code Review System.

The initial stage of the research involved an analysis of existing AI-based code review approaches, automated analysis tools, and related research in AI-assisted software engineering. Based on this analysis, the main challenges of modern systems were identified:

- Insufficient consideration of the complete context of a software system;
- The problem of generating false positive and false negative recommendations;
- Difficulty in accurately assessing security issues;
- The challenge of adapting to project-specific standards and development rules;
- The need for effective collaboration between human reviewers and AI systems.

The main idea of the presented methodology is that the AI-based code reviewer should not operate in isolation based solely on code fragments. Instead, the system should utilize multi-source context, including:

- the list and structure of modified files;
- descriptions of code changes;
- software architecture information;
- code dependencies;
- static analysis results;
- testing results;
- Pull Request descriptions;
- historical information related to software development.

The use of multi-source context enables the AI model to assess not only a specific code fragment but also the context and impact of the change within the overall software system.

Hybrid Automated AI-Based Code Review System Architecture

The functional architecture of the proposed Hybrid Automated AI-Based Code Review System consists of the following main modules:

1. Code Change Analysis Module

This module is responsible for analyzing the changes introduced in a Pull Request. It identifies which files have been modified, what types of changes have been made, and which components may be affected by these changes.

2. Static Analysis and Linter Integration Module

This module integrates the results of static analysis tools and linters. Its purpose is to provide the AI model with issues detected through formal rules for further interpretation.

3. Project Context Extraction and Structuring Module

This module collects and organizes structural information about the software project. It processes file relationships, dependencies, and architectural characteristics.

4. AI Model Prompt Construction Module

The purpose of this module is to construct an appropriate prompt for the AI model. The prompt should contain not only the code fragment but also the contextual information required for accurate analysis.

5. Review Comment Generation Module

This module generates specific, explainable, and practically useful comments for developers based on the analysis performed by the AI model.

6. Risk Assessment and Prioritization Module

This component evaluates the severity of identified issues and determines their priority, allowing human reviewers to focus first on the most critical problems.

7. Human Reviewer Feedback Processing Module

This module enables the use of human reviewer feedback for further improvement of the system. The interaction between these modules and the overall information flow within the system are illustrated in Figure 1.

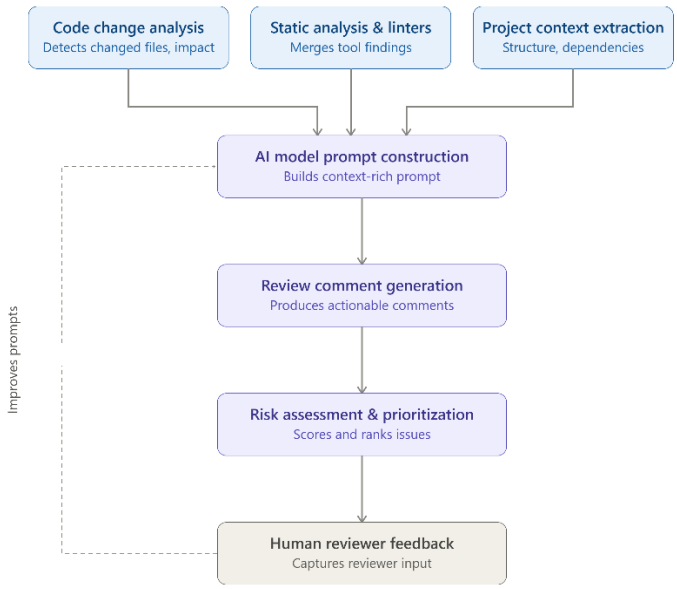

Figure 1. Conceptual architecture and information flow of the proposed Hybrid Automated AI-Based Code Review System

The core idea behind the architecture of the Hybrid Automated AI-Based Code Review System is that the code review process should not be considered merely as an automated inspection of modified code fragments or the generation of comments. In modern software projects, evaluating code changes requires considering a broader context, as a specific change may affect other system components, the overall architecture, security aspects, and the maintainability of the software product.

According to this approach, the Hybrid Automated AI-Based Code Review System represents a hybrid intelligent system that integrates various types of analysis, knowledge representation, and decision-support mechanisms:

- rule-based checking;
- static analysis;
- project context processing;
- LLM capabilities;
- system-generated recommendations and human reviewer feedback loop.

The integration of these components enables the proposed system not only to detect problems but also to analyze their causes, assess risks, and provide developers with well-justified recommendations.

In modern software engineering practice, code changes are often implemented through Pull Requests (Gousios et al., 2014). This process enables other developers to review the change, assess its quality, and make a decision regarding its integration. In the proposed system, a Pull Request represents one of the primary sources of change-related context, containing not only information about the modified code but also additional metadata: which files have been changed, which modules are affected by the change, what type of modification has been introduced, what dependencies exist between the modified components and other parts of the system, and what the purpose of the change is according to the Pull Request description. For example, the same code modification may be evaluated differently depending on where it is applied. A change implemented in a data processing module may have a different level of risk compared to a change made in a user interface component. Therefore, analyzing only a code fragment is not sufficient. The system must understand the location and significance of the change within the overall software architecture.

One of the important components of the Hybrid Automated AI-Based Code Review System is the integration of static analysis and rule-based checking mechanisms. Static analysis

tools enable the automatic detection of issues related to code quality, complexity, potential errors, and security vulnerabilities, while rule-based checks ensure compliance with predefined programming standards, architectural rules, and project-specific requirements. It should be noted that the results of static analysis and rule-based checks often represent only technical indicators and require additional interpretation. For example, static analysis may indicate that a particular function is overly complex or that a potential security issue exists, while rule-based checks may identify non-compliance with coding standards. However, determining the severity and significance of an identified issue within the context of a specific project requires additional knowledge. At this stage, the use of an LLM enables the results of static analysis and rule-based checks to be combined with project context and transformed into more understandable and actionable recommendations.

Project Context Extraction and Multi-Source Information Processing is another important component of the Hybrid Automated AI-Based Code Review System. Multi-source context implies that the proposed system does not rely on a single source of information, such as the modified code, when making decisions. Instead, it analyzes multiple types of information, including code changes, the project's file structure, dependencies between modules, architectural decisions, the results of static analysis, test results, and the history of previous changes. For example, if new code uses a particular library, it may be difficult to determine, based solely on the code itself, whether this is an appropriate design decision. However, by considering the project's architecture and the dependencies between its components, the system can determine whether the proposed change is consistent with the existing software environment.

The role of the Large Language Model in the developed system is not limited to generating textual comments. It is used for interpreting the obtained information, integrating data collected from various sources, analyzing the causes of problems, and suggesting improvement approaches.

It is important that the recommendations generated by the system are explainable recommendations. For a developer, it is not sufficient to know only that a particular part of the

code is problematic. It is important to understand why this change represents a problem, what impact it may have on the system, and why a specific improvement is recommended. Such an approach increases human trust in AI systems and facilitates effective collaboration (Gunning & Aha, 2019).

Human-AI Collaboration in the Code Review Process is a crucial aspect of the proposed system (Amershi et al., 2019). It should be emphasized that the proposed system is not intended to replace human reviewers. Its goal is to augment human capabilities and reduce routine work.

The AI-Based Code Review System can quickly inspect large volumes of changes, detect potential issues at an early stage, prepare initial recommendations, and identify high-risk changes. However, the final decision remains with the human reviewer, who can consider factors that may be unavailable to the system, such as business context or organizational priorities.

Results and Discussion

Within the framework of the study, a conceptual model of the Hybrid Automated AI-Based Code Review System has been developed. The aim of this model is to improve the code review process through the integrated use of artificial intelligence, static analysis, and project context. The main distinguishing aspect of this approach is that code review is considered not merely as a task of generating comments on code changes, but as a multi-stage analysis process that includes understanding the context of changes, identifying potential issues, assessing risks, justifying recommendations, and supporting human reviewers' decision-making.

The presented model is based on the principle that the quality assessment of software changes should not be limited to the analysis of modified code fragments only. Due to the increasing complexity of modern software systems, determining the potential impact of a change requires consideration of a broader software context, including system architecture,

dependencies between components, testing results, and other information related to the development process.

The Hybrid Automated AI-Based Code Review System implements this approach through the use of multi-source context. The integration of static analysis and rule-based approaches enables the detection of known types of issues in the code and violations of predefined rules, while the Large Language Model provides deeper interpretation of the obtained information and generates explainable recommendations.

The main contribution of this research lies in the development of a conceptual architectural model of the Hybrid Automated AI-Based Code Review System, which considers the code review process as a decision-support process based on multi-source context processing. The proposed model integrates static analysis results, rule-based checks, project context information, and Large Language Model capabilities to generate explainable and risk-oriented recommendations. Its main aspects are as follows:

1. Context-Aware Code Review Approach

Unlike traditional automated analysis, this approach considers not only the changed code but also the broader project context. This enables the evaluation of not only "what has changed," but also the purpose of the change, its potential impact, and whether it complies with the architectural requirements of the system.

2. Hybrid Integration of Multiple Analysis Techniques

The model combines methods of different types:

- rule-based checks;
- static analysis;
- context analysis;
- LLM-based intelligent analysis.

This integration reduces the limitations arising from the use of a single analysis approach.

3. Explainable AI-Based Code Review

The presented model focuses not only on detecting problems but also on explaining their causes and suggesting improvement strategies. Explainable recommendations are important because developers, when making decisions, require not only automated notifications but also their technical justification.

4. Human-Centered AI Collaboration

An important aspect of the research is preserving the role of the human reviewer. The AI system is considered a supportive tool that enhances human effectiveness and does not replace human expertise.

Expected Outcomes and Practical Implications

The implementation of the presented concept is expected to provide the following outcomes:

1. Development of a conceptual model for the Hybrid Automated AI-Based Code Review System, defining the main components of an AI-based Code Review system and their interactions.
2. Development of an algorithm for Pull Request context processing, enabling the AI model to receive the information required for accurate code review.
3. Development of a method for code change risk assessment, helping developers and reviewers focus on high-priority issues.
4. Definition of a mechanism for integrating static analysis results with AI-based recommendations, combining formal analysis with semantic reasoning.
5. Development of an approach for generating explainable recommendations in the code review process, improving the practical usability of the generated recommendations.
6. Definition of a mechanism for incorporating human reviewer feedback, enabling the system to better adapt to the requirements of a specific software project in future applications.

From a practical perspective, the presented model can serve as a foundation for developing a prototype of an AI-Based Code Review System that can be integrated into modern software development processes, including GitHub Pull Requests and CI/CD pipelines.

Such a system can be particularly beneficial for software teams where code changes are introduced frequently, the time resources of experienced developers are limited, consistent enforcement of code quality standards is required, and early detection of security issues is important.

Despite the potential of the presented approach, the study has several limitations. First, this paper describes a conceptual model that requires further practical implementation. Future research should focus on the development of a prototype and its evaluation on real-world software projects. In addition, a comparative analysis of different Large Language Models is important, as their capabilities vary in terms of code understanding, reasoning, and the quality of recommendation generation. Potential directions for future research include: implementation of a prototype of an AI-Based Code Review System; integration with GitHub and CI/CD environments; definition of evaluation metrics; experimental assessment of the quality of AI-generated comments; and development of an adaptive model based on human reviewer feedback.

Conclusion

This paper presents the concept of a Hybrid Automated AI-Based Code Review System, which aims to improve the software code review process through the integrated use of artificial intelligence, static analysis, and project context processing.

The presented concept combines the advantages of different approaches, including static analysis accuracy, structured rule-based checks, project context processing, and the ability of Large Language Models to process and analyze semantic and contextual information from source code.

The contribution of this research lies in the development of a conceptual architectural model of the Hybrid Automated AI-Based Code Review System, which integrates deterministic analysis methods (static analysis, rule-based checks) with LLM-based contextual reasoning within a decision-support process. The presented approach considers code review as a context-aware intelligent process, where the AI system is not limited to merely detecting issues but also provides analysis of their causes, risk assessment, and generation of explainable recommendations.

It is important to emphasize that the presented approach does not consider AI as a replacement for human reviewers. Instead, the system is viewed as an intelligent support tool for human reviewers, reducing routine tasks, improving review efficiency, and contributing to enhanced software quality.

REFERENCES

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). Guidelines for human-AI interaction. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3290605.3300233

Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., & Engler, D. (2010). A few billion lines of code later: Using static analysis to find bugs in the real world. *Communications of the ACM, 53*(2), 66–74. https://doi.org/10.1145/1646353.1646374

Bosu, A., & Carver, J. C. (2013). Impact of developer reputation on code review outcomes: An empirical investigation. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 29–38. https://doi.org/10.1109/ESEM.2013.15

Brown, T. B., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Chen, M., et al. (2021). Evaluating large language models trained on code. *arXiv*. https://arxiv.org/abs/2107.03374

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal, 15*(3), 182–211. https://doi.org/10.1147/sj.153.0182

Gousios, G., Pinzger, M., & van Deursen, A. (2014). An exploratory study of the pull-based software development model. https://doi.org/10.1145/2568225.2568260

Gunning, D., & Aha, D. W. (2019). DARPA's explainable artificial intelligence (XAI) program. *AI Magazine, 40*(2), 44–58. https://doi.org/10.1609/aimag.v40i2.2850

Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? *Proceedings of the 35th International Conference on Software Engineering*, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

Tabatadze, B. (2026). AI-assisted programming in practice: Conceptual foundations and data-informed observations. *Computational and Applied Science, 1*(1), 84–100. https://casjournal.ge/index.php/cas/article/view/4

Yin, Y., Zhao, Y., Sun, Y., & Chen, C. (2023). Automatic code review by learning the structure information of code graph. *Sensors, 23*(5), 2551. https://doi.org/10.3390/s23052551