GEORGE BROWN PRESS
JOURNALS, BOOKS & PROCEEDINGS

GBP Proceedings Series

Article

2024 International Conference on Art and Design, Education, Media and Social Sciences (DEMSS 2024)

Design and Research on Evaluation System of Computer Programming Code Quality

Yuchen Zhao [1,*]

[1] University of Southampton Southampton, SO17 3BD, United Kingdom
* Correspondence: Yuchen Zhao, University of Southampton Southampton, SO17 3BD, United Kingdom

Abstract: With the ongoing development of software engineering and programming education, the demand for evaluating programming code quality is increasing. High-quality code not only enhances software maintainability and performance but also helps developers and learners optimize their coding skills. This paper focuses on the design and implementation of an evaluation system for computer programming code quality, proposing a multi-dimensional evaluation method that includes readability, complexity, efficiency, and security. The system combines static analysis with machine learning to automate code analysis and optimization recommendations, providing users with objective feedback on code quality. Additionally, this paper explores application cases of the system in programming education and software development, demonstrating its effectiveness in improving code quality and enhancing users' programming skills. Finally, the research contributions are summarized, and directions for future improvements are proposed to further enhance the system's intelligence and applicability.

Keywords: code quality; code evaluation system; static analysis; machine learning; programming education

1. Introduction

In modern software development and programming education, code quality significantly impacts software reliability, maintainability, and user experience. High-quality code not only reduces maintenance difficulty and costs but also substantially improves software performance and security. Therefore, scientifically and objectively evaluating code quality has become a topic of broad interest. Code evaluation encompasses multiple dimensions, including readability, complexity, runtime efficiency, and security, which together determine the code's effectiveness in practical applications. However, traditional evaluation methods largely rely on manual code reviews and static analysis tools, which are often inefficient and insufficiently comprehensive, particularly in large and complex codebases where manual review can be subjective and time-consuming.To address these issues, this paper proposes a computer programming code quality evaluation system that combines static analysis with machine learning. The system provides a multi-dimensional evaluation that applies to both educational contexts, as feedback on code quality for students, and software development environments, aiding in code review and optimization efforts. By analyzing factors such as readability, logical complexity, execution efficiency, and potential security risks, the system provides developers with detailed feedback and improvement suggestions, helping them elevate their coding standards and overall code

Received: 08 December 2024
Revised: 21 December 2024
Accepted: 07 January 2024
Published: 09 January 2025

quality. Moreover, this research examines specific applications of the system in programming education and software development, showcasing its practical benefits in enhancing code quality and supporting users' programming skill development.This study aims to deliver an automated, objective, and efficient solution for code quality evaluation to meet the growing demand for high-quality code. The research results provide theoretical support for both academia and industry and offer important design insights and practical references for future code quality evaluation tools [1].

2. Theoretical Foundation of Code Quality Evaluation

2.1. Definition and Standards of Code Quality

Code quality encompasses several key characteristics that serve as essential standards to determine if code meets development requirements and can be maintained and extended over time. These characteristics are foundational to creating software that is not only functional but also efficient, reliable, and adaptable to future changes. High-quality code simplifies development processes, reduces maintenance costs, and enhances overall software stability.The first critical characteristic of code quality is readability. Readability refers to how easily developers can understand and work with code. High readability is achieved through clear and consistent naming conventions, appropriate commenting, and a well-organized structure. Code that is easy to read allows developers to grasp its functionality quickly, facilitates collaboration, and reduces the likelihood of errors during modifications. For instance, meaningful variable names, logical indentation, and comprehensive comments ensure that other developers or future team members can easily follow the logic without extensive documentation. As a result, readability directly reduces the learning curve for new contributors and simplifies communication within teams.Another important aspect of code quality is maintainability, which indicates how easily code can be modified, debugged, or enhanced [2]. Maintainable code is modular, follows best practices for design, and minimizes dependencies. This quality enables developers to make quick updates when requirements change or when errors need fixing. A maintainable codebase allows for efficient troubleshooting, lowers the risk of introducing new bugs during modifications, and makes it simpler to add new features. Structuring code into smaller, reusable components, for instance, enhances maintainability by isolating functionality, making it easier to replace or update individual parts without affecting the entire codebase.Execution efficiency is another crucial measure of code quality, especially for applications that handle large volumes of data or require real-time processing. Efficient code maximizes performance, ensuring tasks are completed promptly and resources are utilized effectively. In scenarios requiring high concurrency or intensive data processing, execution efficiency becomes essential for maintaining fast system response times and managing computational resources, such as memory and CPU, more effectively [3]. For example, well-optimized algorithms and efficient data handling can reduce processing time significantly, contributing to a smoother user experience and cost savings in terms of resource consumption.The fourth critical component is security, which has become a priority in modern software due to increasing cybersecurity threats. Quality code minimizes vulnerabilities by adhering to security best practices, such as input validation, access control, and secure coding techniques. For example, preventing SQL injection attacks and buffer overflows can safeguard against unauthorized data access and potential application crashes. Ensuring code security is vital not only to protect sensitive data but also to maintain system reliability and user trust.To comprehensively evaluate these aspects of code quality, the industry has developed various standards and tools. Static analysis tools assess code readability and maintainability by checking syntax, code structure, and adherence to best practices. These tools, such as SonarQube and ESLint, allow developers to identify issues early in the development process, reducing errors before code is deployed. Complexity analysis tools help developers locate highly complex sections that may hinder readability and maintainability, suggesting areas where simplification or refactoring

could enhance code quality.Security standards like the CWE (Common Weakness Enumeration) framework help developers identify and mitigate common security risks. By following CWE guidelines, developers can avoid vulnerabilities that could compromise system stability and data protection. These standards and tools collectively provide a systematic framework for evaluating code quality, offering both theoretical and practical support to developers seeking to create secure, maintainable, efficient, and readable code. Through the consistent application of these standards, developers can significantly enhance the robustness and longevity of software systems.

2.2. Existing Code Quality Evaluation Methods

Currently, code quality evaluation methods are mainly categorized into static analysis, dynamic analysis, code review, and machine learning-based automated evaluation. Each method has distinct features, enabling a multi-dimensional evaluation of code quality and offering developers quality improvement suggestions. The combination of these methods helps achieve a comprehensive assessment of code quality [4].Static Analysis is an evaluation method that detects potential issues without executing the code. It examines syntax, structure, dependencies, and more, identifying defects, redundancies, and complexity issues. Common static analysis tools include SonarQube, Checkstyle, and ESLint. These tools automatically evaluate code quality based on predefined rules, generating detailed quality reports that help developers identify readability and maintainability issues. Additionally, static analysis can detect common security vulnerabilities, providing efficient support for managing code quality, particularly in large projects.Dynamic Analysis evaluates code performance and resource usage during execution, mainly identifying runtime bottlenecks and anomalies. Dynamic analysis uncovers runtime issues, such as memory leaks and processor overloads, which static analysis cannot capture. Popular dynamic analysis tools include JProfiler, VisualVM, and Dynatrace. By monitoring code execution, these tools help developers identify and optimize low-efficiency sections. However, dynamic analysis results may vary based on hardware configuration and load conditions, necessitating real production environments for accurate results.Code Review is a traditional yet effective quality evaluation method, where team members review each other's code, identifying issues and suggesting improvements. Code reviews, often conducted manually or through peer review, can identify business logic issues that tools might miss. Platforms like GitHub and GitLab support code review, allowing developers to perform reviews before code submission. Though dependent on team experience, code reviews provide detailed feedback, particularly valuable in complex projects.Machine Learning-based Automated Evaluation is a new and increasingly popular approach. By training models, machine learning can automatically analyze code structure and logic to judge quality. For example, Google's Tricorder and Facebook's Sapienz use machine learning for automated quality evaluation and vulnerability detection. These systems learn quality rules from extensive data, providing intelligent evaluation support. However, machine learning-based evaluation typically requires large datasets, and model accuracy depends on data quality, posing some challenges to its widespread application.In summary, existing code quality evaluation methods each have strengths. Static and dynamic analysis focus on code structure and performance, code review offers flexibility through human feedback, and machine learning-based automated evaluation holds strong potential for intelligent analysis. In practical applications, combining multiple methods achieves comprehensive and accurate code quality assessments, effectively enhancing evaluation results [5].

3. System Design and Architecture

3.1. System Architecture

The system architecture consists of four core modules: the front-end display module, back-end processing module, data storage module, and analysis engine module. Each

module is responsible for specific functions and works collaboratively to create a comprehensive code quality evaluation system. This architecture supports static analysis, dynamic analysis, and intelligent evaluation, ensuring users receive multi-dimensional feedback on code quality.The front-end display module provides an interactive interface for users, featuring code input, result displays, and optimization feedback. Through a simple and intuitive interface, users can upload code, view quality reports, and receive improvement suggestions. The evaluation results are presented in charts, scores, and textual descriptions, enabling users to easily understand the strengths and weaknesses of their code [6]. The front-end communicates in real time with the back-end to ensure a smooth user experience and fast response.The back-end processing module is the core of the system, handling code files received from the front end. This module includes code parsing and task scheduling. Code parsing converts user-submitted code into structured data for subsequent analysis, while task scheduling allocates tasks based on system resources to enhance overall processing efficiency.The data storage module stores code evaluation results, user code samples, and model training data. Using a distributed database, it ensures data persistence and rapid querying, supporting historical data comparison analysis. This module is also responsible for data backup and security management, ensuring stable system operation and protecting user data privacy.The analysis engine module is the primary computational component, responsible for code quality evaluation tasks. This module includes a static analysis engine, a dynamic analysis engine, and a machine learning analysis engine. The static analysis engine checks code readability, structure, and security; the dynamic analysis engine assesses performance and resource usage in a simulated runtime environment; and the machine learning analysis engine predicts code defects and optimization opportunities based on training data. These engines work together to ensure accurate and comprehensive evaluation results. The overall architecture employs a modular and distributed design, allowing each module to be independently extended and updated to adapt flexibly to changing code quality evaluation requirements [7].

3.2. Evaluation Algorithm Design

The design of evaluation algorithms is the core of the entire system, utilizing a combination of multi-dimensional, layered algorithms to perform a comprehensive analysis of code quality. The primary components of the evaluation algorithms include static analysis, dynamic analysis, and machine learning-based intelligent evaluation, each complementing the others to ensure scientific and accurate results.Static analysis algorithms evaluate the code without execution, focusing on structure, naming conventions, logical complexity, and potential security vulnerabilities. Static analysis follows a rule-based approach, checking code issues against predefined coding standards and quality metrics. To ensure thorough analysis, the design incorporates metrics such as cyclomatic complexity, code duplication, and variable naming consistency. For example, cyclomatic complexity quantifies code logic complexity, helping to identify segments that may be difficult to maintain and test; duplication detection uses hashing to identify repeated code blocks, reducing redundancy and enhancing maintainability. These static analysis algorithms can detect readability and structural issues in the early stages of development.Dynamic analysis algorithms assess code performance and resource usage during execution, monitoring performance metrics such as memory usage, CPU load, and I/O operation frequency. Dynamic analysis is designed to simulate real execution environments, using probe technology to gather resource consumption data in real time. For instance, memory leak detection uses heap monitoring and garbage collection analysis to identify unreleased memory blocks, while CPU usage detection analyzes thread frequency and task distribution to pinpoint resource-heavy code sections. Dynamic analysis helps developers identify performance bottlenecks and provides a foundation for optimization.Machine learning-based intelligent evaluation is a key innovation, using large datasets to train models that

automatically assess code quality. This module includes two main models: a quality scoring model and a defect prediction model. The quality scoring model uses labeled code samples to learn how to assign quality scores to new code, employing regression algorithms for scoring predictions; the defect prediction model applies classification algorithms such as decision trees, random forests, or neural networks to predict potential issues based on historical code and defect data. Before inputting data, the system performs feature engineering on the code, extracting features like length, complexity, and comment density to ensure model effectiveness. The evaluation results are validated through cross-validation and test sets to guarantee prediction accuracy and robustness.To provide comprehensive and accurate evaluation results, the system integrates multiple algorithms, combining outputs from static analysis, dynamic analysis, and machine learning models with weighted aggregation. The system dynamically adjusts weights based on code type and user needs; for example, dynamic analysis is prioritized in performance-sensitive code, while static analysis is weighted higher for readability-focused code. This multi-algorithm approach generates an overall score and provides detailed recommendations and optimization directions.Overall, the evaluation algorithm design emphasizes a multi-dimensional, layered evaluation approach, combining foundational static and dynamic analysis with intelligent machine learning predictions. This design enhances system accuracy and adaptability, meeting users' diverse needs for code quality improvement [8].

4. System Functional Modules

The system functional modules form the core of the code quality evaluation system, handling everything from analysis to feedback and optimization. Each module works in concert to deliver a comprehensive quality report and actionable suggestions through static analysis, dynamic performance evaluation, and security assessment. These modules provide multi-dimensional quality assessments and streamline the review process through automation, ensuring high-quality code across various dimensions.The code quality analysis module is the central part responsible for evaluating quality metrics, divided into static code analysis, dynamic performance analysis, and security detection. Static analysis checks syntax structure, naming conventions, comment completeness, and complexity to improve readability and maintainability. Dynamic performance analysis monitors runtime data such as memory, CPU, and I/O to identify performance bottlenecks and provide optimization guidance [9]. Security detection focuses on identifying vulnerabilities, such as SQL injection and buffer overflow, ensuring code safety in real-world execution. This module synthesizes the evaluation results into a quality report with recommendations, giving developers clear directions for improvement.The user feedback and learning module connects the system to users, helping them understand evaluation results while enhancing their coding skills. This module presents the quality report in an accessible format, including sub-scores, graphical analyses, and textual suggestions to help users grasp the evaluation details. Additionally, the system offers optimization tips and reference examples, guiding users in making effective improvements. Through feedback and learning, users gradually adopt better coding practices and improve their skills.The historical data storage and comparison module stores evaluation records and historical data, offering quality comparison across time periods. After each analysis, results are saved to the database, allowing users to track progress over time. The module supports quality comparisons between multiple versions, helping users identify quality changes before and after optimization. This long-term data storage and comparison function provides developers with a reference for ongoing quality improvement, encouraging sustainable code optimization practices.The machine learning model update module ensures that machine learning-based evaluation algorithms remain effective as data accumulates. This module periodically updates and retrains models based on historical evaluation data and user feedback, ensuring the accuracy and timeliness of code quality assessments. For example, the system can use labeled user feedback to fine-tune models,

maintaining high accuracy in quality scoring and defect prediction. Automated model updates allow the system to adapt to changing code quality requirements, maintaining strong competitiveness and relevance.The result synthesis and scoring module consolidates evaluation results from different modules, creating a comprehensive code quality score and report. Using weighted aggregation, this module combines static, dynamic, and security assessment scores, adjusting weights based on the dimensions most relevant to user needs. For instance, dynamic analysis weight increases for users focused on performance, ensuring the score aligns with user priorities. The final score and report include both an overall score and detailed sub-scores with suggestions, helping users understand their code's strengths and weaknesses from multiple perspectives, guiding them toward effective optimization.The functional modules in the system provide comprehensive, intelligent code quality evaluation services through multi-dimensional analysis, user feedback, historical data tracking, and model updating. This complete design helps users enhance code quality, promotes their coding skill development, and ensures the system's accuracy and practicality in code evaluation.

5. Data Processing and Machine Learning Applications

Data processing and machine learning applications are central to achieving intelligent evaluation within the system. Through data collection, preprocessing, and feature extraction, the system builds and refines machine learning models to provide precise quality predictions and assessments. Data collection gathers multi-dimensional samples from user-submitted code,including complexity, line count, naming conventions, and comments, creating a rich set of code features. To ensure data diversity, the system also pulls samples from open-source code repositories, particularly annotated segments with high and low quality, which enriches the dataset for training models.In data preprocessing, the system cleans and normalizes data, removing irrelevant information and ensuring feature consistency. For instance, variable naming conventions and length are standardized to enable uniform analysis. The system uses feature extraction techniques, transforming data into a format suitable for machine learning, focusing on core attributes like code length, complexity, and semantic structure. This process leverages NLP techniques to analyze comments for completeness and accuracy, thereby capturing logical and structural characteristics that aid in quality assessment.The machine learning model applies two primary models for quality scoring and defect prediction. The quality scoring model uses regression algorithms to assign a quality score to new code, trained on labeled examples. For instance, if a code sample exhibits proper naming conventions, logical structure, and concise comments, the model may predict a score of 85, indicating high quality. The defect prediction model employs classification algorithms to predict potential vulnerabilities and issues, such as resource leaks or boundary overflow, providing targeted suggestions for improvementWith advanced data processing and machine learning, the system delivers precise, data-driven evaluations and continuously refines model accuracy to maintain relevance. This approach enables the system to flexibly handle diverse code samples, providing developers with scientific and effective guidance for improving code quality [10].

6. System Implementation and Case Analysis of Application

The implementation of the code quality evaluation system integrates each core module to deliver comprehensive and accurate assessments. By combining front-end interface design, back-end processing capabilities, data storage solutions, and intelligent analysis engines, the system provides a full workflow from code submission to quality evaluation and feedback. The front-end interface, designed with a user-friendly approach, enables users to upload code, access evaluation results, and view detailed suggestions for improvement. In the back-end, the processing module efficiently handles code parsing and

task scheduling, ensuring smooth data flow across modules.The data storage module ensures secure and efficient storage of user-submitted code, evaluation results, and model training data using distributed databases. This module supports rapid querying and historical data comparison, enabling users to track code quality improvements over time. The analysis engine module leverages static and dynamic analysis engines to assess readability, performance, and security, while machine learning algorithms predict potential issues and generate quality scores. The integration of these engines provides users with a thorough and actionable evaluation report.

In a practical application scenario, a software development team used the system to evaluate a newly developed module for readability, maintainability, and performance. After uploading the code, the static analysis engine flagged areas with high complexity and duplicate code blocks, recommending refactoring to improve readability and reduce redundancy. The dynamic analysis engine further identified a memory leak issue through real-time resource monitoring, which allowed the developers to address the issue before deployment.Additionally, the machine learning-based defect prediction model detected a potential boundary overflow vulnerability, providing specific lines of code for review and suggesting preventive measures. The comprehensive feedback helped the team optimize the module's structure and performance while mitigating security risks. Over subsequent iterations, the historical comparison feature enabled the team to track improvements, demonstrating the system's effectiveness in fostering continuous quality enhancement.Through the seamless integration of multi-dimensional analysis and data-driven feedback, the system effectively supports both educational and practical software development environments, offering robust solutions for code quality management and optimization.

7. Conclusion

The system achieves multi-dimensional, intelligent code quality evaluation by integrating static analysis, dynamic performance evaluation, and machine learning. The modular design ensures comprehensive analysis across structure, performance, and security, providing developers with accurate scores and optimization suggestions. Data-driven machine learning applications enable continuous improvement, enhancing evaluation precision and adaptability. Overall, this system effectively improves the efficiency and scientific rigor of code quality evaluation, providing developers with solid support for code optimization and offering an innovative solution for quality management in programming education and software development.

References

1. Chen, Hsi-Min, Bao-An Nguyen, and Chyi-Ren Dow. "Code-quality evaluation scheme for assessment of student contributions to programming projects." Journal of Systems and Software 188 (2022): 111273, doi: 10.1016/J.JSS.2022.111273.
2. Chen, Hsi-Min, et al. "Analysis of learning behavior in an automated programming assessment environment: A code quality perspective." IEEE access 8 (2020): 167341-167354, doi: 10.1109/ACCESS.2020.3024102.
3. Lu, Yao, et al. "Improving students' programming quality with the continuous inspection process: a social coding perspective." Frontiers of Computer Science 14 (2020): 1-18, doi: 10.1007/s11704-019-9023-2.
4. Hijazi, Haytham, et al. "Quality evaluation of modern code reviews through intelligent biometric program comprehension." IEEE Transactions on Software Engineering 49.2 (2022): 626-645, doi: 10.1109/TSE.2022.3158543.
5. Wedyan, Fadi, and Somia Abufakher. "Impact of design patterns on software quality: a systematic literature review." IET Software 14.1 (2020): 1-17, doi: 10.1049/iet-sen.2018.5446.
6. Mekterović, Igor, et al. "Building a comprehensive automated programming assessment system." IEEE access 8 (2020): 81154-81172, doi: 10.1109/access.2020.2990980.
7. Rani, Pooja, et al. "A decade of code comment quality assessment: A systematic literature review." Journal of Systems and Software 195 (2023): 111515, doi: 10.1016/J.JSS.2022.111515.
8. Kanika, Shampa Chakraverty, and Pinaki Chakraborty. "Tools and techniques for teaching computer programming: A review." Journal of Educational Technology Systems 49.2 (2020): 170-198, doi: 10.1177/0047239520926971.

9. Afzali, Hammad, et al. "Towards verifiable web-based code review systems." Journal of Computer Security 31.2 (2023): 153-184, doi: 10.3233/JCS-210098.
10. Combéfis, Sébastien. "Automated code assessment for education: review, classification and perspectives on techniques and tools." Software 1.1 (2022): 3-30, doi: 10.3390/SOFTWARE1010002.