International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 02 | Feb - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Automated Software Quality Assurance Using CI/CD Tools

Tanay D. Jaybhaye[1], Amol D. Wakhare[2]

[1]Student, Department of CSE, Deogiri Institute of Engineering and Management Studies, Chh. Sambhajinagar, Maharashtra, India

[2]Asst. Professor, Department of CSE, Deogiri Institute of Engineering and Management Studies, Chh. Sambhajinagar, Maharashtra, India

[1]tanayjaybhaye1561@gmail.com, [2]amolwakhare@dietms.org

Abstract - Modern software development environments require fast delivery with the retention of high quality standards. Continuous Integration and Continuous Delivery (CI/CD) pipelines have greatly enhanced automation in software development; however, the current pipelines are dependent on isolated quality checks using predefined thresholds, which do not offer a holistic assessment of the readiness of the software. Software quality is a multidimensional concept that considers parameters such as reliability, security, maintainability, testing efficiency, and code duplication. The assessment of these parameters individually often results in subjective and unreliable release decisions.

This study presents a comprehensive CI/CD-based automated software quality scoring and release decisioning framework that uses a holistic model to assess multiple software quality parameters together. The framework uses a weighted multi-criteria assessment approach to normalize the parameters and calculate a Final Quality Score (FQS), which is an objective measure of software readiness. The framework automatically approves or rejects software releases based on predefined quality gate rules. Experimental results on real-world software projects show that the proposed framework is able to distinguish between high-quality and low-quality software builds effectively, enhance decision accuracy, and reduce manual effort required for release decisioning in contemporary DevOps environments.

Key Words: CI/CD Pipeline, Software Quality Evaluation, Automated Quality Gate, DevOps, Multi-Criteria Scoring Model, Software Release Decision.

1. INTRODUCTION

The fast-paced evolution of software engineering has led to a dramatic shift in the software development, testing, and deployment processes of modern software systems. The traditional software development methodologies like Waterfall involved sequential activities, where testing and validation of the software were done only after completing the development activities. This led to issues like delayed detection of defects, higher maintenance costs, and lower reliability of the software.

The advent of Agile and DevOps has brought a paradigm shift in software development, where the entire process is now automated and continuous. Continuous Integration and Continuous Delivery (CI/CD) has enabled developers to integrate code changes into a common repository at a faster pace, where automated tools compile, test, analyze, and prepare for deployment.

However, despite these improvements, the current state-of-the-art CI/CD pipelines are mostly dependent on isolated quality assessments with predefined thresholds like code coverage constraints, defect counts, or vulnerability severity levels. These isolated assessments do not represent the multi-dimensional characteristics of software quality, which have other attributes like reliability, maintainability, security, testing efficiency, and code reusability. This makes the release process subjective, inconsistent, and dependent on manual interpretations of combined results from various tools.

To overcome these challenges, the proposed research work introduces an automated software quality assessment and release decision solution integrated into a CI/CD pipeline. The solution gathers software quality data from various automated tools and aggregates them using a weighted multi-criteria assessment model to calculate a Final Quality Score (FQS). This FQS is used as an objective software readiness indicator for automated release approvals or rejections based on dynamic quality gate rules.

The proposed solution is expected to offer a consistent, automated, and data-driven approach to software quality assessment. By integrating multiple software quality tools, normalizing diverse quality metrics, and providing a single decision-making score, the solution is expected

to improve the reliability and efficiency of modern DevOps-based software development environments.

2. LITERATURE REVIEW

Recent breakthroughs in software engineering underscore the growing need for automation in software quality assurance in Continuous Integration and Continuous Delivery (CI/CD) settings. Contemporary software development methodologies focus on fast software delivery with high reliability and security requirements. Automated pipelines have become a critical component for code integration, test execution, and quality analysis in real-time settings.

Several research studies have shown that CI/CD pipelines greatly enhance software reliability with early bug detection, reduced integration problems, and continuous feedback mechanisms. Automated testing infrastructure and static code analysis tools are crucial for early bug detection, code maintainability, and potential security problems in early development phases. These tools are useful for enhancing software quality with minimal human effort and errors.

Researchers have also pointed out that software quality is a multidimensional concept and cannot be measured with a single quality metric. Contemporary quality measurement techniques use a combination of metrics such as code coverage, reliability, maintainability, and security risks to measure comprehensive software readiness. Multi-criteria decision-making models have been proposed to combine diverse quality metrics from heterogeneous sources into a single decision framework for more objective release decisions.

Dynamic quality gate tools have been proposed in contemporary CI/CD settings for automated release approval based on predefined quality criteria. These tools have improved decision-making consistency and reduced human intervention. However, current solutions have limitations in using static thresholds and lack integration with multiple quality tools. Moreover, most of the existing systems offer disintegrated reports rather than a single quality score, which is confusing for decision-makers to understand overall software readiness.

Thus, despite considerable advances in automated software quality assurance, there are still challenges in integrating multiple quality tools, normalizing diverse metrics, and developing a single automated decision framework for consistent and objective release decisions

3. PROBLEM STATEMENT

In today's CI/CD-based software development ecosystem, various automated tools are employed to evaluate different facets of software quality, such as code coverage analysis tools, static code analysis platforms, and security vulnerability scanners. While these tools provide essential quality metrics, they are all independent and provide isolated reports.

As a consequence, software development teams are left to manually analyze various quality metrics, which often leads to inconsistent, subjective, and time-consuming release decision-making processes. Current quality gate approaches are limited by fixed threshold rules that assess individual metrics rather than overall software quality holistically. This piecemeal strategy does not account for the multidimensional nature of software quality and may lead to erroneous release approvals or unjustified rejections.

Thus, there is a pressing need for an integrated automated solution that can synthesize various software quality metrics into a comprehensive evaluation framework to enable objective software readiness assessment and automated release decision-making.

4. RESEARCH GAP

Based on the literature review, the following research gaps are identified:

- Current methods of quality assessment in CI/CD are based on isolated evaluation of metrics.
- Most current systems are based on fixed threshold quality gates that are not adaptable or capable of overall quality assessment.
- Current quality assessment tools provide disjointed reports that do not offer a single quality score for decision-making.
- Manual assessment of various quality metrics is time-consuming and less consistent.
- There is limited work on automated frameworks that combine multiple quality metrics based on multi-criteria quality assessment models in CI/CD systems.

These gaps highlight the need for a unified automated quality scoring framework that integrates multiple quality tools and provides an objective and consistent release decision mechanism.

5. PROPOSED FRAMEWORK OVERVIEW

This study introduces the Integrated CI/CD-Based Automated Software Quality Scoring and Release Decision Framework, which has been designed to deliver a comprehensive and automated assessment of software quality. The framework has been designed to work in a CI/CD environment and utilize multiple software quality assessment tools to gather and process software quality data in real-time.

The framework starts with the integration of the source code into a version control system. Once the source code changes are committed to the version control system, the automated framework is triggered using a CI/CD tool. The framework then executes the build and compilation processes to ensure that the source code is executable and does not contain compilation errors.

Once the build execution is complete, the framework executes automated unit tests to verify the functional correctness of the source code. After the execution of the unit tests, the framework executes the code coverage analysis to evaluate the test cases. The framework also executes the static code analysis to detect bugs, code smells, duplications, and maintainability issues in the source code. Furthermore, the framework also executes the security vulnerability analysis to detect security risks in the source code.

All the collected data is aggregated and normalized using a multi-criteria evaluation approach. A weighted scoring model is then applied to compute the Final Quality Score (FQS), which represents the software quality.

Using the quality gates, the framework automatically determines whether the software build should be accepted or rejected for the release process. A real-time dashboard is also generated to visualize the collected data, computed scores, and release decisions.

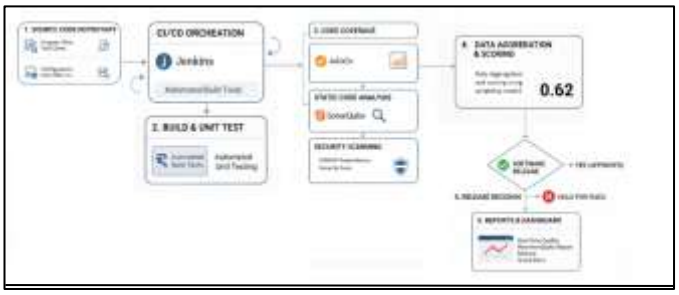

Fig.1: Proposed Automated Software Quality Evaluation Framework

6. SYSTEM ARCHITECTURE

The proposed system is designed to be an automated workflow that is part of a Continuous Integration and Continuous Delivery (CI/CD) system to continuously evaluate the quality of the software under development. The system architecture is based on a modular and layered approach to enable scalability, automation, and efficient processing of the quality data.

The system has five major components: source code repository, CI/CD orchestration, quality assessment tools, quality scoring, and visualization.

The source code of the software is initially stored in a version control system, which contains the source code, test cases, configuration data, and dependency information. Once changes are made to the source code and committed to the version control system, the CI/CD system automatically triggers the continuous quality evaluation process.

The CI/CD orchestration is responsible for build automation, test execution, and integrating various quality assessment tools. This component ensures that all the quality assessment processes are executed sequentially.

The quality assessment tools are responsible for collecting various software quality metrics. The code coverage analysis tools evaluate the test cases, static code analysis tools evaluate the source code, and security analysis tools evaluate the security risks in the source code.

The quality scoring component processes the collected data, converts the data into a normalized format, and uses a weighted model to compute the Final Quality Score (FQS). The FQS is then used by the automated quality gate decision module to determine whether the software build should be accepted or rejected.

The system also has a component that generates a real-time dashboard displaying the quality data, scoring results, and the build decisions.

Fig. 2: System Architecture of Automated Quality Evaluation Framework

7. QUALITY SCORING MODEL

A key contribution of the proposed framework is the development of a unified Quality Scoring Model that combines multiple heterogeneous software quality metrics into a single numerical evaluation value. Since different quality metrics are measured using different scales and units, normalization is required to ensure comparability.

7.1 Quality Metrics Considered

The system evaluates software quality based on the following major attributes:

- **Coverage Score (CS):** Represents testing effectiveness based on code coverage percentage.
- **Reliability Score (RS):** Derived from the number of detected bugs and critical issues.
- **Security Score (SS):** Based on vulnerability severity levels detected in dependencies.
- **Maintainability Score (MS):** Reflects code quality based on static analysis results.
- **Duplication Score (DS):** Indicates the percentage of duplicated code.

These attributes collectively represent the multidimensional nature of software quality.

7.2 Metric Normalization

Since the raw quality metric values are in different formats, they are normalized to a range of 0 to 1 using min-max normalization techniques. Higher values of the normalized data indicate higher software quality.

For the positive quality indicators like coverage and maintainability, the normalization score is calculated using the following formula:

Normalized Score = (Actual Value – Min Value) / (Max Value – Min Value)

For negative indicators like bugs and vulnerabilities, inverse normalization is done so that the lower the number of bugs, the higher the score.

7.3 Weighted Multi-Criteria Evaluation Model

Each quality attribute does not contribute equally to the overall software readiness. Hence, the weighted scoring model is used for the calculation of the Final Quality Score (FQS). The weight value for each quality attribute is predefined on the basis of its relative importance in the evaluation of software quality.

The overall quality score is calculated as follows:

FQS = (0.30 * CS) + (0.20 * RS) + (0.20 * SS) + (0.15 * MS) + (0.15 * DS)

Where:

FQS=Final Quality Score

CS, RS, SS, MS, DS = Normalized quality scores

Weight values are assigned in such a manner that the evaluation of reliability, security, maintainability, and effectiveness of the testing process remains balanced.

7.4 Automated Quality Gate Decision Logic

To support automated release decisions, the system defines a dynamic quality gate mechanism based on the computed FQS.

The decision rule is defined as follows:

If FQS ≥ 0.6 → Build Approved

If FQS < 0.6 → Build Rejected

This threshold ensures that only software builds meeting minimum quality standards are approved for release, thereby improving reliability and reducing deployment risks.

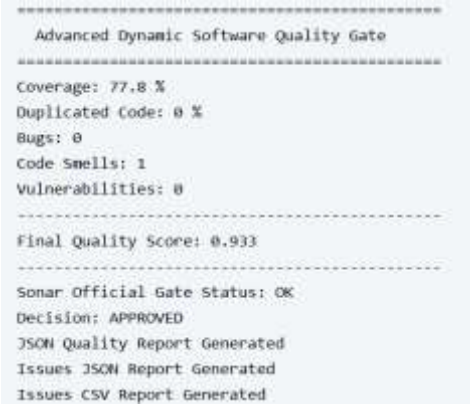

Fig. 3: Quality Gate Score

8. IMPLEMENTATION DETAILS

The proposed framework utilizes the most popular and commonly used open-source tools for its implementation, thereby making it more applicable in the real world.

For the CI/CD orchestration, an automation server is used, which manages the execution of the pipeline, build, and quality integration tools. The pipeline is set up in such a manner that it triggers automatically upon the update of the codes and executes the quality evaluation process sequentially.

For the static code analysis, a quality inspection platform is used, which identifies bugs, security, maintainability, and duplication in the codes. The code coverage analysis is performed using a coverage measurement tool, which determines the percentage of the code that gets executed during the testing phase. The security vulnerability scan is performed using a dependency analysis tool, which

identifies the commonly known security threats in the codes.

All the quality metrics are exported in the form of XML and JSON data formats from the quality tools. A quality scoring engine, which is customized, processes the quality report and utilizes the relevant quality metrics for the calculation of the Final Quality Score using the Weighted Evaluation Model.

The final results are stored in the form of centralized data and visualized in the form of an HTML-based dashboard, which displays the results in real-time for the quality status and the release decisions.

9. EXPERIMENTAL SETUP AND EVALUATION

To test the efficacy of the proposed framework of automated software quality evaluation, experiments were conducted on real-world software projects developed using the Java-based web application development paradigm. The experimental design of the proposed system aimed to test the efficacy of the system in accurately evaluating the quality of the software, computing the unified quality scores, and facilitating the decision-making process in the release of the software.

The experiments were conducted in a Continuous Integration and Continuous Deployment (CI/CD) environment, which was configured to include various tools for the automation of the build process, testing, and quality evaluation of the code. The environment was designed to automatically trigger the execution of the quality evaluation process upon updates to the codebase.

The quality evaluation process involved the sequential execution of various stages of quality evaluation, which include compilation, unit testing, code coverage analysis, static code inspection, and vulnerability scanning.

Quality metrics were collected from various integrated tools and fed into the proposed quality scoring engine. The system normalized the quality metric values and leveraged the proposed weighted multi-criteria decision-making technique to calculate the Final Quality Score (FQS).

Two different software projects were selected for quality evaluation to test the efficacy of the system in differentiating between high-quality and low-quality code.

10. EXPERIMENTAL RESULTS

10.1 Case Study 1: Low Quality Software Project

The first project reflected a software system with low quality characteristics. The project showed low code coverage, a number of detected bugs, and moderate security vulnerabilities. Static code analysis identified a number of maintainability problems in the code.

After executing the proposed system, the quality metrics were collected, and the system calculated the Final Quality Score using the weighted evaluation model. The calculated FQS for this project was below the threshold, which led to the rejection of the software release.

This shows the system's ability to detect low-quality software and reject the deployment of unreliable software.

Fig. 4: CI/CD Pipeline Execution for Low-Quality Project

10.2 Case Study 2: High Quality Software Project

The second project was a well-maintained software system, which had a high level of code coverage, few defects, high maintainability, and negligible security vulnerabilities. The static code analysis results showed the system's reliability, low code duplication, and so forth.

On execution of the pipeline, the system processed the collected metrics and calculated a higher Final Quality Score than the first project. As the calculated FQS was higher than the threshold, the software build was automatically approved for release.

This proves that the proposed framework correctly identifies a high-quality software build and facilitates a reliable decision-making process for the release of the software build

Fig. 5: CI/CD Pipeline Execution for High-Quality Project

11. QUALITY SCORE COMPARISON

Quality Aspect	CalcWebApp Project	Clean Calculator Project
Coverage Score	0.04	0.78
Reliability Score	0.6	1
Security Score	0.8	1
Maintainability Score	1	1
Duplication Score	1	1
Final Quality Score (FQS)	0.594	0.933
Decision	REJECTED	APPROVED

Table.1: Presents a comparative analysis of normalized quality scores obtained for both evaluated software projects.

The results clearly demonstrate that the proposed framework effectively distinguishes between software builds with different quality levels. The low-quality project failed to meet the minimum quality threshold, while the high-quality project successfully passed the automated quality gate.

12. DASHBOARD VISUALIZATION MONITORING

To improve the usability and real-time monitoring of the system, the proposed system introduces an automated web-based software quality dashboard. The dashboard uses JSON data generated during the execution of the pipeline and provides a visual representation of quality attributes, computed scores, and release decisions.
The quality attributes are shown in a detailed manner, including code coverage percentage, identified bugs, security vulnerability levels, maintainability scores, and duplication levels. In addition, the Final Quality Score is shown in a graphical format, along with approval or rejection status.

This provides a high level of transparency, saving time and effort in interpreting multiple reports, and allows developers and project managers to easily monitor the readiness of the software.

Fig. 6: Automated Software Quality Dashboard Interface

13. PERFORMANCE DISCUSSION

From the above experimental evaluation, it is clear that the proposed framework has various advantages over traditional software quality assessment.
Firstly, the proposed framework has successfully integrated various quality assessment tools into a single system, which helps to avoid fragmented reporting, a common problem in traditional CI/CD environments.
Secondly, the weighted multi-criteria scoring model has offered a holistic approach to representing software quality by considering various attributes simultaneously, rather than depending on individual metric thresholds.
Thirdly, automated quality gates' decision logic has helped to achieve objective and data-driven decisions for releasing software.
Moreover, the real-time dashboard has helped to improve monitoring efficiency.
From the above discussion, it is clear that the proposed framework has successfully improved software quality governance, reliability, and decision-making in modern DevOps environments.

14. CONCLUSION

In the modern software development environment, ensuring high-quality software and quick delivery is a major challenge. Conventional quality evaluation mechanisms require manual interpretation of multiple, independent quality report results, resulting in inconsistent, time-consuming, and subjective release decisions. In addition, conventional CI/CD quality gate mechanisms only utilize isolated threshold rule sets to measure and evaluate software quality, which does not consider the multidimensional nature of software quality.
The proposed research aimed to develop a new integrated CI/CD-based automated software quality

scoring and release decision system to address the limitations of conventional quality evaluation mechanisms and provide a comprehensive, objective, and automated approach to software quality evaluation. In this regard, the proposed system integrates multiple conventional quality assessment tools to automatically collect software quality metrics related to testing effectiveness, reliability, maintainability, and security.

The main achievement of the proposed research is the development of a new unified weighted multi-criteria evaluation model to normalize the diverse quality metrics and calculate a Final Quality Score (FQS) to represent the overall quality of the software product. Using a unified score, the proposed system facilitates the interpretation of quality and supports the automation of release decisions.

The experimental evaluation of the proposed system, conducted on real-world software projects, confirmed the effectiveness of the proposed system in differentiating high-quality and low-quality software. The proposed system successfully automated the entire quality evaluation process, saving time and effort, and enhancing transparency via real-time visualization dashboards.

The proposed system provides a reliable, scalable, and data-driven approach to automated software quality governance in the contemporary DevOps environment, ensuring significant improvements to release reliability and continuous quality monitoring throughout the entire software development lifecycle.

15. FUTURE WORK

While the proposed framework shows promising performance and practicality, several improvements can be taken into consideration as part of the future work.

Future work may be directed towards integrating machine learning approaches to optimize the weights of quality attributes based on the project's historical data and defect trends. This would allow the development of intelligent quality evaluation models with the ability to improve the accuracy of the decision process over time.

Further, the proposed framework may be extended to support predictive analytics approaches for predicting the future trends of software quality and identifying areas of potential risks during the deployment phase. Integration with cloud-based DevOps environments and containerized deployment platforms may be another extension of the proposed framework for improving its scalability and real-time monitoring support.

Another promising extension of the proposed framework may be the inclusion of other software quality attributes, such as performance, usability, and monitoring, for developing a holistic quality evaluation model.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the department and institution for providing the necessary infrastructure and support required to carry out this research work. The guidance and encouragement received during the development and evaluation of the proposed system are greatly acknowledged.

REFERENCES

[1]K. Ramdass and S. Jain, "The Role of DevSecOps in Continuous Security Integration in CI/CD Pipelines," *Journal of Quantum Science and Technology*, vol. 6, no. 2, pp. 45–58, 2025.

[2]S. Chittala, "Impact of CI/CD Automation on Developer Productivity and Software Quality," *Journal of Software Engineering Applications*, vol. 17, no. 1, pp. 12–24, 2024.

[3]D. Lopez and J. Garcia, "Quality Improvement using Automated CI/CD Pipelines in DevOps," *Journal of Software Engineering Research*, vol. 11, no. 3, pp. 201–215, 2023.

[4]R. Kumar and A. Singh, "Shift-Left Testing in CI/CD Environments," *International Journal of Software Engineering and Applications*, vol. 14, no. 2, pp. 33–45, 2023.

[5]Y. Zhang, H. Li, and X. Wang, "Automated Quality Assurance in DevOps-Based CI/CD Pipelines," *Journal of Systems and Software*, vol. 188, 2022.

[6]J. Wang, L. Chen, and Y. Liu, "Automated Testing in CI/CD Pipelines," *IEEE Software*, vol. 39, no. 3, pp. 55–63, 2022.

[7]H. Yang, K. Zhou, and F. Liu, "Testing Maturity and Deployment Performance in CI/CD," *Journal of Systems Engineering*, vol. 27, no. 4, pp. 301–312, 2022.

[8]S. Soares, R. Oliveira, and M. Ribeiro, "Adoption of Continuous Integration and its Impact on Software Quality," *Information and Software Technology*, vol. 134, 2021.

[9]E. Rahman and L. Williams, "Continuous Quality Assurance in DevOps," *IEEE Software*, vol. 38, no. 5, pp. 72–79, 2021.

[10] Sharma and S. Gupta, "Automated Software Quality Monitoring using CI/CD Pipelines," *International Journal of Software Engineering and Applications*, vol. 12, no. 3, pp. 45–56, 2021.

[11] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps*, IT Revolution Press, 2018.

[12] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010.

[13] M. Fowler, *Continuous Integration: Improving Software Quality and Reducing Risk*, Addison-Wesley, 2006.

[14] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, Addison-Wesley, 2015

[15] R. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2015.

[16] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, CRC Press, 2014.

[17] ISO/IEC 25010, *Systems and Software Quality Requirements and Evaluation (SQuaRE) — Software Quality Model*, ISO, 2011.

[18] J. Bell, "Static Code Analysis for Software Quality Improvement," *IEEE Computer*, vol. 51, no. 4, pp. 80–85, 2018.

[19] D. Spinellis, *Code Quality: The Open Source Perspective*, Addison-Wesley, 2006.

[20] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*, Springer, 2006.

[21] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, 2004.

[22] T. Mens and T. Tourwé, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[23] SonarSource, "SonarQube Documentation." [Online]. Available: https://docs.sonarqube.org

[24] OWASP Foundation, "OWASP Dependency Check Documentation." [Online]. Available: https://owasp.org/www-project-dependency-check/

[25] Jenkins Project, "Jenkins User Documentation." [Online]. Available: https://www.jenkins.io/doc/

[26] JaCoCo Team, "JaCoCo Code Coverage Library." [Online]. Available: https://www.jacoco.org