Quality Gates in Software Development: Concepts, Definition and Tools

Nadica Uzunova[1], Luka Pavlič[1] and Tina Beranič[1]

[1]*Faculty of Electrical Engineering and Computer Science - University of Maribor, Koroška cesta 46, Maribor, Slovenia*

Abstract

In the domain of software development, it is crucial to maintain high quality in order to meet the increasing demands of users and the rapid pace of technological advancement. This paper explores the concept of quality gates, looking into their historical origin and the methodologies for their establishment and implementation within the software development life-cycle. Quality gates are defined checkpoints ensuring the software meets predefined quality criteria before proceeding to the following stages. These gates include various criteria, where the balance between manual and automated quality assurance is crucial. The paper also highlights tools that support quality gates, emphasising their role in the quality assurance field.

Keywords

Quality assurance, Software development, Software engineering, Automatization, Static code analysis, Tools,

1. Introduction

The software development industry is characterised by rapid innovation and increasing user expectations, which demand a strong emphasis on software quality. Quality in software is multi-dimensional and includes aspects such as functional accuracy, user experience, reliability, performance and security [1]. As organisations increasingly rely on software solutions for their core operations, the importance of maintaining high-quality software has reached unprecedented levels.

To address these demands, our research explores the concept of "quality gates". Originating from the manufacturing industry, quality gates have been adapted to fit the specific needs of software development. They serve as predefined checkpoints that evaluate software quality based on specific criteria before allowing the software to proceed to the next stage of development. This approach helps in preventing the reproduction of defects, reducing rework and ensuring that the software meets high standards of quality at every phase [2].

This paper addresses three primary research questions:

- **RQ1: What are quality gates and how do they relate to software quality?** This question explores the definition, purpose, and significance of quality gates.
- **RQ2: How are quality gates established and implemented?** This question examines the methodologies and criteria for establishing effective quality gates.
- **RQ3: What tools are available to support quality gates?** This question reviews tools that facilitate the implementation and monitoring of quality gates.

We have organised this paper around our research questions in three main sections. The first section discusses the definitions and historical background of quality gates, establishing their foundational importance in software development. In the next section, we delve into the methodologies for defining quality gate criteria, balancing manual and automated processes, and the steps involved in their implementation. And lastly, we review various tools that support the implementation and monitoring of quality gates, evaluating their features and benefits.

SQAMIA 2024: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 9–11, 2024, Novi Sad, Serbia

nadica.uzunova@um.si (N. Uzunova); luka.pavlic@um.si (L. Pavlič); tina.beranic@um.si (T. Beranič)
0009-0005-7661-2606 (N. Uzunova); 0000-0001-5477-4747 (L. Pavlič); 0000-0001-6518-5876 (T. Beranič)

2. The concept of Quality Gates

To fully grasp the significance of quality gates, it is essential to define what they are and how they function within the software development life-cycle.

While there is no one standard definition of what quality gates are, Vladimir Ambartsoumian et al. in their research paper have addressed some key definitions from various authors [3]:

- Charvat (2003) describes quality gates as a mechanism that employs formal checklists throughout the project's life-cycle. At each gate, there is a formal sign-off and acceptance process, which includes an assessment of the product's quality and integrity. Additionally, it ensures that critical information is communicated to the appropriate stakeholders to maintain transparency and accountability [4].
- Flohr (2008) defines quality gates as critical milestones and decision points within a project. These gates are characterised by predefined criteria that focus specifically on quality. This approach helps in making informed decisions about whether the project should proceed to the next phase based on meeting these quality-focused criteria [2].
- Schneider (2004) views quality gates as checkpoints that consist of a set of predefined quality criteria. These criteria must be met for a project to move from one stage of its life-cycle to the next. This definition emphasises the role of quality gates in ensuring that each phase of the project meets specific quality standards before progressing [5].

Based on these definitions, we describe quality gates as follows:

Quality gates are predefined checkpoints within the software development life-cycle, positioned at various stages of the project. At these checkpoints, specific criteria must be satisfied before the project can advance to the next phase.

2.1. Historical Context and Evolution

Historically, the concept of quality gates originates from the manufacturing sector, where they ensured product quality at various production stages [2, 6]. For instance, the RMG (ready-made garment) industry has successfully implemented quality gates to enhance product quality and reduce defects, highlighting their effectiveness in different contexts. In the RMG industry, quality gates are used to aggregate checking steps and minimise the effort for quality control by summarising information on quality parameters of work-in-process products [7].

The evolution of software quality standards began with individual researchers developing initial quality models, which eventually led to formal standards like ISO/IEC 9126 in 1991 and its subsequent modifications. Initially, the implementation of quality gates in software development was informal, relying on individual project managers to enforce quality criteria. Over time, structured processes such as the Stage-Gate® process emerged, guiding projects from concept to completion through well-defined phases and decision points [2, 8].

Today, quality gates are easily adopted in modern development frameworks like Agile and DevOps, which emphasise continuous integration and delivery. This integration allows for ongoing quality assurance and early issue detection throughout the development cycle [9].

2.2. Relationship with Quality Standards and Metrics

Having explored the historical context, we now turn to the relationship between quality gates, quality standards, and metrics. Quality gates are intrinsically linked to established quality standards, serving as practical mechanisms for ensuring compliance at various stages of the software development life-cycle. Standards such as ISO/IEC 25010 define essential software quality attributes, including functionality, reliability, usability, efficiency, maintainability, and portability [10, 11, 12]. Quality gates operationalize these standards by embedding them into the development process through specific, measurable criteria and metrics [11].

For instance, ISO/IEC 25010 provides a comprehensive framework for evaluating software quality, detailing characteristics and sub-characteristics that software must exhibit to be considered high quality [10]. These attributes are translated into actionable metrics within quality gates. Examples include [11, 12]:

- **Performance Efficiency**: Evaluated through metrics like response time and resource utilisation.
- **Reliability**: Measured by metrics including fault tolerance and availability.
- **Security**: Ensured through metrics such as confidentiality, integrity, and accountability.

Integrating these standards into quality gates ensures that software development processes align with internationally recognised benchmarks. For example, CGM CompuGroup Medical obtained ISO/IEC 25000 certification for their decision support software, evaluated for functional suitability and maintainability by AENOR and AQC Lab. This certification demonstrates adherence to high-quality standards, ensuring the software's reliability and effectiveness [10].

On the other hand, metrics play a pivotal role in the effective implementation of quality gates by providing quantifiable measures of software quality attributes. These metrics enable the objective assessment of software at various stages of development, ensuring compliance with predefined quality criteria [2]. Examples of key metrics include [13]:

- **Cyclomatic Complexity**: Measures the complexity of the code by quantifying the number of linearly independent paths through the program. Lower complexity often indicates more maintainable and understandable code.
- **Code Coverage**: Indicates the percentage of the code-base that is covered by automated tests. Higher coverage suggests more thorough testing and potentially fewer undiscovered bugs.
- **Mean Time Between Failures (MTBF)**: A reliability metric that estimates the average time between system failures, reflecting the robustness of the software.
- **Vulnerability Density**: Measures the number of security vulnerabilities per thousand lines of code (KLOC), helping to assess the security posture of the software [11].

Such metrics not only provide a clear picture of the software's current quality but also highlight areas that require improvement, facilitating continuous enhancement of the development process. Metrics are crucial for defining criteria based upon which quality gates will be evaluated [2].

3. Establishing and implementing Quality Gates

The process of defining quality gates usually begins with the identification of project goals. This involves clearly defining the overall goals of the project and ensuring that these goals align with the organisational objectives. This alignment provides a clear direction for quality assurance efforts, ensuring that the project stays on track and meets the desired outcomes. Once the project goals are defined, the next step is to formulate specific questions based on these goals. These questions are designed to determine if the goals are being met and should cover various aspects of software quality, such as performance, security, and maintainability. By addressing these questions, the team can focus on critical areas that influence the overall quality of the software [14, 15].

After formulating the questions, it is essential to identify appropriate metrics that can provide answers to these questions. These metrics should be measurable, relevant, and aligned with industry standards and best practices. Choosing the right metrics ensures that the team can accurately assess whether the software meets the predefined quality criteria. Setting acceptable thresholds for each metric is the next step. These thresholds represent the desired values that must be met for the criteria to be considered satisfied. For example, a threshold for code coverage might be set at 85%, meaning that at least 85% of the code must be covered by automated tests. Setting clear thresholds helps in objectively evaluating the software quality at each stage of the development process [2, 9].

Once the metrics and thresholds are defined, it is crucial to document each criterion clearly. This documentation should include details such as the metric, the desired value, the predicate for evaluation, and the object of measurement. Clear documentation ensures that all stakeholders understand the criteria and can align their efforts accordingly. Finally, it is important to regularly review and refine the criteria based on feedback and changing project requirements. Continuous improvement is essential to ensure that the criteria remain relevant and effective throughout the project life-cycle. Regular reviews help in adapting the criteria to new challenges and ensuring that the quality gates effectively contribute to the overall project success [2, 9, 6].

Here are some examples that illustrate the application of the mentioned criteria:

- **Code Complexity:**
  - **Metric:** Cyclomatic complexity
  - **Desired Value:** Less than 10 for any single method
  - **Implementation:** Using static analysis tools, cyclomatic complexity is computed for each method. If any method exceeds a threshold of 10, it triggers a review and refactoring process [1].
- **Bug Density:**
  - **Metric:** Number of bugs per thousand lines of code (KLOC)
  - **Desired Value:** Less than 0.5 bugs per KLOC
  - **Implementation:** During testing, bug density is tracked and calculated. If it exceeds the acceptable level, the development team focuses on resolving the issues before proceeding [16].

Defining criteria for quality gates involves several key dimensions that impact project management and outcomes. These dimensions include individuality of definition, timing of definition, and method of definition. Each dimension has distinct advantages and challenges that must be considered to optimise the effectiveness of quality gates.

Criteria for quality gates can be standardised across all projects or tailored to specific projects. Standardised criteria ensure consistency and comparability, facilitating uniform quality policies and easier benchmarking. However, they may require contextual interpretation for specific projects. Tailored criteria, on the other hand, are highly relevant to individual project contexts but are more resource-intensive and complicate cross-project comparisons due to the lack of uniform benchmarks.

The timing of criteria definition is critical and can occur at various project stages, including the project start, planning phase, or execution phase. Early definition allows for stronger management influence and longer visibility for developers, ensuring criteria alignment with initial project goals. Later definition incorporates detailed project information, resulting in more relevant criteria but with less time to influence project direction.

The method of defining criteria greatly affects their relevance and measurability. A systematic top-down approach, such as the Goal Question Metric (GQM) method, ties criteria directly to business objectives and makes them quantifiable, providing clear targets for project teams. An experience-based approach leverages senior developers' intuition and is quicker and less resource-intensive but may lack comprehensive goal coverage and result in more trivial criteria [2].

Balancing these dimensions allows for the definition of criteria that are both relevant and effective, ensuring quality gates enhance project quality and visibility. The choice of criteria definition strategy should align with the project's phase and available resources to maximise the impact of quality gates.

3.1. Manual vs. Automated Processes in Quality Gates

The implementation of quality gates can involve both manual and automated processes, some of which are shown in Figure 1. Each approach has its strengths and limitations, and often, a combination of both provides the most effective quality assurance strategy [17].

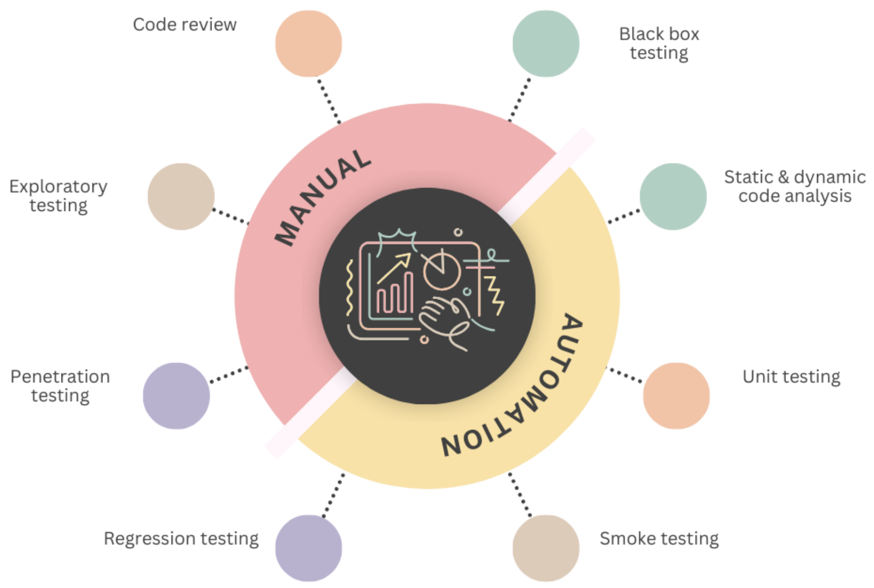


Figure 1: Manual vs. Automation processes

Manual processes in quality gates involve human intervention, such as code reviews and manual testing. These processes provide critical human insights, catching contextual and logical errors that automated tools might miss and fostering collaboration through peer reviews and discussions. Manual processes are essential for complex scenarios and usability testing, where human judgement is crucial. Key examples include:

- **Code Reviews:** Developers manually review code changes to ensure adherence to coding standards and best practices. This process can catch issues such as poor code structure or deviations from architectural guidelines, as well as facilitate knowledge transfer and team awareness [18, 19].
- **Manual Testing:** Testers execute tests manually, simulating user interactions to identify usability issues and ensure the software behaves as expected in real-world scenarios [19].

Automated processes use tools and scripts to evaluate software against predefined criteria, including automated testing, static code analysis, and continuous monitoring. These processes offer speed, efficiency, and consistency, quickly executing a large number of tests and analyses uniformly across the code base. Automation handles large projects efficiently, supporting scalability. Examples include:

- **Automated Testing:** Tools run automated tests to validate code functionality and performance, significantly reducing the time and effort required for testing [19].
- **Static Code Analysis:** Tools like SonarQube analyse the code for bugs, security vulnerabilities, and code smells, providing immediate feedback to developers and integrating with continuous integration pipelines to enforce quality standards automatically [18].

Combining manual and automated processes provides the most effective quality assurance strategy [17]. Automated tools handle repetitive and time-consuming tasks, providing quick feedback and ensuring consistency, while manual reviews add human insight to identify complex and context-specific issues [20]. This integrated approach ensures comprehensive coverage and maintains high software quality throughout the development life-cycle. An example of this integrated approach would be:

- **Manual Code Reviews and SonarQube Integration:** Developers perform manual code reviews to assess the structure, design, and logic of the code. Simultaneously, SonarQube performs automated static analysis, checking for code smells, bugs, and security vulnerabilities. If SonarQube identifies issues, it can block the build, prompting developers to address these issues before the code is merged. This combination ensures that both human insights and automated checks are utilised, providing a comprehensive quality assessment [18, 19].

4. Tools supporting Quality Gates

In this section, in relation to Research Question 3 (RQ3), we explore tools that support quality gates by automating, monitoring, and enforcing quality criteria. We will present five tools that frequently appeared in our searches for independently supporting the implementation of quality gates: SonarQube, Sigrid, Maverix.ai, Veracode, and Squore.

SonarQube is an open-source platform with enterprise options, providing continuous inspection of code quality through static code analysis. It detects bugs, code smells, and security vulnerabilities while measuring code coverage and maintainability. SonarQube integrates directly into the development pipeline, enforcing code quality standards by blocking builds if certain criteria, such as minimum code coverage or maximum bug count, are not met. This ensures that only code meeting predefined quality standards advances through the development process [21].

Sigrid by Software Improvement Group (SIG) is a paid tool designed for software quality and risk management. It provides in-depth analysis of code quality and architecture, monitors technical debt, and ensures compliance with standards. Sigrid supports quality gates by continuously monitoring code quality and providing detailed reports on various metrics. It can automatically block code changes that do not meet quality criteria, ensuring that all code meets the required standards before progressing [22].

Maverix.ai is a paid AI-driven tool for code quality analysis. It uses artificial intelligence to detect code issues, suggest improvements, and enforce coding standards. Maverix.ai supports quality gates by providing real-time feedback on code quality and blocking code that does not meet predefined standards. This tool helps teams maintain high quality by ensuring that only code adhering to best practices and standards is integrated into the main codebase [23].

Veracode is a paid application security platform offering static analysis, dynamic analysis, and software composition analysis to identify security vulnerabilities. It supports quality gates by integrating security checks into the development process, blocking code that does not meet security standards from progressing. Veracode aims to provide detailed reports on security issues and help developers address vulnerabilities early in the development lifecycle [24].

Squore by Vector is a software quality assessment tool that combines both static and dynamic analysis to provide a comprehensive evaluation of software quality. It supports quality gates by offering detailed insights into code quality, maintainability, and technical debt. Squore integrates with various development tools and environments to continuously monitor and report on code quality metrics. It can enforce quality standards by blocking code that does not meet predefined criteria, ensuring that only high-quality code is integrated into the main codebase. Squore's dashboards and reports provide actionable insights, helping teams improve code quality and reduce risks throughout the development process [25].

As these tools were selected as examples, it is important to mention that there are many other tools that support quality gates in one way or another - either by simulating quality gate behavior or integration with other tools. Some examples include Jenkins, which can simulate quality gate behavior through plugins and GitLab, which integrates with various quality tools trough CI/CD pipelines.

4.1. SonarCloud

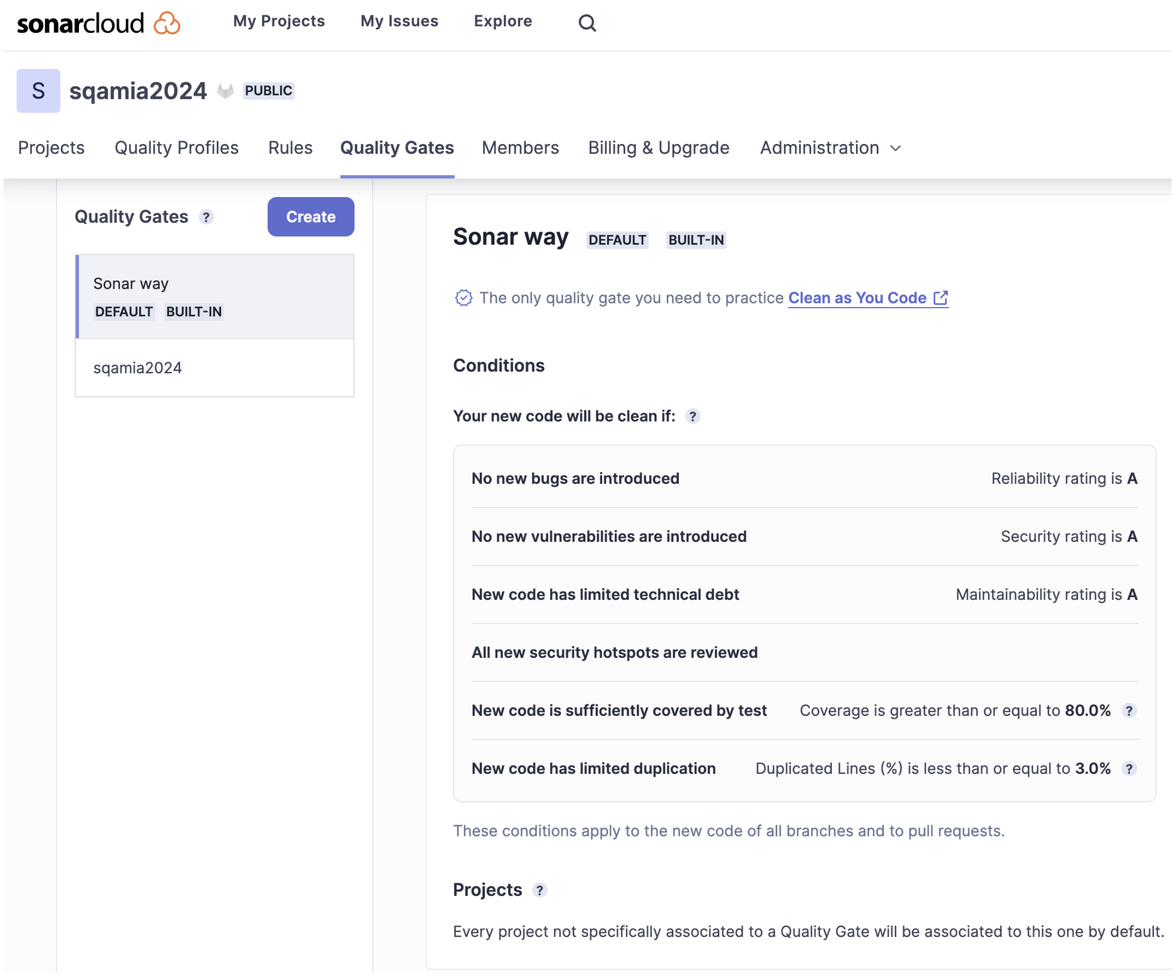
SonarCloud - SonarCube's cloud version, is selected for a detailed overview as it provides a free version and represents a comprehensive example of automatic quality gate setting, measuring, and tracking relevant to the context of this study [26].

SonarCloud supports quality gates by automating the assessment and enforcement of predefined code quality criteria [27]. It uses a variety of standards and metrics, which include [26, 1, 28]:

- **Bug Detection:** Identifies and categorizes bugs, facilitating prompt resolution of critical issues. Builds can be blocked if the number of critical bugs exceeds a certain limit.
- **Code Smells:** Detects maintainability issues that could lead to technical debt if unresolved. SonarQube can enforce refactoring by setting thresholds for acceptable code smells.
- **Security Vulnerabilities:** Identifies and reports security weaknesses to prevent potential breaches. Code with severe vulnerabilities can be blocked from progressing.
- **Duplications:** Promotes code reuse and maintainability by measuring the amount of duplicate code. High duplication rates can trigger build failures, encouraging developers to write cleaner code.

SonarCloud integrates with CI/CD pipelines, including popular tools like Jenkins, GitHub, and Azure DevOps. This integration allows for automatic scanning and enforcement of quality gates at various stages of the development lifecycle. For instance, a GitHub Actions pipeline can be configured to run SonarQube analysis after each build, automatically failing builds that do not meet the quality gate criteria [26, 28].

Quality gates, as displayed in Figure 2 in SonarCloud are configured through a combination of predefined and custom criteria. Users can set thresholds for various metrics, such as code coverage, bug counts, and duplications. These thresholds determine whether a build passes or fails based on the quality metrics evaluated during the analysis [26, 28].



Sonar way DEFAULT BUILT-IN

The only quality gate you need to practice Clean as You Code

Conditions

Your new code will be clean if:

No new bugs are introduced	Reliability rating is **A**
No new vulnerabilities are introduced	Security rating is **A**
New code has limited technical debt	Maintainability rating is **A**
All new security hotspots are reviewed	
New code is sufficiently covered by test	Coverage is greater than or equal to **80.0%**
New code has limited duplication	Duplicated Lines (%) is less than or equal to **3.0%**

These conditions apply to the new code of all branches and to pull requests.

Projects

Every project not specifically associated to a Quality Gate will be associated to this one by default.

Figure 2: SonarCloud quality gates

There are several options for configuring quality gates in SonarCloud:

- **Predefined quality gates**: SonarCloud comes with default quality gates that cover common metrics and thresholds suitable for many projects.

- **Custom quality gates**: Users can define custom quality gates with accordance their specific project needs. This includes setting specific conditions for metrics like new bugs, code coverage on new code, and security vulnerabilities [26, 28].

These quality gates are applied at the project level, ensuring that each project adheres to its unique quality standards. Configuration involves specifying which quality gate applies to the project and adjusting metric thresholds as needed [26, 28].

In the background, SonarCloud performs static code analysis using its extensive rule set, continuously integrating with the development pipeline to provide real-time feedback. This automated process ensures that any deviations from the defined quality standards are promptly identified and addressed, maintaining high code quality throughout the development lifecycle [26, 28].

To summarize, SonarCloud examplifies how automated tools can support quality gates by providing comprehensive metrics and coherent integration into development pipelines. By automating the assessment and enforcement of quality criteria, SonarQube helps maintain high standards throughout the software development lifecycle [26, 28]. This overview demonstrates the importance of such tools in ensuring consistent code quality, reducing defects, and enhancing overall software reliability, making it a valuable reference for this study's examination of quality gates.

5. Conclusion

This study has explored the fundamentals of quality gates, analyzing their theoretical foundations and practical applications. By synthesizing insights from studies, we have demonstrated that quality gates act as control points in the software development process, ensuring high-quality outputs through systematic checks at various stages. The integration of both manual and automated processes within quality gates provides a balanced approach, leveraging the strengths of each to enhance overall software quality. Tools such as SonarQube, Sigrid, Maverix.ai, Veracode, and Squore play critical roles in automating and enforcing quality criteria, while manual processes ensure contextual accuracy and human judgement.

Our findings for Research Question 1 define quality gates as predefined checkpoints within the software development life cycle, established to ensure that specific quality criteria are met before progressing to the subsequent phase. These gates serve as critical control points, assessing various software attributes such as code coverage, bug density, and compliance with coding standards to maintain the project's overall quality.

Regarding Research Question 2, our research indicates that the definition and implementation of quality gates involve collaborative efforts among stakeholders, emphasizing metrics derived from industry standards and organizational objectives. The implementation process includes setting explicit criteria, thoroughly documenting them, and consistently reviewing and refining these criteria. This approach employs both manual and automated methods to achieve comprehensive quality assurance.

In response to Research Question 3, we have identified several tools that facilitate the support of quality gates. Notable examples include SonarQube, Sigrid, Maverix.ai, Veracode, and Squore. These tools provide functionalities such as static code analysis, real-time feedback, and security vulnerability detection. By utilizing these tools, organizations can effectively enforce quality criteria and ensure high software quality throughout the development life cycle.

By addressing these research questions, this paper provides a comprehensive understanding of quality gates, their definition, implementation, and the tools that support them, thereby contributing to the enhancement of software development practices. Future research will focus on empirical studies to evaluate the effectiveness of quality gates in various development environments and methodologies. Additionally, investigating the integration of quality gate tools with other development platforms and exploring their adaptation to emerging technologies such as AI and machine learning will provide valuable insights. Furthermore, we aim to explore the best practices in criteria definition, their application dependency, and how to customize criteria throughout the project lifecycle. This will involve detailed studies to understand how different project phases can influence the effectiveness of quality gates and to develop guidelines that can be adapted to various software development contexts. By expanding our research in these areas, we hope to provide a more robust and comprehensive framework for implementing quality gates in software development.

Acknowledgments

The authors acknowledge financial support from the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0057).

References

[1] C. Vassallo, F. Palomba, A. Bacchelli, H. C. Gall, Continuous code quality: Are we (really) doing that?, in: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018, pp. 790–795. doi:10.1145/3238147.3240729.

[2] T. Flohr, Defining suitable criteria for quality gates, in: R. R. Dumke, R. Braungarten, G. Büren, A. Abran, J. J. Cuadrado-Gallego (Eds.), Software Process and Product Measurement, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 245–256.

[3] V. Ambartsoumian, J. Dhaliwal, E. Lee, T. Meservy, C. Zhang, Implementing quality gates throughout the enterprise it production process, Journal of Information Technology Management 22 (2011) 28–38.

[4] J. P. Charvat, How to use quality gates to guide IT projects, https://www.techrepublic.com/article/how-to-use-quality-gates-to-guide-it-projects/, 2003.

[5] R. Schneider, Quality gates: A new device for evaluation in cross-lingual information retrieval, in: Proceedings of the LREC-2004 Workshop of Transparency and Integration in Cross-Lingual Information Retrieval, Lissabon, Portugal, 2004.

[6] D. Steidl, F. Deissenboeck, M. Poehlmann, R. Heinke, B. Uhink-Mergenthaler, Continuous software quality control in practice, in: 2014 IEEE international conference on software maintenance and evolution, IEEE, 2014, pp. 561–564.

[7] Q. Rhaman, M. F. Hossain, A. Rahi, Quality management for rmg's: A quality gate concept, 2023.

[8] R. G. Cooper, The stage-gate idea to launch system, Wiley International Encyclopedia of Marketing (2010).

[9] G. Schermann, J. Cito, P. Leitner, H. C. Gall, Towards quality gates in continuous delivery and deployment, in: 2016 IEEE 24th international conference on program comprehension (ICPC), IEEE, 2016, pp. 1–4.

[10] N. Norbert, K. András, Review of softwarequalityrelated iso standards (2021).

[11] M.-C. Lee, Software quality factors and software quality metrics to enhance software quality assurance, British Journal of Applied Science & Technology 4 (2014) 3069–3095.

[12] B. Dugalic, A. Mishev, Iso software quality standards and certification. (2012).

[13] M.-C. Lee, Software measurement and software metrics in software quality, International Journal of software engineering and its application 7 (2013) 15–33.

[14] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, H. C. Gall, Context is king: The developer perspective on the usage of static analysis tools, in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2018, pp. 38–49.

[15] L. Pavlič, M. Okorn, M. Heričko, The use of the software metrics in practice, in: CEUR Workshop Proc., volume 2217, 2018, pp. 27–30.

[16] F. Nyberg, J. Skogeby, Code quality & quality gates (2019).

[17] E. Enoiu, D. Sundmark, A. Causevic, P. Pettersson, A comparative study of manual and automated testing for industrial control software, 2017, pp. 412–417. doi:10.1109/ICST.2017.44.

[18] A. Bacchelli, C. Bird, Expectations, outcomes, and challenges of modern code review, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 712–721.

[19] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, G. Bavota, Towards automating code review activities, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 163–174.

[20] A. Bhanushali, Impact of automation on quality assurance testing: A comparative analysis of manual vs. automated qa processes, International Journal of Advances in Scientific Research and Engineering 26 (2023) 39.

[21] SonarSource SA, SonarQube 10.5 Documentation, https://docs.sonarsource.com/sonarqube/latest/, 2024.

[22] Software Improvement Group, Sigrid | Software Assurance Platform, https://docs.sigrid-says.com/how-does-sigrid-work, 2023.

[23] Maverix.ai, Maverix.ai - Knowledgebase, https://maverix.ai/help/mergedProjects/KB/Knowledgebase.htm, 2024.

[24] Veracode, Veracode Docs, https://docs.veracode.com/, 2024.

[25] Vector Informatik GmbH, Squore Documentation, https://squore.vector.com/doc/user-guides/welcome.html, 2024.

[26] SonarSource SA, SonarCloud Documentation, https://docs.sonarsource.com/sonarcloud/, 2024.

[27] R.-H. Pfeiffer, The impact of continuous code quality assessment on defects, in: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2021, pp. 624–628. doi:10.1109/ICSME52107.2021.00069.

[28] A. Puspaningrum, M. A. A. Hilmi, M. Mustamiin, M. I. Ginanjar, et al., Vulnerable source code detection using sonarcloud code analysis, arXiv preprint arXiv:2307.02446 (2023).