DOI:10.5937/IIZS25387A

SOFTWARE QUALITY MONITORING IN AGILE WORKING ENVIRONMENT

Vuk Amizic[1], Igor Vecstejn[1], Nemanja Tasic[1] Tamara Milic[1]
[1]University of Novi Sad, Technical Faculty "Mihajlo Pupin", Zrenjanin, Serbia
e-mail: vuk.amizic@tfzr.rs, igor.vecstejn@tfzr.rs, nemanja.tasic@tfzr.rs, tamara.milic@tfzr.rs,

Abstract: The Goal of this paper is to review software quality in agile environments as well as software quality monitoring in agile working environments. An overview of ISO standards for different quality aspects has also been reviewed, as it proposes the international standardization scope for quality evaluation. Besides functional and non-functional aspects of software quality during the software development lifecycle, psychological factors were also considered.
Key words: software quality monitoring, agile working environment, ISO, quality standardization, functional aspects, non-functional aspects, software development lifecycle

INTRODUCTION

This research focuses on the quality aspects of software development, along with the agile development approach used throughout the development process. By analyzing three different ISO standards related to quality (ISO/IEC 25010:2023, ISO/IEC/IEEE 12207:2017, and ISO/IEC 25019:2023), this review outlines a strategy for building high-quality software products within an Agile team environment. Mentioned ISO standards cover both the functional and non-functional quality characteristics that are required for delivering reliable and high-quality software. Agile practices were covered, showing how quality can be maintained even in fast-paced, iterative development cycles. It also observes how Agile teams can use these standards to guide their work, making sure that the software meets user requirements while also performing well and adapting to change. Overall, this research provides an overview of how Agile teams can balance technical quality with flexible, collaborative ways of working to achieve better, quality-focused software outcomes.

Agile Software Development

Agile Software Development is a methodology where a software product is developed through iterations, while delivery is done incrementally. Considering the delivery itself, it is frequent, with the ability to adapt to customers' requirements changes [1]. Since software development utilizes a creative approach by using self-organized, cross-functional teams, overall collaboration is very important. Project managers and developers claim that agile software development means SCRUM or XP. Thus, by refining the question of what agile software development is (by adding more factors to the question), the answer becomes blurry. Sometimes, the definition of agile software development contradicts itself, regarding the terminology and existing concepts. However, pure agile or traditional software development happens very rarely, and most practices are only close to being agile [2]. Since agile methodology became widespread in organizations that mostly relied on traditional or structured development methods, such as the waterfall approach, in order to increase the benefits of different methods, organizations started a hybrid approach that combines agile, waterfall, and structured software development methodologies [3]. Therefore, due to the nature of agile methodology itself, there are obstacles for adopting quality assurance mechanisms to some level [1].

International Organization for Standardization

ISO/IEC 25010:2023 Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - Product Quality Model considers the information and communication technology products, as well as software products for organizational and personal activities, and is composed of nine characteristics (divided into subsets) that relate to quality properties of the products. These characteristics are used to describe and evaluate the quality of a software product from multiple perspectives. Main characteristics are functional suitability (how well the product provides the required functions), performance efficiency (how the product performs in relation to the resources it uses), compatibility (how the product can work with other systems), and interaction capability (the product's ability to support communication and cooperation with users and other systems). Other characteristics include reliability (the stability of the product during normal use), security (the protection of data and control of access), maintainability (how easily the product can be changed or updated), flexibility (the ability of the product to adapt to changes in the environment or requirements), and safety (the protection of people, property or the environment during system operation) [4].
ISO/IEC/IEEE 12207:2017 Systems and software engineering - Software Life Cycle Processes considers the processes, activities and tasks which take place during the acquisition (when software is obtained from an external source), supply (when software is provided to a customer), development (when software is created or modified), operation (when software is used in its intended environment), maintenance (when software is updated, corrected or improved), or disposal (when software is retired or removed from use) of software systems, products and services, while the full life cycle of software systems, products or services is considered. The standard defines a process model that includes technical and management aspects of software engineering and supports the organization and control of software work across all phases. It includes process definitions that guide the planning (defining the work to be done), implementation (carrying out development activities) and control (monitoring and managing progress) of software development and maintenance, as well as supporting processes (activities that assist development, such as documentation or testing) and organizational processes (activities related to managing resources, infrastructure and quality) that help manage quality, configuration, documentation and other important aspects across the software life cycle [5].
ISO/IEC 25019:2023 Systems and Software Engineering - Quality Requirements and Evaluation (SQuaRE) - Quality in Use Model provides a set of quality characteristics for specifying, measuring, evaluating and improving quality in use (the degree to which a software product meets user needs during actual operation), where the focus is on the user's experience with the product during real tasks in a real context. This standard expands the concept of software quality by introducing characteristics that describe how well users can achieve their goals (effectiveness), how efficient the software is during actual use (efficiency), how satisfied users are (satisfaction), how well the system avoids risks (freedom from risk), and how effectively it works in different usage contexts (context coverage). The model is intended to support the development of systems that meet both user expectations and quality goals from the perspective of people who interact with the software directly[6].

RELATED WORK

Software Quality in an Agile Environment

Management of quality requirements in the agile software development context is considered a challenging task [7]. Agile development methodology mechanisms of quality assurance must be established to ensure high-quality software production. Addressing quality issues is one of the important factors, and adopting the correct strategy is of great importance in the early stages of development. To ensure quality, two distinct techniques can be employed, which are constructive and analytic techniques. While constructive techniques consider prototyping, specification techniques (such as standards and processes), and elicitation techniques,

analytic techniques, on the other hand, focus on dynamics (validation and testing) and statics (verification and inspection) [1]. Since agile methodology considers working software products as a priority, development is divided into iterations (sprints), where each iteration (sprint) consists of planning, designing, development, testing, and feedback. The foundation of each iteration is a backlog, which is interpreted as a user requirement. However, user requirements can change, and the possibility of re-prioritizing features of development can be introduced. Regarding the possible changes, after iteration delivery, the customer performs a review and provides comments to determine the status of a product (iteration can be accepted, rejected, or requested for a change). Since iteration review presents a significant quality assurance mechanism in the agile development process, there are three important factors for each iteration to ensure higher quality, and they are the customer's availability (feedback on the current stage of development), communication medium (distance barrier, emails, telephone), and supporting tools (documentation) [1].
Regarding different aspects that have an impact on the quality of software, quality requirements themselves should be observed as well. They are considered non-functional requirements since they describe non-functional aspects of software such as maintainability, reliability, performance, and security. However, regardless of a software product being delivered, a system may not be regarded as good by the user if it does not meet certain performance, usability, or similar [8].
One more factor that impacts the overall software quality in an agile environment, according to [9], is agile software testing. By carefully planning and implementing agile testing practices such as continuous integration, test-driven development, and pair programming, software quality is improved by reducing defects and improving maintainability.
Dwelling deeper on the non-functional aspects of software quality, characteristics of functionality, reliability, usability, efficiency, and maintainability should be observed. Functionality represents a set of software attributes that have specific properties that provide functions that are required by the user. Reliability considers the ability to maintain specific levels of performance under specific conditions for a certain amount of time. Usability is a term used for software attributes that represent a measure of the effort needed for users to learn to use the product. Efficiency is a set of software attributes for the ability of the software product to provide a relationship between the level of performance of the said software, and the amount of resources that are used under stated conditions. Finally, maintainability represents a set of software attributes that are required to avoid various unexpected effects from specified modifications [10].

Software Quality Monitoring in an Agile Environment

The process of software engineering is useful for organizing development activities within the software development life cycle (SDLC). Common SDLC begins by analyzing the project's requirements and goes through the system's design, implementation, testing, deployment, and maintenance. The main goal of the agile methodology focuses on the ability to change, by minimizing cost and unnecessary rework, which is mostly found in traditional software engineering practices [11].
Agile methods are expected to deal with many unstable requirements by utilizing various techniques. Most notable techniques were simple planning, short iteration, early release, and frequent customer feedback. By performing those techniques, agile methods are able to deliver product releases in a much shorter period of time, when compared to the waterfall approach. However, there are several quality techniques that can be imposed to increase overall quality assurance, and they are: on-site customer, refactoring, acceptance testing, and early customer feedback. Observing the first quality technique, On-Site customer practice is something commonly found in agile methodology. By supporting the development team through the whole development process, customers can refine and correct requirements. Second quality technique - refactoring is considered a technique of restructuring an existing body of code by altering its internal structure without changing its external behavior. Each refactoring action should be small and should be done in sequence to prevent possible errors. This makes an

existing system resilient to getting seriously broken during the reconstruction. Third quality technique - acceptance testing, is a type of testing that is performed after all unit test cases have passed. Contrary to waterfall acceptance testing, agile acceptance testing occurs much earlier and more frequently and is done many times during iterations. Finally, the quality technique of early customer feedback is considered the most valuable characteristic of agile methods. By having short releases and quick movement to a development phase, the team is able to get customer feedback as soon as possible, which may provide valuable information for the development team [12].

Metrics for Software Quality Monitoring in an Agile Environment

Quality metrics are an important part of software engineering, as they transform both the customer expectation and operational performance into certain metrics that can be measured, evaluated, and compared. Metrics may be influenced by different factors such as functionality, usability, reliability, and portability [13]. Regarding the metrics of quality assessment, the Q-Rapids Tool is a result of a project whose main goal is to provide continuous quality assessment to support decision-makers in the management of quality in agile software development. Different metrics can be performed in correlation with the Q-Rapids quality model. The quality model itself consists of Quality Metrics, which are low-level indicators measuring a specific characteristic that is assessed from raw data stored in a data source. Later, Quality Metrics are aggregated into Quality Factors, which define significant concepts in relation to quality. After that, Quality Factors are combined to compute higher-level model elements, which are Strategic Indicators. Considering the quality assessment, the Q-Rapids Tool is used to gather data from heterogene, external data sources. After that, data is aggregated into Strategic Indicators following the Q-Rapids Quality Model. Predictions of the quality assessment are used to assist decision-makers in their decisions, considering product evolution. By using prediction techniques such as ARIMA, ETS, and Neural Networks, the evolution of strategic indicators is assembled. This functionality provides the ability to react to potential issues related to concrete aspects of the product, as well as quality issues that could emerge. However, if a prediction points out possible quality issues, what-if analysis can be used to evaluate improvement options and their impact on strategic indicators. Regarding that, quality assessment could be below some user-defined thresholds for one or more quality model elements, and due to Quality Requirement Semi-automatic Generation, a quality alert may be generated [7].

Psychological Factors

Psychological safety in an agile environment can enhance software quality by utilizing constructive and supportive environments. That kind of environment emphasizes learning from mistakes and encourages quality-related error reports, where mistakes can be turned into opportunities and continuous learning. Knowledge sharing and skill expertise are aimed at enhancing software quality, especially when team members feel more comfortable and contribute to the collective knowledge pool. Regarding the safety and psychological factor, human needs should be met, but it requires companies, management, agile teams, and their individual members to uphold and propagate norms that foster a sense of safety [14].
Regarding the self-organizing teams, handling the complexity and pressure of deadlines during the project development phase can be improved in a self-organized environment. Decisions may be taken on behalf of a self-organized team, and they could adapt accordingly to the situation. Contrary to command-and-control teams, self-organizing teams may have improved chances for personal development instead of solely focusing on completion of the project. By having a mature enough organization, productivity may increase, and dynamic changes may be more welcome even in the late implementation phase. Team size could impact the overall time required for certain actions as well, where fewer but more competent, experienced developers could deliver desired outputs in less time, with better quality [15].

RESULTS AND DISCUSSION

In this paper, the primary contribution was to review existing literature related to software quality and software quality monitoring in an agile environment, with a focus on functional and non-functional quality aspects and their impact on software quality while taking ISO standards into account.

CONCLUSION

In conclusion, Agile software development plays an important role in improving overall software quality by addressing both functional and non-functional aspects through the development process. Functional quality focuses on whether the software meets intended requirements, such as delivering the right features and behavior of software, while non-functional quality considers how well software performs under certain conditions, such as speed, reliability, security, and ease of use. The iterative approach of agile software development allows teams to continuously improve both types of quality through continuous testing and feedback. Self-organization of teams may benefit from psychological factors like autonomy, shared responsibility, and motivation while encouraging better collaboration, faster and more accurate decision-making, as well as more creative problem-solving. Together, both technical and human factors enable agile teams to adapt to change effectively and deliver high-quality software that meets both user needs and system demands.

REFERENCES

[1] R. Shuib and S. Hassan, "Towards adopting software quality assurance in agile development methodology," Turkish Journal of Computer and Mathematics Education, vol. 12, no. 3, pp. 2152–2157, 2021.
[2] M. Kuhrmann et al., "What makes agile software development agile?," IEEE transactions on software engineering, vol. 48, no. 9, pp. 3523–3539, 2021.
[3] A. Mishra and Y. I. Alzoubi, "Structured software development versus agile software development: a comparative analysis," Int J Syst Assur Eng Manag, vol. 14, no. 4, pp. 1504–1522, Aug. 2023, doi: 10.1007/s13198-023-01958-5.
[4] "ISO/IEC 25010:2023(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model." Accessed: Sept. 04, 2025. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en
[5] "ISO/IEC/IEEE 12207:2017(en), Systems and software engineering — Software life cycle processes." Accessed: Sept. 03, 2025. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:12207:ed-1:v1:en
[6] "ISO/IEC 25019:2023 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model." Accessed: Sept. 04, 2025. [Online]. Available: https://www.iso.org/standard/78177.html
[7] L. López, A. Bagnato, A. Ahberve, and X. Franch, "QFL: data-driven feedback loop to manage quality in agile development," in 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), IEEE, 2021, pp. 58–66. Accessed: Sept. 03, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9402163/
[8] P. Karhapää et al., "Strategies to manage quality requirements in agile software development: a multiple case study," Empir Software Eng, vol. 26, no. 2, p. 28, Mar. 2021, doi: 10.1007/s10664-020-09903-x
[9] M. I. Khan et al., "Analysis of Agile Software Testing, It's Impact on Software Quality and Cost," Int. J. Comput. Intell. Control Copyrights@ Muk Publ, vol. 14, no. 1, pp. 974–8571, 2022

[10] B. Dugalic and A. Mishev, "ISO Software Quality Standards and Certification.," in BCI (Local), 2012, pp. 113–116. Accessed: Sept. 03, 2025. [Online]. Available: https://ceur-ws.org/Vol-920/p113-dugalic.pdf
[11] A. Thool and C. Brown, "Securing Agile: Assessing the Impact of Security Activities on Agile Development," in Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, Salerno Italy: ACM, June 2024, pp. 668–678. doi: 10.1145/3661167.3661280.
[12] M. Huo, J. Verner, L. Zhu, and M. A. Babar, "Software quality and agile methods," in Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004., IEEE, 2004, pp. 520–525. Accessed: Sept. 06, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/1342889/
[13] K. Chakravarty and J. Singh, "A Study of Quality Metrics in Agile Software Development," in Machine Learning and Information Processing, vol. 1311, D. Swain, P. K. Pattnaik, and T. Athawale, Eds., in Advances in Intelligent Systems and Computing, vol. 1311. , Singapore: Springer Singapore, 2021, pp. 255–266. doi: 10.1007/978-981-33-4859-2_26.
[14] A. Alami, M. Zahedi, and O. Krancher, "The role of psychological safety in promoting software quality in agile teams," Empir Software Eng, vol. 29, no. 5, p. 119, Sept. 2024, doi: 10.1007/s10664-024-10512-1.
[15] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S. Z. Sarwar, "Agile software development: Impact on productivity and quality," in 2010 IEEE International Conference on Management of Innovation & Technology, IEEE, 2010, pp. 287–291. Accessed: Sept. 06, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5492703/