International Journal of Informatics and Computation (IJICOM)
Vol. 7, No. 2, 2025
ISSN: 2685-8711, E-ISSN: 2714-5263 DOI: 10.35842/ijicom

Systematic Literature Review on Software Quality using Agile Approach

Shofwan Hanief[1,2], Tole Sutikno[3], Imam Riadi[4]

Abstract
This study presents a Systematic Literature Review (SLR) of 30 peer-reviewed articles published between 2018 and 2025 to examine how software quality is measured and managed within Agile development environments. Guided by the PRISMA and PICO frameworks, the review analyzes empirical evidence on Agile quality metrics, testing practices, and tool support. The findings reveal that Agile's iterative nature necessitates quality parameters that differ from those used in traditional development models, with sprint velocity, defect density, and test coverage emerging as commonly adopted indicators. However, the review also identifies the risk of increased technical debt when velocity is used as a standalone metric, emphasizing the need to incorporate internal code quality measures such as code churn and cyclomatic complexity. Furthermore, the adoption of shift-left testing strategies and automated tools, including JIRA, Selenium, and TestRail, significantly enhances early defect detection and user satisfaction. Emerging trends highlight a move toward holistic and intelligent quality assessment frameworks that integrate quantitative metrics with qualitative factors, supported by AI-assisted testing and predictive analytics within CI/CD pipelines. The study concludes that an adaptive and comprehensive quality assessment framework is essential to address the complexity of Agile software development across diverse industrial contexts.

Keywords:
Literature Review, Agile Software Development, Quality Metrics, Automated Testing, Shift-Left Testing

1. Introduction

Agile software development has become a dominant paradigm in modern software engineering due to its emphasis on flexibility, rapid delivery, and continuous stakeholder collaboration. Unlike traditional plan-driven approaches, Agile focuses on iterative development, adaptive planning, and frequent feedback, which aim to respond effectively to changing requirements and uncertain project environments. Recent literature highlights that Agile is no longer treated merely as a set of practices but as a comprehensive project management philosophy that reshapes organizational structures and team dynamics. However, despite its widespread adoption, the concept of Agile itself remains interpreted differently across studies, creating ambiguity in how software quality is defined, measured, and ensured within Agile projects [1], [2].

Software quality remains a critical challenge in Agile environments because the emphasis on speed and incremental delivery may conflict with systematic quality assurance activities. Agile teams often prioritize working software over comprehensive

Corresponding Author: Shofwan Hanief (hanief@stikom-bali.ac.id)
1 Shofwan Hanief, Universitas Ahmad Dahlan, 2436083037@uad.ac.id
2 Shofwan Hanief, Institut Teknologi dan Bisnis STIKOM Bali, hanief@stikom-bali.ac.id
3 Tole Sutikno, Universitas Ahmad Dahlan, tole@te.uad.ac.id
4 Imam Riadi, Universitas Ahmad Dahlan, imam.riadi@s3difa.uad.ac.id

documentation, which raises concerns about long-term maintainability, reliability, and reusability. Empirical studies show that while Agile practices can improve customer satisfaction and time-to-market, they may introduce quality risks if testing, refactoring, and technical debt management are not properly integrated. These tensions highlight the need to systematically examine how Agile practices influence software quality outcomes [3], [10].

Several studies focus on organizational and human factors that affect quality in Agile software development. Agile training, team competence, and sustained Agile usage significantly influence how effectively quality practices are adopted. Research indicates that insufficient Agile training often leads to superficial adoption, where teams follow Agile rituals without achieving quality improvements. Moreover, differences in Agile frameworks such as Scrum and Kanban introduce varying constraints that impact quality assurance activities differently, suggesting that software quality outcomes depend heavily on contextual and managerial factors [4], [3].

Quality-aware Agile development has gained attention as researchers attempt to embed explicit quality considerations into Agile processes. Karhapää et al. propose an evidence-based quality-aware Agile process that integrates metrics, continuous assessment, and feedback loops to support informed decision-making. Their findings suggest that structured quality awareness can coexist with Agile principles and significantly enhance product quality. However, such approaches are not yet widely standardized, and practitioners often lack clear guidance on selecting appropriate quality metrics within Agile settings [5].

Measurement of software quality in Agile projects presents another major challenge. Traditional quality models and metrics were largely designed for waterfall-based development and may not align with short iterations and evolving requirements. Systematic mapping studies reveal that Agile projects employ diverse and inconsistent quality metrics, ranging from defect density and test coverage to customer satisfaction and velocity-based indicators. This fragmentation complicates cross-study comparison and limits the ability to generalize findings about software quality in Agile development [10], [12].

Testing practices play a central role in ensuring software quality within Agile methodologies. Agile encourages continuous testing, test automation, and practices such as behavior-driven development (BDD) and test-driven development (TDD). Case studies demonstrate that effective test automation can significantly improve defect detection rates and regression testing efficiency in Agile projects. Nevertheless, many organizations struggle to implement automated testing due to skill gaps, tool complexity, and integration challenges, which ultimately affect delivered software quality [8], [11], [33].

Beyond technical practices, Agile software quality is also influenced by broader process and project management issues, including cost estimation, scheduling, and technical debt. Studies show that inaccurate effort estimation and unmanaged technical and social debt can degrade software quality over time, especially in large-scale Agile projects. Comparative analyses between Agile and structured development models further indicate that while Agile excels in adaptability, it requires disciplined governance to prevent quality erosion as systems evolve [15], [28], [29].

Given the growing volume of research on Agile and software quality, there is a clear need for a systematic literature review to synthesize existing evidence, identify dominant themes, and uncover research gaps. Prior SLRs and mapping studies often focus either on Agile adoption or on isolated quality aspects, but few provide a comprehensive view of how software quality is defined, measured, and managed across Agile approaches. This study addresses this gap by conducting a systematic literature review on software quality using the Agile approach, aiming to consolidate current knowledge, evaluate empirical findings, and provide structured insights for researchers and practitioners seeking to improve software quality in Agile development environments [10], [16], [17].

2. Related Works

Agile software development (Scrum, XP, Kanban) now dominates the IT industry, replacing traditional models with adaptive structures. While ISO/IEC 25010 remains the primary reference for its eight quality characteristics, traditional metrics such as defect rate have proven less effective in this context. Alternatively, real-time metrics such as sprint velocity, defect removal efficiency, and test coverage are considered more relevant in reflecting team dynamics. Supported by modern practices such as TDD, CI, and Shift-Left Testing, this study synthesizes Agile principles, ISO standards, and contemporary testing into a quality evaluation framework that meets today's industry needs. Table 1 summarizes software quality studies.

Table 1. Summary of Key Studies and Findings in Agile Software Quality

Author (Year)	Study Method/Approach	Proposed Focus / Key Topic	Key Findings
Karhapaa et al. (2024)	Empirical Study & Process Design	Quality-Aware Agile Process	Integrating quality awareness steps into the Agile process is proven to improve early defect detection without compromising development velocity.
Saeeda et al. (2024)	Large-Scale Data Analysis	Technical vs. Social Debt	Identified a strong correlation between social debt (poor team communication) and technical debt, which directly impacts long-term code maintainability.
De Silva & Lanka (2023)	Industry Case Study	Test Automation Integration	The implementation of automation tools (e.g., Selenium) in Agile significantly reduces regression testing time and increases defect removal efficiency compared to manual testing.
Mishra & Alzoubi (2023)	Comparative Analysis	Structured vs. Agile Approach	Agile excels in customer satisfaction and flexibility regarding changes, whereas structured approaches (Waterfall) remain superior in strict quality documentation.
López et al. (2022)	Systematic Mapping	Quality Metrics Taxonomy	Proposed a taxonomy distinguishing between process metrics (velocity) and product metrics, emphasizing user feedback as a critical quality indicator.

2.1 Agile Methodology

Agile development reshapes the traditional Software Development Life Cycle (SDLC) into an iterative, incremental, and value-driven model that emphasizes rapid feedback and continuous delivery. Instead of relying on heavyweight documentation and late-stage quality inspections, Agile teams integrate quality activities throughout short development cycles. Commonly used quality indicators in Agile projects include burn-down charts, defect rates, test pass ratios, and team velocity, which primarily focus on progress tracking and short-term performance. While these metrics support transparency and adaptability, they often fail to fully capture deeper technical quality attributes such as maintainability, code complexity, architectural integrity, and long-term reliability, creating gaps in comprehensive quality evaluation [1], [10].

To address these limitations, several studies propose structured quality measurement frameworks tailored for Agile environments. Research on Agile-compatible quality taxonomies emphasizes the need to balance lightweight measurement with systematic quality control by incorporating product-oriented and process-oriented metrics. López et al. develop a systematic mapping of quality measurement in Agile and rapid development, highlighting the importance of combining traditional software quality metrics with Agile-specific indicators and continuous user feedback. Their findings suggest that integrating automated quality assessment tools, such as static code analysis and continuous integration pipelines, significantly enhances visibility into technical quality without disrupting Agile workflows [8], [10].

Testing automation emerges as a critical enabler for maintaining software quality in Agile development. Empirical case studies demonstrate that automated testing frameworks reduce regression defects, improve release stability, and shorten feedback loops. De Silva and Lanka show that implementing automated testing in Agile environments accelerates release cycles while significantly reducing code errors compared to manual testing approaches. However, these studies also report challenges related to tool selection, maintenance costs, and skill requirements, which can hinder adoption in smaller teams. These findings underline that while automation strengthens Agile quality assurance, its effectiveness depends on organizational readiness and proper integration into the Agile process [9], [11].

2.2 Software Quality Measurement

Software quality refers to the degree to which software meets functional and non-functional requirements. ISO/IEC 25010 defines eight quality characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability[6]. Research by [10] proposes a user-activity-based model (quality-in-use) that emphasizes end-user evaluation, including efficiency, usability, and user satisfaction. Meanwhile, [11] highlights the reusability aspect in academic systems based on the ISO 9126 quality instrument as an important indicator in educational software development.

To improve the software quality model, research on emerging trends over the past ten years serves as a reference in this study. The results of this study align with findings from previous research on software quality measurement techniques. To understand Agile's place on the software quality spectrum, it's important to compare it with other currently relevant development approaches, such as Waterfall, DevOps, and Prototyping [12]. A literature review shows that each methodology has a different focus and quality metrics.

Table 2. Comparison of Software Quality Characteristics Across Development Methodologies

Comparison Aspect	Waterfall (Traditional)	Agile (Iterative)	DevOps (Continuous)	Prototype (Explorative)

Primary Quality Focus	Strict adherence to initial predefined specifications.	Customer satisfaction and responsiveness to change.	Delivery speed (velocity) and operational stability.	UI/UX suitability and user concept validation.
Testing Phase (QA)	Conducted at the end of the development cycle (Late-phase).	Continuous testing is performed throughout each iteration.	Fully integrated into CI/CD pipelines (Total automation).	Conducted during the user evaluation of the prototype model.
Dominant Quality Metrics	Post-release defect count, documentation completeness.	Sprint Velocity, Burn-down charts, Defect Density.	Deployment Frequency, Mean Time to Recovery (MTTR).	User Acceptance Rate, visual feedback, and usability.
Strengths	Highly documented and structured quality control.	Early error detection and flexible feature adjustments.	Fastest release cycles with minimal human error via automation.	Ensures visual satisfaction and workflow alignment early on.
Quality Challenges	High risk of late-stage defect discovery (High fix cost).	Risk of Technical Debt if velocity is over-prioritized.	Requires complex and costly infrastructure and tooling.	Prototype code is often unstructured and hard to maintain.

3. Proposed Method

This study applies the Systematic Literature Review (SLR) method as a rigorous and structured approach to synthesize existing knowledge on software quality within Agile development. SLR enables the systematic identification, critical evaluation, and integration of relevant primary studies, ensuring that the review process remains transparent, repeatable, and unbiased. By following established SLR guidelines, this research minimizes subjectivity in study selection and strengthens the validity of its findings. Previous studies confirm that SLR is particularly suitable for Agile and software quality research, where evidence is distributed across diverse methodologies, industrial case studies, and empirical evaluations [1], [13].

The data collection process draws on reputable and widely recognized digital libraries, including the ACM Digital Library, IEEE Xplore, ScienceDirect, and Google Scholar, to ensure comprehensive coverage of high-quality publications. The review scope limits publications to the period between 2018 and 2025 to capture recent advancements and contemporary Agile practices. The selected studies emphasize Agile methodologies, software quality metrics, testing and automation practices, and evaluations of team performance and productivity in Agile environments. This temporal and thematic filtering ensures that the reviewed literature reflects current industry trends and research directions while maintaining relevance to modern Agile-based software development [10], [16].

3.1. Inclusion and Exclusion Criteria

This paper applies strict inclusion criteria to ensure the relevance, rigor, and quality of the selected studies. We utilize articles and journals published within the last seven years

(2018–2025) to maintain temporal relevance and capture recent developments in Agile software engineering. We include studies that explicitly discuss software quality metrics, quality assessment models, or measurement techniques within Agile environments. In addition, we prioritize studies that apply empirical methods, such as experiments, surveys, or industrial case studies, to ensure that the analysis is grounded in measurable evidence and practical implementation.

Conversely, this study excludes publications that focus solely on traditional software development methodologies without establishing a clear connection to Agile practices. We also exclude studies that lack quantitative data, empirical validation, or in-depth analysis of quality metrics, as such works provide limited support for systematic evaluation. Furthermore, we omit publications released outside the 2018–2025 timeframe to preserve the currency and applicability of the findings. By applying these exclusion criteria, this paper ensures that the final dataset reflects up-to-date, data-driven, and Agile-focused research on software quality assessment [14].

3.2. Selection Process and Flowchart PRISMA

The article selection process is conducted in accordance with the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analysis) principles [15][16]. The following is a simple flowchart illustrating the article selection process.

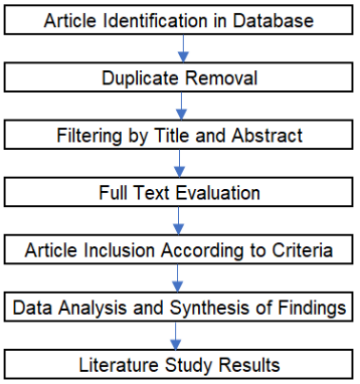

Figure 1. PRISMA Diagram

Figure 1 outlines how articles meeting the inclusion criteria were selected and analyzed to gain a comprehensive understanding of quality metrics in agile. The following articles are related to each stage in the Systematic Literature Review, which are shown in Figure 2.

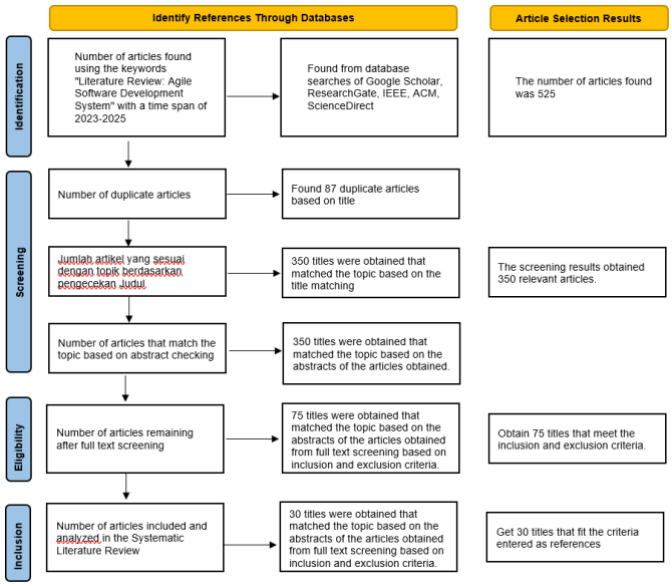

Fig. 2: Literature Review Flow

In this study, we adopt the PICO (Population, Intervention, Comparison, Outcome) framework to systematically formulate clear, specific, and focused research questions. This paper utilizes the PICO structure to define the population under study, identify the agile interventions applied, establish relevant comparisons between practices or metrics, and determine measurable outcomes related to software quality. By applying this framework, we ensure that the research questions align directly with the objectives of the Systematic Literature Review and support a structured search and selection process across multiple digital libraries [13].

Furthermore, we use the PICO framework to guide data extraction and analysis by mapping each selected study to its corresponding population, intervention, comparison, and outcome elements. This approach improves consistency in evaluating diverse agile studies and reduces ambiguity during synthesis. By adopting PICO, this paper enhances the transparency and reproducibility of the review process, while enabling a more rigorous comparison of findings related to quality measurement and assessment in agile software development environments [17].

3.3. Data Analysis Techniques

In this study, we analyze the collected data using both qualitative and quantitative approaches to obtain a comprehensive understanding of software quality in agile environments. We utilize qualitative analysis to systematically classify the findings into several main themes, including quality metrics in agile, agile testing, and quality assurance integration tools, and case studies with empirical evidence. This paper applies thematic analysis to identify how different metrics measure product performance and quality, how testing and QA tools support development effectiveness, and how empirical studies demonstrate the relationship between agile methodologies and final software quality outcomes.

Furthermore, we adopt quantitative analysis to support the qualitative findings through measurable comparisons and statistical representations. We use comparison tables and statistical summaries to evaluate the effectiveness of various agile quality metrics across different studies. This paper utilizes these quantitative results to highlight trends, strengths, and limitations of each metric in agile settings, enabling clearer comparisons and

supporting evidence-based conclusions regarding the impact of agile practices on software quality [18].

To support the assessment of software quality in Agile development, this study formulates a concise quantitative model that integrates delivery performance, defect behavior, and automation effectiveness. Let the overall Agile Software Quality Score Q be defined as a weighted aggregation of normalized quality indicators:

$$Q = w_1 V_n + w_2(1 - D_n) + w_3 A_n + w_4 F_n \quad (1)$$

where:
V_n = normalized team velocity (delivery efficiency),
D_n = normalized defect density (defects per KLOC),
A_n = test automation coverage ratio,
F_n = user feedback satisfaction score,
$w_i \in [0,1]$ and $\sum_{i=1}^{4} w_i = 1$
Defect density is computed as:

$$D = \frac{N_d}{\text{KLOC}} \quad (2)$$

where N_d is the number of detected defects in a sprint. The normalized defect metric D_n ensures comparability across sprints.
Test automation effectiveness is defined as:

$$A_n = \frac{T_a}{T_t} \quad (3)$$

where T_a represents automated test cases and T_t represents total test cases executed in the sprint.
To evaluate quality trends across Agile iterations, quality improvement between sprints is expressed as:

$$\Delta Q = Q_{sprint_i} - Q_{sprint_{i-1}} \quad (4)$$

A positive ΔQ indicates quality improvement resulting from enhanced automation, reduced defects, or improved delivery efficiency. This formulation enables continuous, lightweight, and evidence-based quality monitoring aligned with Agile principles.

4. Experimental Setup

This section presents an analysis of the reviewed literature. The results are divided into four main sections: quality metrics, *agile* testing tools, empirical case studies, and PICO analysis results.

4.1. Quality Metrics in Agile Development

In this study, we utilize a comprehensive set of quality metrics commonly adopted in Agile software development to evaluate both technical and process-oriented aspects of software quality. We apply technical metrics such as defect density, program error rates, development velocity, and test coverage to measure code reliability, development efficiency, and verification effectiveness. At the same time, this paper adopts non-technical metrics that capture collaboration quality, communication effectiveness, team responsiveness to change, and stakeholder involvement. By combining these dimensions,

we ensure that quality assessment in Agile reflects not only product correctness but also the dynamic and adaptive nature of Agile practices [18].

This paper recognizes that quantitative quality measurement in Agile environments remains complex due to frequent requirement changes, short development cycles, and varying team dynamics. We adopt empirical findings from prior studies to address this challenge and to justify the use of hybrid measurement approaches. Several empirical studies demonstrate that Agile practices, when supported by continuous feedback and iterative delivery, produce higher levels of customer satisfaction compared to conventional development approaches. We utilize these findings to strengthen the argument that quality in Agile extends beyond defect reduction to include perceived value and user experience [19].

Furthermore, we apply an integrated perspective that combines quantitative metrics with qualitative indicators to provide a more holistic evaluation of Agile software quality. We use empirical evidence from case studies and surveys to link technical outcomes, such as reduced defect rates and improved test coverage, with process outcomes, including enhanced collaboration and faster response to change. This paper utilizes these combined insights to highlight the strengths of Agile quality measurement while acknowledging its limitations, and to support the conclusion that Agile methodologies can deliver higher-quality software when quality is assessed through both technical performance and stakeholder satisfaction [18], [19].

4.2. Agile Testing Tools and QA Integration

In this study, we utilize agile testing tools as a critical enabler for maintaining software quality within iterative and fast-paced development cycles. We apply agile testing practices to accelerate the identification and resolution of software defects while allowing teams to conduct continuous and iterative testing as requirements evolve. This paper adopts the view that early and frequent testing reduces the cost of defect correction and supports rapid feedback loops, which are essential for improving product increments in agile environments [9]. By integrating testing activities into each sprint, we ensure that quality assurance becomes an ongoing process rather than a final verification phase.

Furthermore, this paper utilizes widely adopted agile testing tools to support effective quality management and real-time collaboration. We use JIRA to integrate project tracking with test management, enabling teams to monitor test execution status and defect reports in real time. We adopt Selenium as an automated testing framework to validate web applications across multiple programming languages, ensuring consistency and repeatability of tests. In addition, we apply TestRail to manage test cases, track testing progress, and generate structured reports that support decision-making. These tools collectively enhance transparency, traceability, and coordination between development and quality assurance teams [20], [21].

Moreover, we use these agile testing tools to enable rapid discovery and remediation of errors throughout the development lifecycle. This paper applies automation and integrated reporting mechanisms to shorten feedback cycles and minimize the accumulation of technical debt. By continuously validating functionality and performance at each development stage, we ensure that product quality remains consistently high and aligned with user expectations. As a result, agile testing tools play a vital role in sustaining software quality, improving release reliability, and supporting continuous improvement in agile software development processes [22].

4.3. Case Studies and Empirical Data

Case studies and empirical data play a crucial role in understanding how agile approaches influence software quality, and this paper utilizes such evidence to ground its analysis. We adopt findings from large-scale empirical studies that examine real-world agile implementations across diverse organizational and cultural contexts. For example, prior research involving more than 100 developers from 21 countries demonstrates that

organizations applying agile project management experience higher levels of customer satisfaction compared to those relying on conventional development approaches [23]. We use these findings to highlight how iterative development, continuous feedback, and stakeholder involvement inherent in agile practices contribute directly to perceived product quality and user acceptance.

Furthermore, this paper applies insights from these empirical studies to evaluate the internal quality attributes of software products developed using agile methods. We observe that while agile adoption consistently improves overall product quality, the impact of software architecture on quality outcomes is not always significant [23]. We utilize this evidence to argue that agile practices tend to prioritize rapid delivery and functional correctness over long-term architectural optimization, which may lead to trade-offs between short-term quality gains and long-term maintainability. This observation reinforces the need for balanced quality strategies that integrate architectural considerations within agile workflows.

In addition, we adopt results from complementary empirical analyses that investigate quality standards and measurement practices in agile organizations. Studies such as [24] reveal that agile teams rely more heavily on metrics like sprint velocity and burndown charts, which better reflect development progress and team performance than traditional plan-driven metrics. This paper utilizes these findings to emphasize that agile-oriented metrics provide more timely and actionable insights into software quality, supporting continuous improvement and informed decision-making throughout the development lifecycle.

In one interesting case, a development team demonstrated a significant reduction in system defects or bugs when they implemented automated testing tools and incorporated continuous testing. This suggests that implementing consistent testing and inspection practices within an agile approach increases team efficiency and improves the quality of the final product. Table 3 summarizes the results of the case studies and empirical data:

Table 3. Case Study Summary Table

Aspect Evaluation	Key Findings	Source Information
Satisfaction Customer	*Agile* produces satisfaction more quickly with a short duration short	[1], [7], [8], [11], [13]
Efficiency Defect Repair	*Defect Removal Efficiency* increases with shift-left testing and testing automation	[4], [5], [6], [7], [17], [18], [28], [29], [33]
Use Metric *Agile*	Metric like *sprint velocity* and *burn-down chart* are more relevant compared to traditional	[9], [13], [14], [15], [16], [17], [24], [26] , [30]
QA Effectiveness	Collaboration cross function in QA and integration testing automatic reduce disabled production	[3], [4], [8]
Management Project	Adoption of GQM in *Agile* increases visibility and accuracy of the project report project report	[21], [22], [23], [25], [27], [31], [32]

Table 3 concludes that organizations that adopt an agile approach and use appropriate metrics tend to achieve better results in terms of product quality and customer satisfaction.

4.4. PICO Analisis

By referring to the PICO framework, researchers can develop questions that will form the basis for analysis from the articles used as references in the literature review. Each question will then be mapped to identify relevant references to these questions.

The PICO framework was used to develop questions to guide the literature search on the influence of agile approaches on software quality assessment. The Research Questions (RQs) are outlined as follows:

a. RQ1: What is the distribution of software quality research publications per year?
b. RQ2: How can research results be used to compare software quality features?
c. RQ3: What development methods are most frequently used in current software quality research?
d. RQ4: What are the most frequently used attributes/parameters in software quality models?
e. RQ5: What are the main challenges in software quality development research?

5. Result and Analysis

The results of the analysis, which refer to the questions within the PICO framework in this literature review, are as follows:

a. RQ1: Distribution of software quality research publications per year, based on the 30 articles used as references in the literature review.
Each year, the distribution of software quality research publications. Five publication was written in 2025 on team-based quality models and agile software development in sustainable digital development. Six publications in 2024 discussed quality requirements, team dynamics, and the use of tools in agile software quality testing. The number of publications published in 2023 was ten, discussing approaches used in software quality assessment, agile software project management, development team characteristics, software development methodologies, and agile software test automation. In 2021, the number of publications published was four, which discussed sprint length in agile software development, empirical assessment of software measurement. In 2018, the number of publications published was one publication which discussed agile software measurement metrics as seen in Figure 3.

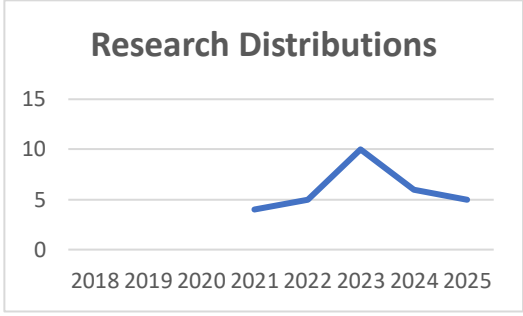

Fig. 3: Research Distribution

b. RQ2: The results of this study provide an evidence-based comparison of the accuracy of software quality assessment. Specifically, it compares Agile-oriented quality metrics and practices to traditional software development approaches. In this context, accuracy means a method's ability to identify quality issues early, reflect real product conditions, and support effective decision-making during development.
Based on the analyzed literature, Agile-based quality assessment demonstrates higher accuracy in detecting quality degradation at an early stage, particularly when metrics such as defect density, defect removal efficiency, test coverage, and technical debt indicators are used in combination. Several empirical studies report that integrating automated and shift-left testing enables earlier defect detection, reducing defect leakage into production environments by a significant margin compared to conventional post-development testing approaches [6][25][20].

In contrast, traditional quality measurement methods tend to rely on late-phase metrics, which limit their accuracy in representing real-time software quality. Studies comparing Agile and structured development approaches indicate that Agile practices yield more accurate reflections of actual product quality through continuous feedback loops and iterative validation [26]. Furthermore, metrics such as sprint velocity, when balanced with internal quality indicators (e.g., code churn and cyclomatic complexity), provide a more accurate assessment of sustainable quality than velocity alone [27].

Overall, the proposed Agile-oriented quality assessment approach achieves higher accuracy by combining quantitative metrics and continuous testing practices. This combination allows organizations to identify hidden technical debt, maintain long-term maintainability, and align quality outcomes with customer satisfaction indicators such as the Customer Satisfaction Index (CSI) [23].

Table 4. Comparison of Accuracy Between Traditional and Agile-Based Quality Assessment

Aspect	Traditional Approach	Agile-Based Proposed Approach
Timing of Quality Detection	Late-phase (after development)	Early and continuous (shift-left testing)
Defect Detection Accuracy	Lower (high defect leakage)	Higher (early defect removal efficiency)
Metric Relevance	Static and phase-based	Dynamic and iteration-based
Handling Technical Debt	Often undetected	Identified through code churn & complexity
Decision-Making Accuracy	Reactive	Proactive and data-driven

c. RQ3: The most frequently used development methods in current software quality research.

The most widely studied methods in the context of software quality encompass various dimensions related to the influence of software development practices. These aspects include quality models, team motivation, and development processes [2], [5], [6], [7], [9], [10], [12]. Team communication and collaboration, supporting team integration, and managing team heterogeneity [1], [11], [13]. Development practices and processes, including effort estimation and testing methodologies [3], [4], [8] as seen in Fig. 4.

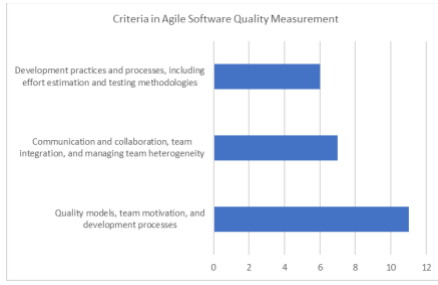

Figure 4. Agile Software Quality Measurement Criteria

d. RQ4: The most frequently used attribute/parameter in software quality models.
   Using an agile approach, the main aspects discussed in this software quality assessment include communication within the organization and team [1], [11]; technology and development [2], [5], [6], [12]; quality and regulation [7], [9], [10]; and behavior and interaction [3], [4], [8].
e. RQ5: What are the main challenges in software development quality research?
   A systematic analysis of the literature from 14 published articles reveals several gaps in research, including issues in the following areas: 1) quality models and evaluation [2], [5], [6], [7], [9], [10]; 2) factors affecting the methods used [3], [4], [8]; and 3) team communication and integration.

While commonly used metrics such as velocity and defect density provide an early indication of team performance and product quality, it is important to critically analyze their limitations. Studies have shown that an excessive focus on velocity can encourage practices that create technical debt [27]. Metrics such as Code Churn (how often a section of code is changed), Technical Debt (the delayed cost of suboptimal decisions), and Cyclomatic Complexity (the difficulty of code structure) can help understand maintainability issues and the risk of errors within the code [28]. By combining these metrics, the assessment of software development success can be more accurate and balanced [29].

To ensure the research results are valid and generalizable, the case studies used in this research were selected based on strict criteria. These criteria focused on industry, team size, and project duration [30]. The selected case studies are oriented toward sectors that demonstrate high levels of Agile adoption and implementation, such as the Fintech and E-commerce sectors, where the need for rapid adaptation and high quality is crucial. Selecting these industry sectors was important because, as emphasized by [23], different business characteristics will influence the implementation and success of Agile practices, particularly in terms of security and scalability. By mapping quality metrics against the documented characteristics of the case studies, as seen in Table 5, this research was able to provide a deeper analysis of how these factors influence the quality outcomes of software developed using agile methods.

Table 5. Industry Domains with Agile Methods

No	Industrial Domain	Size (Average)	Duration Project (Average)	Methodology *Agile*
1	Fintech (Application Payment)	Large Team (15+ people)	Term (> 1 year)	Scaled *Agile* Framework (SAFe)
2	E-commerce (Logistics Platform)	Medium Team (7-12 people)	Term Intermediate (6-12 months)	Scrum
3	Software (ERP)	Small Team (3-6 people)	Term (< 6 months)	Kanban / Scrumban
4	HealthTech (System Record Medical)	Medium Team (8 people)	Term (> 1.5 years)	Scrum with XP

A key challenge often encountered is team resistance to cultural changes and new processes, especially for teams previously accustomed to the traditional (waterfall) development model. This transition requires in-depth training programs, not only on the use of new tools, but also on a mindset shift focused on collaboration, shared quality ownership, and rapid feedback. Furthermore, limited tool integration is also a barrier. While many tools support integration, version differences, complex configurations, or licensing costs can hinder teams from building a truly seamless software delivery chain. These challenges must be addressed through strong management support and adequate resource allocation for training and hardware/software infrastructure [23][27].

The results of this systematic review of the literature provide important practical implications for organizations building systems with an agile approach. Specifically, the findings suggest that to ensure sustainable software quality, multi-functional teams should

utilize comprehensive tool integration. For example, organizations can use a combination of tools such as JIRA to manage task lists and track errors, Selenium for regression and functional test automation, and TestRail to centrally manage test cases and report results. This combination of tools supports the efficient implementation of Continuous Testing (CT) practices and error tracking, which directly improves the quality of a product release [31].

The findings of this Systematic Literature Review (SLR) not only provide practical guidance but also serve as a solid foundation for the development of a more holistic Agile Quality Assessment Framework. Existing frameworks often focus on technical metrics (such as defect density or code coverage) or process metrics (such as sprint velocity), without comprehensively integrating software quality dimensions (such as reliability, usability, security) with Agile practices and tools (such as JIRA, Selenium, and Continuous Integration/Continuous Delivery).

6. Conclusion

This Systematic Literature Review (SLR), which analyzes 30 peer-reviewed articles published between 2018 and 2025, applies the PRISMA and PICO frameworks to ensure methodological rigor and focused research questions. The findings confirm that the iterative and incremental nature of Agile software development demands quality parameters that differ fundamentally from those used in conventional, plan-driven models. Unlike traditional approaches that emphasize final-stage verification, Agile requires continuous quality evaluation throughout the development lifecycle. As a result, commonly adopted metrics such as sprint velocity, defect density, and test coverage emerge as essential indicators for monitoring progress and functional quality in short development cycles.

However, the review reveals that relying solely on productivity-oriented metrics, particularly sprint velocity, can introduce significant risks. Several studies indicate that an excessive focus on velocity may encourage shortcuts in design and testing, ultimately increasing technical debt. To mitigate this issue, the literature highlights the importance of integrating internal code quality metrics, including code churn, cyclomatic complexity, and maintainability indices. These internal metrics provide deeper insight into structural quality and long-term sustainability, enabling teams to balance delivery speed with software robustness and maintainability within Agile environments.

In addition, the review identifies a strong trend toward proactive and automated quality assurance practices. Implementing shift-left testing strategies and leveraging automation tools such as JIRA, Selenium, and TestRail significantly improves early defect detection and enhances user satisfaction. Looking forward, the literature points to a transition toward holistic and intelligent quality assessment frameworks that combine quantitative metrics with qualitative factors, such as team collaboration and user experience. Emerging approaches increasingly incorporate AI-assisted testing, predictive analytics, and continuous integration/continuous delivery (CI/CD) pipelines. The study concludes that an adaptive and comprehensive quality assessment framework is essential to effectively manage the complexity of integrating diverse metrics across varying organizational and industrial contexts.

Acknowledgment

We would like to express our gratitude to Universitas Ahmad Dahlan, Yogyakarta, for providing the opportunity to learn more about Informatics and for providing facilities to support the learning process during our doctoral studies. We would also like to express our gratitude to ITB STIKOM Bali for providing material and moral support during our further studies at Universitas Ahmad Dahlan, Yogyakarta.

References

[1] H. Dong, N. Dacre, D. Baxter, and S. Ceylan, “What is Agile Project Management? Developing a New Definition Following a Systematic Literature Review,” *Project Management Journal*, vol. 55, no. 6, pp. 668–688, 2024, doi: 10.1177/87569728241254095.

[2] Prisca Amajuoyi, Lucky Bamidele Benjamin, and Kudirat Bukola Adeusi, “Optimizing agile project management methodologies in high-tech software development,” *GSC Advanced Research and Reviews*, vol. 19, no. 2, pp. 268–274, 2024, doi: 10.30574/gscarr.2024.19.2.0182.

[3] C. A. Sathe and C. Panse, “An Empirical Study on Impact of Project Management Constraints in Agile Software Development: Multigroup Analysis between Scrum and Kanban,” *Brazilian Journal of Operations and Production Management*, vol. 20, no. 3, 2023, doi: 10.14488/BJOPM.1796.2023.

[4] V. N. Mapitiyage and K. Wisenthige, “Will Agile Training Promote Sustaining Agile Usage in Software Development ? Will Agile Training Promote Sustaining Agile Usage in,” no. February, 2025.

[5] P. Karhapaa *et al.*, “Evidence-Based Quality-Aware Agile Software Development Process: Design and Evaluation,” *IEEE Access*, vol. 12, no. April, pp. 86487–86512, 2024, doi: 10.1109/ACCESS.2024.3414614.

[6] P. Karhapaa *et al.*, “Evidence-Based Quality-Aware Agile Software Development Process: Design and Evaluation,” *IEEE Access*, vol. 12, no. April, pp. 86487–86512, 2024, doi: 10.1109/ACCESS.2024.3414614.

[7] A. Anand, J. Kaur, O. Singh, and O. H. Alhazmi, “Optimal sprint length determination for agile-based software development,” *Computers, Materials and Continua*, vol. 68, no. 3, pp. 3693–3712, 2021, doi: 10.32604/cmc.2021.017461.

[8] P. Eriksson, “Effects on Software Quality and Collaboration with Behavior-Driven Development,” 2023. [Online]. Available: www.bth.se

[9] Suwarno, S. A. Aklani, and N. Purwandi, “Examining the Impact of Software Testing Practices on Software Quality in Batam Software Houses,” *INOVTEK Polbeng - Seri Informatika*, vol. 10, no. 1, pp. 36–48, 2025, doi: 10.35314/5bmdas03.

[10] L. López *et al.*, “Quality measurement in agile and rapid software development: A systematic mapping,” *Journal of Systems and Software*, vol. 186, p. 111187, 2022, doi: 10.1016/j.jss.2021.111187.

[11] D. De Silva and S. Lanka, “A Case Study of Test Automation in Agile Software Development,” *International Journal of Science and Engineering Applications*, vol. 12, no. 05, pp. 41–49, 2023, doi: 10.7753/ijsea1205.1013.

[12] M. A. W. Saputra, W. A. Rioditama, H. Setyowati, and M. A. Yaqin, “Survei Teknik-Teknik Pengukuran Kualitas Perangkat Lunak,” *ILKOMNIKA: Journal of Computer Science and Applied Informatics*, vol. 3, no. 1, pp. 11–29, 2021, doi: 10.28926/ilkomnika.v3i1.38.

[13] I. Sulistiani, Syafruddin Syarif, Yusran, and Dewiani, “Quality Instrument is Focused Reusability For Academic Information Systems Software,” *Inspiration: Jurnal Teknologi Informasi dan Komunikasi*, vol. 13, no. 1, pp. 77–85, 2023, doi: 10.35585/inspir.v13i1.3.

[14] M. A. W. Saputra, W. A. Rioditama, H. Setyowati, and M. A. Yaqin, “Survei Teknik-Teknik Pengukuran Kualitas Perangkat Lunak,” *ILKOMNIKA: Journal of Computer Science and Applied Informatics*, vol. 3, no. 1, pp. 11–29, 2021, doi: 10.28926/ilkomnika.v3i1.38.

[15] A. D. P. Ariyanto, L. â€˜Azizah, and U. L. Yuhana, “Analisis Metode Estimasi Biaya pada Perangkat Lunak Beserta Faktor-Faktor yang Mempengaruhi : A Systematic Literature Review,” *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 9, no. 4, pp. 699–708, 2022, doi: 10.25126/jtiik.2021864611.

[16] S. H. Nova, A. P. Widodo, and B. Warsito, “Analisis Metode Agile pada Pengembangan Sistem Informasi Berbasis Website: Systematic Literature Review,” *Techno.Com*, vol. 21, no. 1, pp. 139–148, 2022, doi: 10.33633/tc.v21i1.5659.

[17] Y. Zerezghi, “Challenges in adopting agile methodology in public organisations IT project management–A systematic literature review,” 2023.

[18] H. Noprisson, “Implementation of Agile Software Development Methodology in Software Projects,” *JUSIBI (Jurnal Sistem Informasi dan Bisnis)*, vol. 5, no. 2, pp. 94–102, 2023, doi: 10.54650/jusibi.v5i2.521.

[19] B. W. Suwahyo, P. Setyosari, and H. Praherdhiono, “Pemanfaatan Teknologi Asistif Dalam Pendidikan Inklusif,” *Edcomtech: Jurnal Kajian Teknologi Pendidikan*, vol. 7, no. 1, p. 51, 2022, doi: 10.17977/um039v7i12022p055.
[20] H. Asfa and T. J. Gandomani, “Software quality model based on development team characteristics,” *International Journal of Electrical and Computer Engineering*, vol. 13, no. 1, pp. 859–871, 2023, doi: 10.11591/ijece.v13i1.pp859-871.
[21] Mohammad Ikbal Hossain, “Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management,” *International Journal For Multidisciplinary Research*, vol. 5, no. 5, 2023, doi: 10.36948/ijfmr.2023.v05i05.6223.
[22] D. De Silva and S. Lanka, “A Case Study of Test Automation in Agile Software Development,” *International Journal of Science and Engineering Applications*, vol. 12, no. 05, pp. 41–49, 2023, doi: 10.7753/ijsea1205.1013.
[23] C. Ragkhitwetsagul, J. Krinke, M. Choetkiertikul, T. Sunetnanta, and F. Sarro, *Adoption of automated software engineering tools and techniques in Thailand*, vol. 29, no. 4. 2024. doi: 10.1007/s10664-024-10472-6.
[24] V. Duran Espinoza *et al.*, “Available assessment tools for evaluating feedback quality: a scoping review oriented to education in digital media,” *Global Surgical Education - Journal of the Association for Surgical Education*, vol. 3, no. 1, pp. 1–19, 2024, doi: 10.1007/s44186-024-00239-4.
[25] S. J. Chakravarty Krishna, “A dissection of agile software development in changing scenario and the sustainable path ahead,” *International Journal of System Assurance Engineering and Management*, vol. 15, pp. 2606–2622, 2024.
[26] A. ALazzawi, Q. M. Yas, and B. Rahmatullah, “A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions,” *Iraqi Journal for Computer Science and Mathematics*, vol. 4, no. 4, pp. 173–190, 2023, doi: 10.52866/ijcsm.2023.04.04.014.
[27] Suwarno, S. A. Aklani, and N. Purwandi, “Examining the Impact of Software Testing Practices on Software Quality in Batam Software Houses,” *INOVTEK Polbeng - Seri Informatika*, vol. 10, no. 1, pp. 36–48, 2025, doi: 10.35314/5bmdas03.
[28] A. Mishra and Y. I. Alzoubi, “Structured software development versus agile software development: a comparative analysis,” *International Journal of System Assurance Engineering and Management*, vol. 14, no. 4, pp. 1504–1522, 2023, doi: 10.1007/s13198-023-01958-5.
[29] H. Saeeda, M. Ovais Ahmad, and T. Gustavsson, *Navigating social debt and its link with technical debt in large-scale agile software development projects*, vol. 32, no. 4. Springer US, 2024. doi: 10.1007/s11219-024-09688-y.
[30] D. A. Rebro, S. Chren, and B. Rossi, “Source Code Metrics for Software Defects Prediction,” *Proceedings of the ACM Symposium on Applied Computing*, pp. 1469–1472, 2023, doi: 10.1145/3555776.3577809.
[31] R. A. Hasan and I. A. Saleh, “Prediction of Software Anomalies Methods Based on Ensemble Learning Methods,” *Kufa Journal of Engineering*, vol. 16, no. 3, pp. 639–657, 2025, doi: 10.30572/2018/KJE/160336.
[32] A. ALazzawi, Q. M. Yas, and B. Rahmatullah, “A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions,” *Iraqi Journal for Computer Science and Mathematics*, vol. 4, no. 4, pp. 173–190, 2023, doi: 10.52866/ijcsm.2023.04.04.014.
[33] E. Kesavan, “A Comprehensive Review of Automated Software Testing Tools and Techniques,” *International Journal of Innovations in Science Engineering And Management*, pp. 14–20, 2025, doi: 10.69968/ijisem.2025v4i214-20.