Vifada Management and Social Sciences, E-ISSN 2987-1999

Designing Software Quality Assurance Framework Integrated With Scrum And RSM For Multi-Project

Devi Veronica [1*] *Muharman Lubis* [2] *Basuki Rahmad* [3]
[1,2,3] *Master of Information Systems Study Program, Faculty of Industrial Engineering, Telkom University*

Received: 2026, 02,06 **Accepted**: 2026, 05,10
Available online: 2026, 06,30

*Corresponding author.
E-mail addresses: devifirvie@student.telkomuniversity.ac.id

KEYWORDS

Keywords:
Software Quality Assurance; Scrum; RSM; Multi-Project; Design Science Research

Conflict of Interest Statement:
The author(s) declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ABSTRACT

Purpose: This study aims to design an integrated Software Quality Assurance (SQA) framework combining Scrum and the Recognize, Scrutinize, Materialize (RSM) approach to mitigate context switching, priority ambiguity, and technical debt in multi-project environments.

Research Design and Methodology: Utilizing a mixed-methods approach with sequential triangulation based on Design Science Research (DSR), secondary data were gathered from SOP studies at PT. Boer Technology. Framework evaluation involved a 1–5 Likert-scale questionnaire administered to a purposive sample of 9 active expert Scrum Masters

Findings and Discussion: Descriptive statistics and inter-rater reliability analysis confirm a high expert consensus, with a 99.02% cumulative positive response rate. Gwet's AC2 yielded a fair agreement of 0.2279, demonstrating that integrating RSM strengthens shift-left testing and continuous quality execution without compromising Scrum's iterative nature

Implications: This framework offers an original, implementable quality governance solution for medium-to-large software industries to enhance compliance, product stability, and business value. Future research should conduct empirical pilot implementations to test scalability and quantitatively measure long-term impacts using metrics like defect density

Introduction

Today, information systems are no longer viewed merely as technical tools, but also as strategic assets that determine the quality of decision-making, operational efficiency, and organizational competitiveness (Hera et al., 2024 ;Gorla et al., 2010). In this context, software quality is a critical factor that directly influences an organization's success—from technical, business, and user experience perspectives (Mastaneh & Mouseli, 2023).

Software engineering has evolved rapidly. Previously limited to coding activities, software engineering has now become a multi-dimensional discipline that emphasizes sustainability, quality, and the system's ability to adapt to changing business needs (Khan et al., 2021; Anwar, 2025). The challenges faced intensify when development takes place in a multi-project environment, where teams must manage multiple projects simultaneously with limited resources, dynamic priorities, and fast-paced delivery pressures.

In industry practice, particularly within IT consulting firms, Agile methodologies and the Scrum framework are widely adopted. This is due to their ability to support iterative development, intensive collaboration, and rapid adaptation to change (Schwaber & Sutherland, 2020; Ariesta et al., 2021). Scrum has made a positive contribution to quality through the mechanisms of empiricism, inspection, and adaptation structured within sprints (Alami & Krancher, 2022). However, various studies indicate that Scrum has limitations in supporting cross-project quality management, consistent documentation,

and quality control from the early design stages, particularly in a multi-project environment (Mortada et al., 2020; Shuib & Hassan, 2021).

On the other hand, Software Quality Assurance (SQA) serves as a systematic mechanism to ensure software quality throughout the Software Development Life Cycle (SDLC), encompassing the quality of requirements, design, code, and testing (Chouhan & Mathur, 2012; Wambua & Maake, 2022). Although SQA has evolved toward a more integrated approach with Agile and DevOps, its implementation in many organizations still tends to be reactive and focused on technical testing, rather than on building quality from the early stages (Mateen et al., 2017;Nikolova, 2020).

In response to these limitations, the RSM (Recognize, Scrutinize, Materialize) approach has emerged as a design methodology focused on understanding user needs, validating ideas, and ensuring design quality from the earliest stages (Sandy & Lubis, 2025;Lubis, 2023). The iterative, user-focused, and utility-oriented nature of RSM makes it compatible for integration with Agile frameworks such as Scrum. However, to date, there has been little research that explicitly integrates SQA, Scrum, and RSM into a single unified framework to address quality challenges in multi-project environments.

Given these conditions, this study aims to design an integrated Software Quality Assurance framework combining Scrum and RSM, capable of bridging development process control, design validation, and end-to-end quality assurance. This framework is expected to serve as a conceptual and practical solution for improving software quality, while also contributing scientifically to the development of the disciplines of information systems and software engineering.

Literature Review

Software Engineering

Software engineering is defined as an established process with its own methods, procedures, and application structure, thus going beyond mere technical coding activities. Over time, this discipline has shifted its primary focus from simply ensuring machine functionality to ensuring the highest quality of software in accordance with its intended purpose. Furthermore, software engineering encompasses strategic capabilities in designing large-scale systems, managing complexity, and overseeing the continuous evolution of software. The application of this discipline is cross-sectoral, utilizing standardized methods and tools that are widely implemented across a range of applications, from commercial information processing to real-time industrial control and automated transportation systems (Khan et al., 2021).

Software engineering is no longer understood merely as the skill of writing code; rather, it has evolved into a complex discipline with a holistic approach to system design and quality management. Critical attributes such as performance, security, and maintainability must be designed from the earliest stages of development and verified through rigorous testing strategies throughout the software lifecycle, rather than being treated as separate phases. This paradigm shift demands a balance between technical innovation and operational stability, where system architecture serves as the primary framework for addressing complexity and maintaining long-term integrity (Anwar, 2025).

Software Quality Assurance

Software Quality Assurance (SQA) is a structured and systematic mechanism aimed at ensuring and documenting the quality of every artifact or deliverable produced at all stages of the Software Development Lifecycle (SDLC) (Chouhan & Mathur, 2012). Generally, SQA encompasses software quality control, which evaluates the extent to which a product meets established requirements, as well as software testing, which refers to the process of executing a program or application to identify potential defects (Wambua & Maake, 2022). This testing can be conducted through various methods and techniques, such as unit testing, integration testing, system testing, and user acceptance testing, all aimed at finding and fixing bugs or errors before the software is released to end-users. Through a combination of strict quality control and testing, SQA plays a crucial role in producing reliable, efficient software that meets user expectations.

Software Quality Assurance (SQA) plays a vital role in ensuring the success of software development. There are several key objectives to be achieved through the implementation of SQA, namely (Mateen et al., 2017):

1. **Preventing Project Failure:** Significantly reduce the risk of project failure by incorporating QA processes throughout all stages of the system development life cycle (SDLC).
2. **Cost Efficiency:** Detect errors early on (during the design phase) to avoid much more expensive and difficult-to-fix costs if they are discovered after the project is completed.
3. **Ensuring Compliance:** Ensuring that the software built fully meets the established functional requirements, deadlines, and budget.
4. **Maintaining System Stability:** Performing integration and regression testing to ensure new features function properly without breaking existing features.

The Software Quality Assurance (SQA) process plays a crucial role in ensuring the quality of the software produced, and is typically led by a QA Tester or QA Engineer. A QA Tester is responsible for testing software, creating test scenarios, and compiling test result reports. Meanwhile, a QA Engineer focuses more on developing automated tests, providing feedback on the tested application, and communicating with relevant teams such as UI/UX, backend, and product managers, as well as handling customer complaints (Hariyanto, 2020).

Agile Software Development Methodology: Scrum

Agile Software Development is a collection of software development methods (Agile Development Methods). Meanwhile, Agile methods are approaches used for incremental development that focus on rapid development, phased software releases, reducing process overhead, and producing high-quality code, with the development process directly involving customers (Ariesta et al., 2021). The foundation of this method is the principle of short-term system development with a high degree of adaptability to all changes.

One of the agile software development approaches is Scrum. The Scrum approach is based on empiricism, using an iterative and incremental approach through self-organizing, cross-functional teams, as well as a set of roles, events, artifacts, and formal rules to optimize predictability and control risk (Hussain et al., 2025). This approach will accelerate the delivery cycle and also improve overall product quality through continuous testing and validation (Fagarasan et al., 2021)

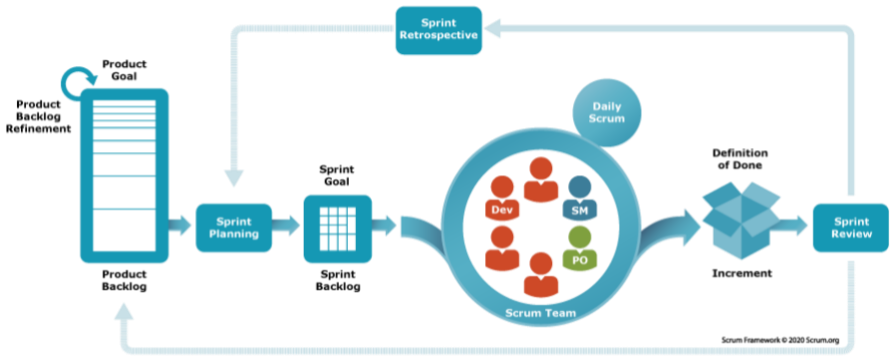

Figure 1. Scrum Framework Process
Source: www.scrum.org

The Scrum framework is a lightweight yet effective process that begins with the Product Backlog, prioritized list of product requirements which is managed by the Product Owner to maximize value (Schwaber & Sutherland, 2020). All development work takes place within a Sprint, a fixed-duration timeframe of one month or less, which serves as the heartbeat of Scrum (Schwaber & Sutherland, 2020). A Sprint begins with Sprint Planning, where the Scrum Team collaborates to select items from the Product Backlog and define the Sprint Goal (Muller et al., 2021). During the Sprint, developers carry out their work, coordinated through 15-minute Daily Scrums, where they check progress toward the Sprint Goal and adjust their plans (Muller et al., 2021). The result of the work performed is an Increment—a tangible step toward the Product Goal that must meet the Definition of Done (Ramin et al., 2020). At the end of the Sprint, the Scrum Team holds a Sprint Review to review the Increment's results with stakeholders, followed by a Sprint Retrospective to identify and plan improvements that will enhance quality and effectiveness for the next Sprint (Schwaber & Sutherland, 2020). It is

important to note that although this framework is clear, software teams sometimes deviate from established Scrum practices due to team uncertainty, organizational structure, or the complexity of the work (Mortada et al., 2020).

A Scrum team typically consists of three main roles or specific areas of responsibility:

1. The Product Owner is responsible for managing the product backlog (Haputhanthrige et al., 2024), is accountable for requirements or user stories, communicates with customers, formulates and prioritizes user stories, and accepts the implementation of user stories at the end of each sprint (Klopp et al., 2020).
2. The Scrum Master is responsible for fostering an understanding of Scrum theory and practice among all members, both within the Scrum team and the organization (Haputhanthrige et al., 2024), ensuring the team operates effectively and productively, and maintaining adherence to the Scrum process (Klopp et al., 2020).
3. Developers are responsible for creating an increment (deliverable) that can be used during the sprint (Kadenic et al., 2023), organizing themselves (self-organized), and taking responsibility for planning and achieving sprint goals, as well as implementing user stories from the Sprint Backlog to produce a potentially shippable increment at the end of the sprint (Klopp et al., 2020).

Software Quality Assurance in Scrum

In the Agile software development paradigm, specifically within the Scrum framework, Software Quality Assurance (SQA) has evolved into an integral and ongoing component. Unlike traditional, siloed approaches, SQA in Scrum functions not merely as a technical verification mechanism but as a synergistic combination of technical practices, iterative processes, and team dynamics to ensure the product meets defined requirements incrementally (Alami & Krancher, 2022; Shuib & Hassan, 2021).

The implementation of SQA in Scrum is supported by two main pillars: traditional technical practices and process-based validation mechanisms:

1. Traditional Technical Practices

Although Scrum emphasizes agility, standard SQA practices are maintained and reinforced. These practices include software testing, which encompasses Unit Testing, System Testing, End-to-End Testing, and User Acceptance Testing (UAT). Additionally, code reviews and the use of static analysis tools have become standard procedures for detecting defects early on. In some implementations, the role of specialists such as Quality Assurance Analysts is retained as "gatekeepers of quality" (Alami & Krancher, 2022).

2. Process-Based Validation

The primary mechanism for adopting SQA in Scrum lies in iteration reviews (such as the Sprint Review). This serves as a validation channel where the product increment is evaluated based on acceptance criteria. Customer involvement in this phase is crucial for providing feedback, determining the product's acceptance status, and offering input for planning the next sprint (Shuib & Hassan, 2021).

RSM (Recognize, Scrunitize, Materialize)

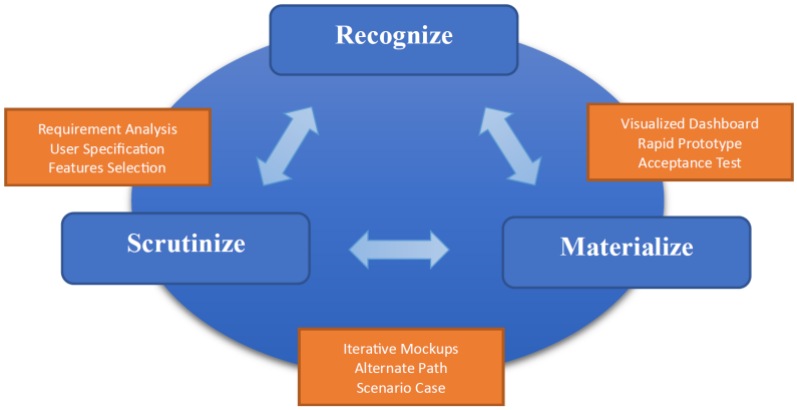

Figure 2. RSM Design Approach Simplified

Source: Lubis, 2023a

The RSM Design Approach is a methodological framework designed to address the complexities of product design and innovation. RSM, an acronym for Recognize, Scrutinize, and Materialize, offers a structured yet iterative, user-focused approach (Sandy & Lubis, 2025; (Lubis, 2023a). This methodology is an ongoing process compatible with Agile methods, allowing it to adapt dynamically to technological changes and user needs (Sandy & Lubis, 2025).

The design process in RSM is divided into three main, mutually integrated phases:

1. **Recognize**:This stage serves as the initial foundation, focusing on problem identification, understanding user needs, and defining specifications (Mardoyo et al., 2024). In this phase, designers conduct requirement analysis and feature selection to ensure the solution's relevance (Lubis, 2023a). This activity involves the use of User Cards (covering the Think, Feel, Do, Say aspects) and contextual orientation to map user behavior (Lubis, 2023b). Additionally, analysis is conducted using an empathetic (empathize) and sympathetic approach to understand the consequences users face (Lubis, 2023b). The results of this stage include user specifications, user stories, and definitions of expected value (user desirability) (Lubis, 2023a; Lubis, 2023c).
2. **Scrutinize:** This phase involves in-depth examination, reflection on inspiration, and testing of the proposed design plan (Sandy & Lubis, 2025). The primary focus of this stage is to validate ideas through case scenarios and alternative paths before full-scale development (Lubis, 2023a). Activities in this phase include UI/UX design, asset creation (such as 3D assets in a VR context), developing test scenarios, and formulating the application's vision and objectives based on market opportunities (Mardoyo et al., 2024; Lubis, 2023b). The outputs of this phase include alternative sketches, iterative mockups, and a business feasibility analysis (Lubis, 2023a; Lubis, 2023b).
3. **Materialize**: The final stage is the realization or materialization of the solution into a functional product ready for testing within a digital ecosystem (Sandy & Lubis, 2025). This phase ensures technical integration and user acceptance validation. These activities include rapid prototyping, dashboard visualization, and the conduct of acceptance tests (Lubis, 2023a). In the context of system development, this stage also involves integrating application functions with existing digital solutions (Lubis, 2023b). The final product can be a functional prototype, a model, or a complete application that has been validated through user feedback and technology feasibility testing (Mardoyo et al., 2024;(Lubis, 2023a).

Findings and Discussion

Findings

This study employs a mixed-methods approach using sequential triangulation. Qualitative data were obtained from exploratory interviews conducted at the outset and thematic analysis of experts' open-ended critiques. Meanwhile, quantitative data were derived from a 1–5 Likert scale assessment, which was statistically transformed to measure the degree of consensus and the reliability of expert validation.

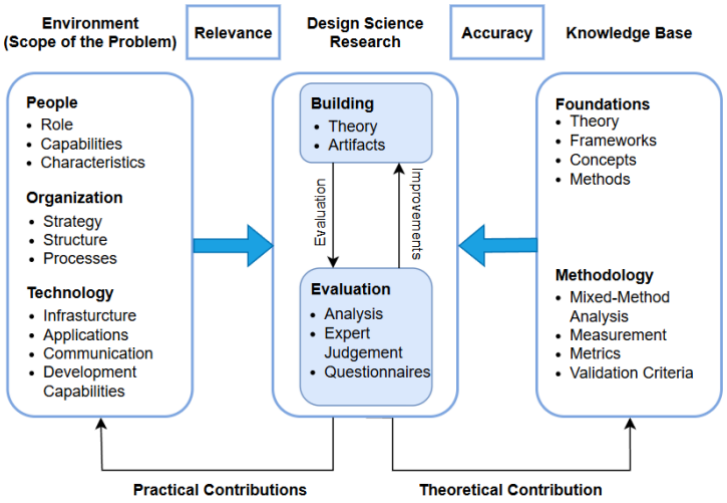

Figure 3. Research Methodology

The conceptual model used in this study is based on the Design Science Research (DSR) framework. The DSR approach was chosen because this study aims to produce an artifact in the form of a Software Quality Assurance framework capable of solving practical problems within an organizational setting. This framework connects three main domains (Hevner et al., 2004): Environment, DSR Research (IS Research), and Knowledge Base.

Research Design

The research design of this study follows the conventional structure that organizes the essential components of a scientific paper, in order to ensure logical coherence and the completeness of the research content. This structure serves as a methodological framework that organizes the flow of the presentation. The following is an overview of the research structure based on the conceptual model of Design Science Research, which consists of six phases in this study.

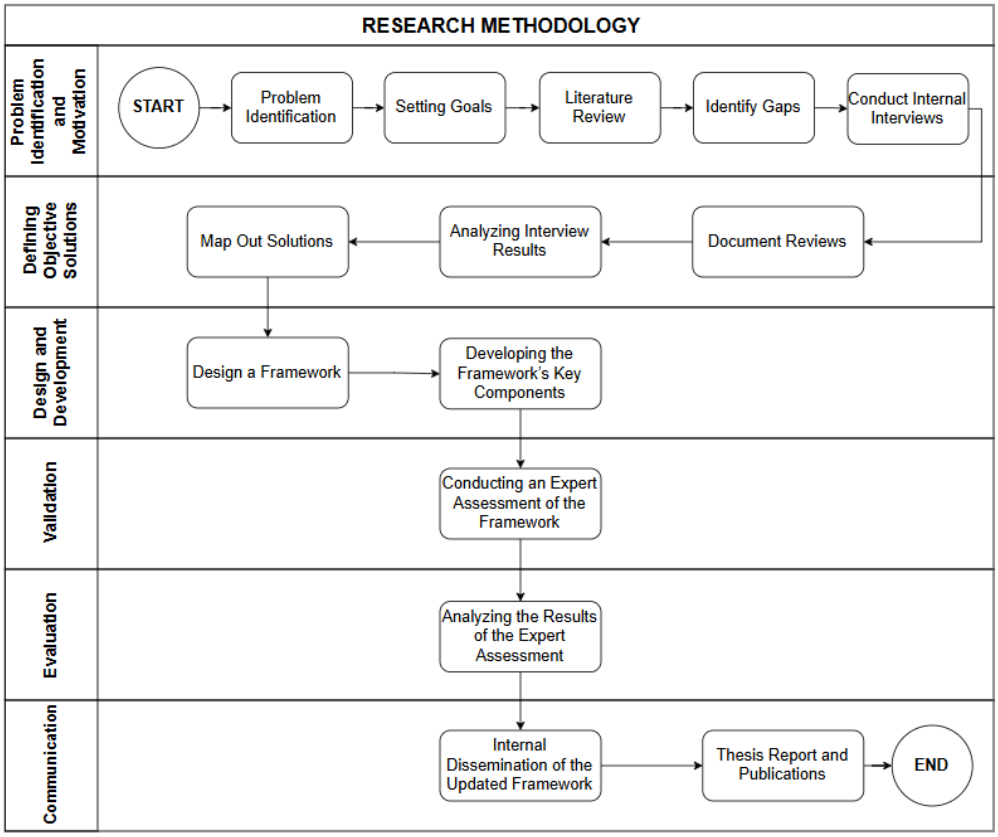

Figure 4. Research Design

Data Source and Collection Methods

Data collection was conducted in a phased and integrated manner, particularly during the Problem Identification and Evaluation phases. The methods used included in-depth interviews, document reviews, and expert judgment in the form of questionnaires.

Table 1. Data Source

Type of Data	Instruments	Research Objectives	Data Sources
SQA Needs, Constraints, and Challenges (Qualitative Data)	Interview Guidelines (Semi-Structured, In-Depth Interviews)	Problem Identification and Justification of Solutions	Practitioners at PT. Boer Technology (Senior Management, Development Team, and Product Team)
Standard Operating Procedures (SOPs), SQA Framework, and Current Business Processes (Secondary Data)	Document Review	Definition of Solution Objectives; Design and Development (Analyzing gaps with Scrum and RSM theory, and supporting solution design)	Policy documents, SOPs, and reports related to business processes at PT. Boer Technology.
Assessment of Framework Acceptance and Effectiveness (Quantitative & Qualitative Data)	Questionnaire	Evaluation (Measuring success, obtaining expert judgment for empirical validation and measurable assessment)	Professional Scrum Masters currently active in the company, with at least 2 years of experience

Internal Sources and Interview Results

The informants involved in this initial data collection were key practitioners representing various perspectives (technical, managerial, and operational).

Table 2. Internal Sources and Interview Results

Source	Position	Experience	Interview Results
IN1	Vp of Product Development	1 years	He emphasized that the company's quality vision is "Delivering high-quality value at the right time." QA plays the most critical role, although the current organizational structure is not yet considered optimal, as it ideally requires at least two full-time QA team members and 100% regression testing. The biggest challenges in multi-project environments are timeline management and maintaining focus. The strategies used to maintain quality under pressure include adjusting project scope and rescheduling sprints.
IN2	Manager of Development	10 years	The main obstacle to team coordination was setting sprint goals. The biggest factors affecting quality were overlapping project deadlines and communication between teams. To address the limitations of the QA team, the strategy was to prioritize projects and automate testing.
IN3	Quality Assurance	1 years	He admits that he often feels overwhelmed when handling more than one project. His biggest challenge in managing multiple projects is frequent context switching and a lack of focus on a single project. The most common testing obstacles he faces are incomplete documentation and a lack of tooling or automation support.
IN4	Programmer	5 years	He frequently communicates with QA, but the main barriers to collaboration are differing definitions of "Done" and a lack of clear feedback from QA. Technical debt arises from rushed development and hasty testing. Working on multiple projects simultaneously affects quality due to a lack of focus and mastery of the projects. Effective collaboration requires communication, documentation and feedback.
IN5	Programmer	8 years	He believes that technical debt significantly impacts quality because opting for quick-fix solutions leads to inconsistent and poorly tested code. Working on multiple projects affects code quality because it requires constantly shifting focus. The biggest challenge in multi-project Scrum is the need to set aside time to prepare progress reports for the Daily Scrum, which takes away from technical work.
IN6	Programmer	10 years	He faced collaboration challenges, such as a lack of shared understanding regarding bug priorities and differences in the Definition of Done. He often found it distracting to divide his focus among multiple projects at once. The biggest challenge in maintaining quality across multiple projects was the frequent context switching that occurred without first taking the time to review the work.
IN7		3 years	Collaboration with QA is sometimes considered effective. The main obstacles to collaboration are differing views on the Definition of Done and incomplete testing

	Fullstack Developer		documentation. Technical debt (neglecting unit testing) increases the likelihood of bugs in older features. Working on multiple projects simultaneously affects quality due to differences in new technologies and the lack of a style guide. To improve collaboration, we recommend holding meetings to share knowledge about how QA and development teams work.
IN8	Programmer	6 years	The biggest challenge in ensuring system stability after code changes is retesting to make sure no existing features are broken. The biggest obstacles to maintaining quality in a multi-project environment are context switching and project prioritization. He suggested that developers and QA work together to create test cases.
IN9	Product Manager	1 years	He defines a successful product as one that is usable, has utility value, and market value, with the key quality indicators being minimal bugs, user experience, and system stability. Communication with the technical team is already effective through Scrum ceremonies. The biggest weakness of QA at present is that the checking process, which should be sufficient at the QA stage, must be repeated on the Product Owner's side. The biggest challenges in a multi-project environment are the Product Owner's role overload, interdependencies between projects, and overlapping backlog prioritization.
IN10	Associate Product Owner	1 years	Prioritizing the backlog in a multi-project environment is considered quite complex because it requires gathering too much information, which divides attention. He noted that there have been instances where sprint outcomes did not meet quality expectations due to misinterpretations of the requirements in the tickets. The main challenge in obtaining quality information is the lack of standard documentation and quality dashboards. To ensure alignment, he recommended involving QA from the sprint planning phase and implementing a QA checklist confirmation process starting from the PRD..
IN11	Product Operation Support	6 months	The most common quality issues encountered by users are functional bugs and system crashes or errors. These issues slip through to production because users lack a full understanding of the system, leading to differing perceptions. The fact that multiple projects are currently underway affects the technical team's response time when handling user issue reports.
IN12	Product Operation Analyst	6 months	Common quality issues encountered by users include functional bugs and system crashes/errors, which are caused by insufficient testing. He observed quality discrepancies across projects due to the multi-project environment. He only analyzed quality data when major issues arose. To improve post-release quality, proactive performance monitoring and logging are necessary.

External Respondents for Expert Judgement

The questionnaire respondents in this study were Scrum practitioners with experience and expertise in implementing Scrum in software development environments. Respondents were selected through purposive sampling to ensure that the assessment of the framework came from individuals with firsthand understanding of Agile processes and Software Quality Assurance (SQA) practices.

Table 3. External Respondents (Experts)

Respondents	Experience as a Scrum Master	Experience with Scrum	Scrum Sertification
ER1	5 years	5 years	PSM 1
ER2	13 years	13 years	SMAC
ER3	15 years	15 years	PSM 1
ER4	5 years	7 years	PSM 1
ER5	5 years	6 years	Certified Scrum Professional
ER6	5 years	5 years	PSM 1
ER7	4 years	7 years	PSM 1
ER8	2 years	2 years	-
ER9	3 years	3 years	PSM 1

Discussion

Multi-Project Challenges and Impact on Quality

The nature of a multi-project work environment demands a high degree of flexibility; however, it also risks causing a decline in quality if not balanced by adaptive governance. An imbalance between workload and the number of projects running in parallel triggers various obstacles that affect all levels of the organization, from strategic management down to technical development teams. Broadly

speaking, the impact of this multi-project environment creates two critical challenges that hinder the efficiency of quality assurance, namely:

1. Context Switching dan Prioritization

The biggest challenge in a multi-project environment is the loss of focus and quality due to context switching, which has been consistently identified by the Development and QA teams. Developers specifically stated that multi-project environments negatively impact code quality due to a lack of focus, project mastery, and differences in technological approaches across projects. From a management perspective, the VP of Product Development identifies timeline management and the need to manage priorities and focus as the primary strategic challenges. This prioritization issue is also felt at the product level, where Product Managers face difficulties in organizing overlapping backlogs, and Associate Product Owners cite challenges in setting priorities due to the need to gather too much information.

2. Technical Debt and Regression

The development team directly attributes the emergence of bugs and quality issues to technical debt, which arises from rushed development and hasty testing. When unit testing is neglected, newly added features have the potential to interfere with existing ones, leading to a high risk of regression bugs. Therefore, the biggest challenge in ensuring stability after code changes is the necessity of conducting thorough retesting to verify that no existing features are broken.

Data Model Development

Data model development in this study was conducted to construct a systematic conceptual representation of the relationship between Scrum processes, Software Quality Assurance (SQA) activities, and the RSM (Recognize, Scrutinize, Materialize) approach based on the empirical data obtained. This data model serves as the foundation for designing an integrated SQA framework capable of addressing software quality issues in a multi-project environment.

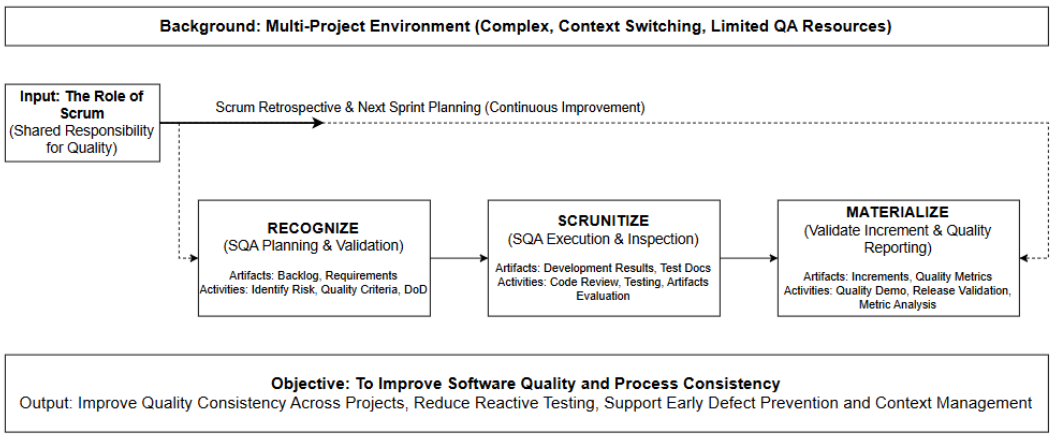

Figure 5. Data Model Development

The initial stage of data model development begins with the identification of key concepts and entities derived from interviews and observations. These concepts include the Scrum process (events, artifacts, and roles), SQA activities, human resources, projects, and software quality as the primary output. The data analysis revealed that software quality is influenced not only by testing activities but also by process consistency, role clarity, and quality control mechanisms from the early stages of development.

Based on these concepts, a data model was developed by positioning the project as the central entity managed using Scrum and executed in parallel within a multi-project context. Each project consists of several sprints. SQA activities are positioned as entities integrated into each sprint, rather than as separate activities at the end of the development cycle. This integration enables quality to be continuously controlled throughout the development process.

The RSM approach is used as a framework for grouping quality activities within the data model. In the Recognize phase, the data model represents activities for identifying quality requirements, risks, and acceptance criteria related to the backlog and sprint planning. The Scrutinize phase includes activities for evaluating and inspecting Scrum artifacts, such as the backlog, development deliverables, and test documentation. The Materialize phase represents the realization of quality improvements in the form of validated software increments ready for release.

The data model also accommodates the relationship between roles in Scrum and SQA activities. Each role contributes to quality, whether in planning, development, or evaluation of work results. This relationship illustrates that quality responsibility does not rest solely with the QA function but is a shared responsibility within the development team.

Overall, the development of the data model resulted in a conceptual structure that illustrates the software quality management process integrated with Scrum and RSM. This model serves as the foundation for designing an integrated SQA framework that supports quality control from the early stages, enhances quality consistency across projects, and reduces reliance on reactive testing at the end of development.

Proposed SQA Framework Integrated with Scrum and RSM

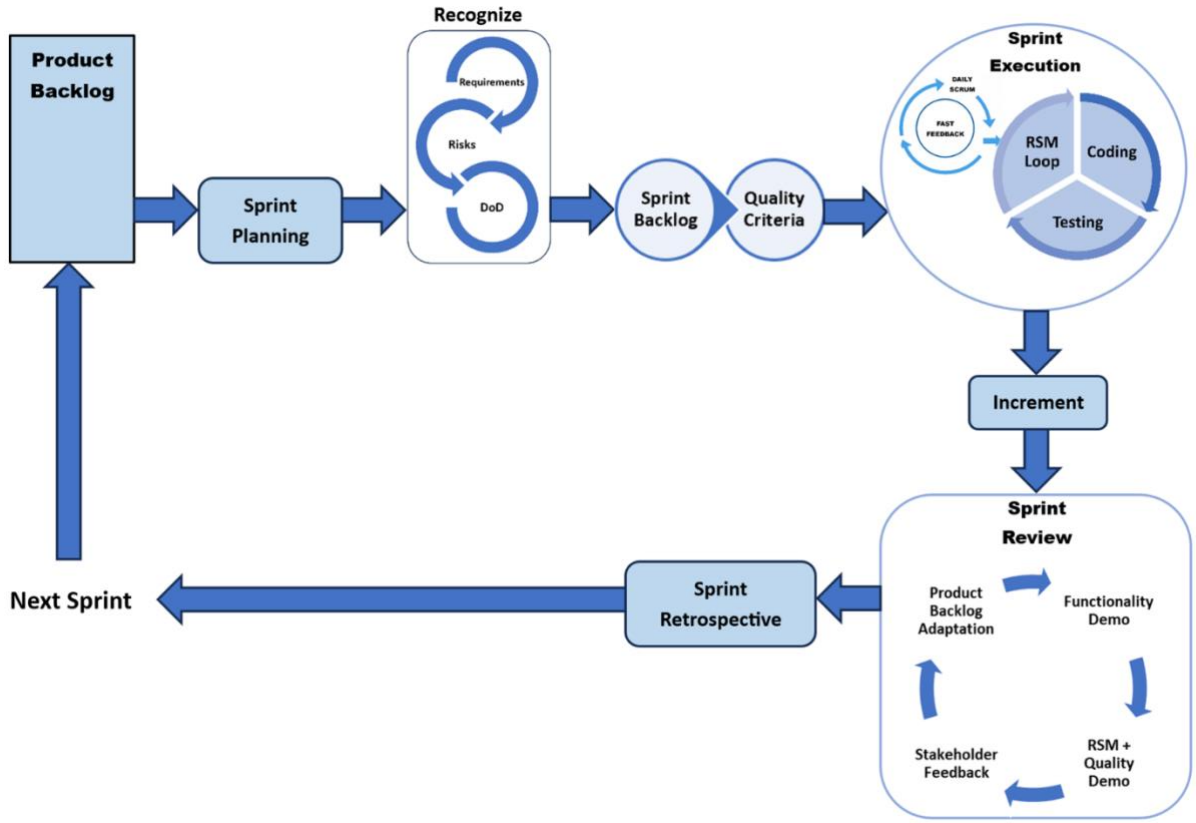

Figure 6. Proposed SQA Framework Integrated Scrum and RSM

This framework is divided into three phases:

1. Quality Planning Phase (Planning & Recognize)

Each Sprint begins with a Sprint Planning session that not only defines the work but also establishes quality standards from the outset. Items are selected from the Product Backlog and then analyzed using the Recognize (RSM) approach. During this phase, the team ensures that feature requirements are clearly understood, identifies risks that may arise during development, and agrees on a Definition of Done (DoD) as the minimum quality standard. Additionally, the team establishes Quality Criteria as a reference for testing and evaluation throughout the Sprint. The main outputs of this phase are the Sprint Backlog and the list of mutually agreed-upon quality criteria.

2. Continuous Quality Execution Phase (Sprint Execution & RSM Loop)

During the Sprint, development proceeds iteratively through a cycle of Coding and Testing that repeats continuously. Every code change is immediately tested to ensure compliance with the Quality Criteria and DoD. This process is supported by Daily Scrums and fast feedback mechanisms, so that quality issues can be detected and fixed immediately without waiting for the end of the Sprint. This approach helps the team build quality directly into the product (build quality in). The output of this phase is an Increment that is ready for review and potentially for release.

3. Quality Review and Adaptation Phase (Sprint Review, Retrospective, and Adaptation)

At the end of the Sprint, the team conducts a Sprint Review to demonstrate the Increment to stakeholders. In addition to showcasing the product's functionality, the team also explains compliance with quality standards and the testing activities that have been performed. Feedback from stakeholders is used to update the Product Backlog. Afterward, the team holds a Sprint Retrospective to evaluate the effectiveness of the workflow and the integration of SQA–RSM, as well as to determine improvements to be implemented in the next Sprint. Through this process, each Sprint not only delivers new features but also continuously improves product quality and processes.

Events within this framework follow the Scrum cycle, with the addition of RSM-based quality activities at each stage.

1. Sprint Planning (Recognize Integration): used to define Sprint items while simultaneously performing the Recognize phase, which involves clarifying requirements, identifying risks, and establishing the Definition of Done and Quality Criteria. This event serves as the foundation for quality planning in every Sprint.
2. Sprint Execution and Daily Scrum (RSM Loop): During the Sprint, development proceeds through the iterative Materialize and Scrutinize cycles. The Daily Scrum serves as a mechanism for rapid feedback to monitor progress, detect quality issues, and manage risks that arise during the Sprint.
3. Sprint Review (Final Scrutinize): The Sprint Review serves as the final quality verification stage for the Increment with stakeholders. In addition to demonstrating functionality, the team reviews compliance with Quality Criteria and the application of RSM as part of the final Scrutinize phase.
4. Sprint Retrospective: The Sprint Retrospective is used to evaluate the effectiveness of work processes and the integration of SQA–RSM. The evaluation results form the basis for process improvements and quality enhancements in the next Sprint.

The artifacts in this framework serve as evidence and quality control tools generated from each event.

1. Product Backlog: contains all product requirements and serves as the primary source for Sprint planning. This artifact is continuously updated based on the results of the Sprint Review and Retrospective.
2. Sprint Backlog: is the result of Sprint Planning that has gone through the Recognize phase. This artifact reflects the Sprint work as well as the team's commitment to the established quality standards.
3. Definition of Done (DoD): is the mutually agreed-upon minimum criteria for completing work. The DoD is used as a quality evaluation standard during the Sprint and at the end of the Sprint.
4. Quality Criteria: specific quality criteria derived from requirements and risks. This artifact serves as the primary reference for Scrutinize (testing) activities and quality evaluation during the Sprint Review.
5. Increment: the development output at the end of the Sprint that meets the DoD and Quality Criteria and is in a potentially releasable state.

The roles of the Scrum team in this framework have been adapted based on the three existing phases:

Table 4. Role Scrum Team

Role	Quality Planning Phase	Sustainable Quality Executing Phase	Quality Review and Adaptation Phase
Product Owner	Prioritizing the Product Backlog Defining business requirements and product value Ensuring that functional and non-functional requirements are represented	Providing clarification on requirements as needed Ensuring that implementation aligns with business value	Validating the alignment of the Increment with business requirements Adapting the Product Backlog based on feedback
Scrum Master	Facilitate Sprint Planning Maintain alignment between Scrum and RSM Ensure the Recognize process runs systematically	Facilitate Daily Scrum Eliminate quality or process impediments Maintain RSM Loop discipline	Facilitate Sprint Review and Retrospective Guide discussions on SQA & RSM process improvements
Developer	Analyze the technical feasibility of backlog items Identify initial technical risks Provide estimates and dependencies	Perform coding (implementation) Write unit tests or perform self-testing Quickly fix defects based on feedback	Demonstrate the technical solution Explain the implementation and improvement approach
QA Engineer	Lead the identification of Quality Criteria Identify quality risks (defect-prone areas, compliance, etc.) Develop or validate the Definition of Done	Conduct continuous testing (scrutinization) Verify compliance with Quality Criteria Provide rapid feedback on quality findings	Provide a demonstration of quality (RSM + Quality Demo) Explain compliance with Quality Criteria Communicate residual quality risks
Tim Scrum (Collective)	Joint discussions during the Recognize phase (Requirements–Risks–DoD) Agreeing on the Sprint Backlog and Quality Criteria	Implementing the principle of shared quality ownership Actively collaborating in a fast feedback loop	Reflecting during the Sprint Retrospective Agreeing on process improvement actions for the Next Sprint
Stake Holder	None	None	Provide feedback on functionality and quality Evaluate whether the product meets expectations

Validation of Expert Judgement

Descriptive statistical analysis was conducted to provide an overview of the initial assessment trends of the 9 (nine) expert raters regarding the 34 items in the questionnaire. The statistical indicators used in this analysis are the mean to measure the central tendency of the data, and the standard deviation to measure the extent of variation or dispersion of the data from its mean.

The results of the descriptive analysis for all items are grouped according to the framework's assessment dimensions and presented in the following table:

Table 5. Descriptive Statistics

Code	Questionnaire Item	Total Score	Mean	Standard Deviation	Rating Trend
P1	The framework maintains the Scrum workflow from the Product Backlog to the Next Sprint.	41	5	0,527	Strongly Agree
P2	RSM integration does not eliminate the iterative and incremental nature of Scrum.	39	4	0,5	Agree
P3	The Scrum Master's role remains clear in guiding the process and ensuring quality.	40	4	0,726	Agree

P4	Scrum artifacts (Product Backlog, Sprint Backlog, Increment) are used consistently.	41	5	0,726	Strongly Agree
P5	Recognize clarifies user requirements before the Sprint.	34	4	1,202	Agree
P6	Quality risks are identified as early as Sprint Planning.	36	4	1,225	Agree
P7	Definition of Done The Definition of Done becomes more measurable through Recognize.	36	4	1,225	Agree
P8	Recognize aligns stakeholder and team expectations.	38	4	0,833	Agree
P9	Scrutinize strengthens quality control during Sprint Execution.	34	4	1,202	Agree
P10	Coding and testing are integrated into the RSM Loop.	40	4	0,527	Agree
P11	Daily Scrums are effective for quality feedback.	36	4	0,866	Agree
P12	Bugs and quality risks are detected earlier through Scrutinize.	39	4	0,866	Agree
P13	The increment meets the Definition of Done.	40	4	0,527	Agree
P14	Materialize ensures the product is ready for stakeholder validation.	39	4	0,5	Agree
P15	Acceptance testing is conducted systematically prior to the Sprint Review.	38	4	0,667	Agree
P16	The Sprint product has good usability.	38	4	0,667	Agree
P17	The framework supports process-based quality control.	37	4	0,928	Agree
P18	Continuous improvement is carried out through the Sprint Retrospective.	39	4	1	Agree
P19	Quality decisions are well-documented.	38	4	0,833	Agree
P20	Product quality is consistent across Sprints.	37	4	0,782	Agree
P21	Functional suitability improves.	40	4	0,527	Agree
P22	Reliability improves.	37	4	0,782	Agree
P23	Usability improves	35	4	0,782	Agree
P24	Maintainability improves	37	4	0,601	Agree
P25	Compatibility improves	38	4	0,667	Agree
P26	The framework enhances the product's business value.	36	4	0,866	Agree
P27	The framework improves the Scrum team's performance.	37	4	1,054	Agree
P28	The framework enhances the ability to adapt to change.	39	4	1	Agree
P29*	The framework adds a significant administrative burden.	24	3	1,323	Neutral
P30	The framework accelerates the bug detection process.	37	4	1,054	Agree
P31	The framework is easy for the Scrum team to understand.	36	4	1	Agree
P32	The framework is realistic to implement on medium-to-large projects.	29	3	1,093	Neutral
P33	RSM integration strengthens SQA practices within Scrum.	36	4	0,866	Agree
P34	The framework is worth recommending for improving software quality.	33	4	1	Agree

Based on the descriptive statistics table, several key data patterns can be analyzed as follows:

1. Acceptability and Positive Validation of the Framework

Overall, the rating trend indicates a very positive response from the raters, with the vast majority of indicators falling into the "Agree" and "Strongly Agree" categories. This indicates that the developed framework is considered effective, relevant, and capable of making a tangible contribution to supporting the Scrum workflow and improving software development quality. The highest consensus (Strongly Agree) was achieved regarding the maintenance of the Scrum workflow from the Product Backlog to the Next Sprint (P1) and the consistency in the use of Scrum artifacts (P4).

2. Variability in Ratings Regarding Administrative Burden and Implementation

Although the overall trend is positive, two indicators resulted in the "Neutral" category: P29* (The framework adds a significant administrative burden) with a Mean score of 3, and P32 (The framework is realistic to implement on medium-to-large projects) with a Mean score of 3. These neutral ratings, accompanied by a relatively high standard deviation (SD > 1.0), indicate divergences or variations in perspective among the raters regarding the potential for increased administrative burden and the feasibility of implementing the framework on medium-to-large projects.

3. Level of Consensus and Data Distribution (Standard Deviation)

Analysis of the standard deviation (SD) values reveals variations in the level of consensus among raters, which can be divided into two main clusters:

- High Consensus (SD < 1.0): The majority of indicators have low standard deviation values (e.g., P1, P2, P10, P13, P21). This demonstrates a solid consensus and uniformity of views among the raters in assessing the success of these functions.
- High Variability (SD > 1.0): Several indicators, such as P5, P9, P27, P29*, P30, P31, and P32, have standard deviation values reaching or exceeding 1.0. This figure reflects significant polarization or differences in perception in the field, indicating that aspects related to team understanding, bug detection acceleration, performance changes, and workload are still perceived differently by each rater.

The frequency distribution data from the 9 expert respondents is as follows:

Table 6. Frequency Distribution

Code	Score 1 (SD)	Score 2 (D)	Score 3 (N)	Score 4 (A)	Score 5 (SA)	Total Experts
P1	0	0	0	4	5	9
P2	0	0	0	6	3	9
P3	0	0	1	3	5	9
P4	0	0	1	2	6	9
P5	1	0	1	5	2	9
P6	1	0	0	5	3	9
P7	1	0	0	5	3	9
P8	0	0	2	3	4	9
P9	1	0	1	5	2	9
P10	0	0	0	5	4	9
P11	0	0	3	3	3	9
P12	0	0	2	2	5	9
P13	0	0	0	5	4	9
P14	0	0	0	6	3	9
P15	0	0	1	5	3	9
P16	0	0	1	5	3	9
P17	0	1	0	5	3	9
P18	0	1	0	3	5	9
P19	0	0	2	3	4	9
P20	0	0	2	4	3	9
P21	0	0	0	5	4	9
P22	0	0	2	4	3	9
P23	0	0	3	4	2	9
P24	0	0	1	6	2	9
P25	0	0	1	5	3	9
P26	0	1	0	6	2	9
P27	0	1	1	3	4	9
P28	0	1	0	3	5	9
P29*	2	3	0	4	0	9
P30	0	1	1	3	4	9
P31	0	1	1	4	3	9
P32	1	1	2	5	0	9
P33	0	1	0	6	2	9
P34	0	1	3	3	2	9
Total	7	13	32	145	109	306

Based on the accumulation of total marginal scores, the distribution characteristics of the framework component assessments can be summarized as follows:

1. Overwhelmingly Positive Acceptance of the Framework

Overall, the framework under evaluation received a very high level of acceptance and validation from the experts. Of the total 306 evaluation responses received, the absolute majority fell into the positive categories: 145 responses (47.38%) rated Score 4 / Agree, and 109 responses (35.62%) rated Score 5 / Strongly Agree. Combined, 83% of all expert evaluations indicated agreement to strong agreement that this framework possesses good functionality, effectiveness, and relevance in supporting software development workflows.

2. Indicators with the Highest Consensus (Absolute Validity)

The experts demonstrated a very solid consensus (fully aligned with the “Agree” and “Strongly Agree” options, with no opinions leaning toward neutral or negative) on several key indicators, including: P1, P2, P10, P14, P21, and P25: All experts (9 individuals) provided ratings focused entirely on Score 4 and

Score 5. This demonstrates the framework's exceptionally high reliability, particularly regarding workflow maintenance, process integration, testing, and system functionality and compatibility.

3. Polarization of Expert Opinions and Critical Comments on P29*

Although the aggregate trend is positive, there is a fairly sharp divergence of views on item P29* (The framework adds a significant administrative burden). This indicator is the only point where the data distribution is concentrated on both negative and positive ratings simultaneously, namely Score 1 (Strongly Disagree) by 2 experts, Score 2 (Disagree) by 3 experts, and Score 4 (Agree) by 4 experts. This split distribution pattern indicates a polarization of perceptions among experts regarding the potential for the framework to add an administrative burden, so this aspect requires special attention or process simplification during implementation.

4. Minor Variability in Other Indicators

A small number of indicators show minor uncertainty among experts, as evidenced by the presence of neutral scores (Score 3). For example, for indicators P11, P23, and P34, 3 experts chose to remain neutral. Additionally, the very minimal occurrence of negative ratings (Score 1) on items such as P5, P6, P7, P9, and P32 (only 1 response each) reflects the presence of small-scale tactical obstacles in the field perceived by a small number of experts regarding the clarity of user needs or the feasibility of implementation at a specific project scale.

Table 7. Inter-Rater Reliability

No	Testing Method	Cofficient Values	Category Interpretation	Statistical Test Characteristics
01	Gwet's AC2	Actual Agreement (Pa) = 0,3505 Chance Probability (Pe) = 0,1588 **Final Value of Gwet's AC2 = 0,2279**	Fair Agreement	Demonstrates better resistance to data distribution bias (kappa paradox) with a low chance probability value.
02	Fleiss' Kappa	Actual Agreement (Pa) = 0,3505 Chance Probability (Pe) 0,3647 **Final Fleiss' Kappa Value = -0,0223**	Poor Agreement	The coefficient value turns negative due to the high probability of chance (Pe > Pa), triggered by data clustering at specific scores (Scores 4 & 5).
03	Intraclass Correlation Coefficient (ICC)	**ICC Single Measures = 0,0804** Lower bound = 0,0211 Upper bound = 0,1833 **ICC Average Measures = 0,1121** Lower bound = 0,0364 Upper bound = 0,2362	Poor Reliability	Using a Two-Way Random Effects Model (α = 0.05). Measures the linear consistency of numerical ratings across raters, rather than merely the agreement on nominal category choices.

Based on the statistical calculations in Table 7, several key findings can be summarized as follows:

1. Analysis of Gwet's AC2 Coefficient

Testing using the Gwet's AC2 method yielded an actual agreement (Pa) value of 0.3505 with a relatively low chance agreement (Pe) value of 0.1588. The combination of these two values resulted in a final Gwet's AC2 coefficient of 0.2279, which falls into the "Fair Agreement" category. These results indicate that, in general, there is a substantive agreement among the raters after proportionally correcting for chance.

2. Analysis of the Fleiss' Kappa Coefficient and the Kappa Paradox

Unlike the results of Gwet's AC2, the test using Fleiss' Kappa yielded a final coefficient value of -0.0223, which is classified as "Poor Agreement." Although the actual agreement value is similar (Pa = 0.3505), the probability of chance in this method surges drastically to 0.3647 (Pe > Pa). This negative coefficient methodologically confirms the occurrence of the Kappa Paradox. This phenomenon is triggered by an uneven data distribution (skewed distribution), where raters' assessments massively cluster in specific score categories, specifically Score 4 (Agree) and Score 5 (Strongly Agree). In the Fleiss' Kappa algorithm, this extreme clustering in one or two categories mathematically inflates the probability of chance to an unrealistic degree, thereby reducing the final coefficient value to a negative number.

3. Analysis of the Intraclass Correlation Coefficient (ICC)

To strengthen the validity of the linearity of the assessment, an Intraclass Correlation Coefficient (ICC) test was conducted using a Two-Way Random Effects Model at a significance level of α = 0.05. The estimation results show: The Single Measures ICC value is 0.0804 with a 95% confidence interval (95% CI) ranging from the lower bound of 0.0211 to the upper bound of 0.1833. The ICC for Average Measures is 0.1121, with a 95% confidence interval ranging from a lower bound of 0.0364 to an upper bound of 0.2362. Both ICC indicators consistently fall into the "Poor Reliability" category. The low ICC coefficients demonstrate that although the experts share a common direction in selecting ordinal categories (the majority agrees), the linear variation in numerical ratings among individual raters still exhibits a fairly wide level of variance.

Conclusion

This study designs an integrated software quality assurance (SQA) framework that combines the Scrum methodology with the Recognize, Scrutinize, Materialize (RSM) design approach as a solution to mitigate quality degradation in multi-project environments. Based on a qualitative and descriptive evaluation of expert assessments, this model structure has proven capable of addressing the research question regarding how to align process controls with shift-left testing. Through the integration of structured testing activities in each iteration, the developed framework has proven functional in mitigating quality risks and priority ambiguities without disrupting the fundamental iterative and incremental characteristics of Scrum.

From both a scientific and practical perspective, the novelty of this research lies in its explicit formulation that integrates SQA, Scrum, and the RSM design method into a single governance model—an area that has not been extensively explored in the agile project management literature. From a scientific perspective, this research expands the application of the Design Science Research (DSR) framework to produce methodological artifacts in complex development environments. From an industrial practice perspective, the results of this design provide high utility in the form of measurable operational guidelines for Product Owners, Scrum Masters, Developers, and QA Engineers to proactively build quality from the early phases and reduce the accumulation of technical debt across projects.

However, this study has limitations because the validation of the new artifact's utility relies on the subjective assessments of nine raters with relatively homogeneous professional backgrounds; therefore, generalizations regarding the model's effectiveness in a broader industrial ecosystem must be considered with caution. Furthermore, the concrete impact of this framework on reducing error density and increasing long-term business value in real-world projects still requires more in-depth empirical evidence. As a recommendation for future research, researchers are advised to conduct pilot projects across various company scales, measure their efficiency impacts using quantitative metrics such as defect density and cycle time, and explore the adaptation of this framework to the characteristics of geographically distributed teams.

References :

Alami, A., & Krancher, O. (2022). How Scrum Adds Value to Achieving Software Quality? *Empirical Software Engineering*, *27*(165). https://doi.org/10.1007/s10664-022-10208-4

Anwar, A. (2025). Software Engineering: A Journey Beyond Code. *Journal of Computer Science and Technology Studies*, *7*(4), 619–627. https://doi.org/10.32996/jcsts

Ariesta, A., Novita Dewi, Y., Ayu Sariasih, F., & Wahyuhening Fibriany, F. (2021). Penerapan Metode Agile Dalam Pengembangan Application Programming Interface System Pada PT XYZ. *Jurnal CoreIT*, *7*(1), 38–42.

Chouhan, R., & Mathur, R. (2012). Role of Software Quality Assurance in Capability Maturity Model Integration. *International Journal of Advanced Research in Computer Engineering & Technology*, *1*(6), 70–77.

Fagarasan, C., Popa, O., Pisla, A., & Cristea, C. (2021). Agile, waterfall and iterative approach in information technology projects. *IOP Conference Series: Materials Science and Engineering*, *1169*(1), 012025. https://doi.org/10.1088/1757-899x/1169/1/012025

Gorla, N., Somers, T. M., & Wong, B. (2010). Organizational impact of system quality, information quality, and service quality. *Journal of Strategic Information Systems*, *19*(3), 207–228. https://doi.org/10.1016/j.jsis.2010.05.001

Haputhanthrige, V., Asghar, I., Saleem, S., & Shamim, S. (2024). The Impact of a Skill-Driven Model on Scrum Teams in Software Projects: A Catalyst for Digital Transformation. *Systems*, *12*(5). https://doi.org/10.3390/systems12050149

Hariyanto. (2020). *Software Quality Assurance pada Perusahaan Pengembang Perangkat Lunak Skala Kecil dan Menengah*. Universitas Islam Indonesia.

Hera, A., Rian, A. Al, Faruque, Md. O., Sizan, M. M. H., Khan, N. A., Rahaman, Md. A., & Ali, M. J. (2024). Leveraging Information Systems for Strategic Management: Enhancing Decision-Making and Organizational Performance. *American Journal of Industrial and Business Management*, *14*(08), 1045–1061. https://doi.org/10.4236/ajibm.2024.148054

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *Design Science in IS Research MIS Quarterly*, *28*(1), 75–105.

Hussain, S. M. A. M., Yalgi, R. S., & Fatima, A. (2025). Comprehensive Review and Adapting the Scrum Framework for Agile Project Management. *International Journal of Creative Research Thoughts (IJCRT)*, *13*(1), 829–836.

Kadenic, M. D., de Jesus Pacheco, D. A., Koumaditis, K., Tjørnehøj, G., & Tambo, T. (2023). Investigating the role of Product Owner in Scrum teams: Differentiation between organisational and individual impacts and opportunities. *Journal of Systems and Software*, *206*, 1–17. https://doi.org/10.1016/j.jss.2023.111841

Khan, F., Kumar, R. L., Kadry, S., & Nam, Y. (2021). The future of software engineering: Visions of 2025 and beyond. *International Journal of Electrical and Computer Engineering*, *11*(4), 3443–3450. https://doi.org/10.11591/ijece.v11i4.pp3443-3450

Klopp, M., Gold-Veerkamp, C., Abke, J., Borgeest, K., Reuter, R., Jahn, S., Mottok, J., Sedelmaier, Y., Lehmann, A., & Landes, D. (2020). Totally Different and yet so Alike: Three Concepts to Use Scrum in Higher Education. *ECSEE '20: European Conference on Software Engineering Education*, 12–21. https://doi.org/10.1145/3396802.3396817

Lubis, M. (2023a). RSM Design Approach Simplified and Extended Version. Kementrian Hukum dan Hak Asasi Manusia Republik Indonesia. Patent 000523874.

Lubis, M. (2023b). RSM Design Step. Kementrian Hukum dan Hak Asasi Manusia Republik Indonesia. Patent 000524020.

Lubis, M. (2023c). RSM User-Oriented & Utility Principles. Kementrian Hukum dan Hak Asasi Manusia Republik Indonesia. Patent 000524056.

Mardoyo, E., Lubis, M., & Ramadani, L. (2024). Analyzing Gen Z Interest in Virtual Reality Learning Environment as a Component of Metaverse Using RSM Design Approach. *Lecture Notes in Networks and Systems*, *803*, 381–392. https://doi.org/10.1007/978-981-99-7569-3_31

Mastaneh, Z., & Mouseli, A. (2023). Holistic View on Information Systems as Logistic in Health Sector. *Evidence Based Health Policy, Management and Economics*, *7*(1), 71–81. https://doi.org/10.18502/jebhpme.v7i1.12356

Mateen, A., Jahanzaib, M., & Iqbal, N. (2017). The Role of Quality Assurance in Software Development Projects: Project Failures and Business Performance. *International Journal of Management, IT and Engineering*, *7*(2).

Mortada, M., Ayas, H. M., & Hebig, R. (2020). Why do software teams deviate from scrum? : Reasons and implications. *Proceedings - 2020 IEEE/ACM International Conference on Software and System Processes, ICSSP 2020*, 71–80. https://doi.org/10.1145/3379177.3388899

Sandy, M. D. H., & Lubis, M. (2025). Conceptual Review Forward RSM Design Approach: Integrasi Lean Six Sigma, Sprint dan Extreme Programming. *Electronic Integrated Computer Algorithm Journal*, *2*(2), 61–70. https://doi.org/10.62123/enigma.v2i2.54

Muller, D., Kropp, M., Anslow, C., & Meier, A. (2021). The Effects on Social Support and Work Engagement with Scrum Events. *Proceedings - 2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2021*, 101–104. https://doi.org/10.1109/CHASE52884.2021.00019

Nikolova, Z. (2020). Testing Strategies in an Agile Context. In S. Goericke (Ed.), *The Future of Software Quality Assurance* (pp. 111–121). Springer International Publishing. https://doi.org/10.1007/978-3-030-29509-7_9

Ramin, F., Matthies, C., & Teusner, R. (2020). More than Code: Contributions in Scrum Software Engineering Teams. *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020*, 137–140. https://doi.org/10.1145/3387940.3392241

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide the Definitive Guide to Scrum: The Rules of the Game*. https://scrumguides.org

Shuib, R., & Hassan, adah. (2021). Towards Adopting Software Quality Assurance in Agile Development Methodology. *Turkish Journal of Computer and Mathematics Education*, *12*(3), 2152–2157.

Wambua, A. W., & Maake, B. M. (2022). Characterizing Software Quality Assurance Practices in Kenya. *International Journal of Software Engineering and Computer Systems (IJSECS)*, *8*(1), 22–28. https://doi.org/10.15282/ijsecs.8.1.2022.3.0093