INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING AND COMPUTER SYSTEMS
E-ISSN: 2180-0650
VOLUME 12, ISSUE 1, 2026, pp. 1 – 15
DOI: https://doi.org/10.15282/ijsecs.12.1.2026.1.0147

RESEARCH ARTICLE

Reconnaissance of the Interdependency Among Requirement Clarity, Sprint Efficiency, and Software Quality: A Conceptual Framework Approach

Syed Shabeeb Raza[1]*, Khalid Mahboob[2], Mustafa Ahmed Khan[3]

[1,2,3]Department of Computer Science, Institute of Business Management, Karachi, Pakistan

ABSTRACT -. This study has developed a theoretical model capable of evaluating the relationships among requirement clarity, sprint efficiency, and software quality in Agile software development. Although Agile adopts a highly flexible and continuous delivery approach, requirement ambiguity remains one of the major issues that negatively impacts both sprint performance and product quality. Grounded in information processing theory, cognitive load theory, and systems theory, this research reveals how unclear or insufficient requirements lead to increased defect rates, reduced sprint velocity, and diminished planning efficiency. Furthermore, the model establishes a set of metrics to support outcome measurement, including the Requirement Volatility Index (RVI), Sprint Velocity (SV), Defect Density (DD), and Quality Index (QI). From a practical perspective, this study affirms that requirement validation processes, stakeholder alignment, and continuous feedback are essential to Agile success. Overall, this research develops a theoretically and empirically grounded model that assists Agile practitioners, project managers, and quality engineers in anticipating and managing requirements-based risks, while also serving as a tool for real-time quality assurance during sprints.

ARTICLE HISTORY
Received : 30 June 2025
Revised : 27 January 2026
Accepted : 27 April 2026
Published : 30 April 2026

KEYWORDS
Requirement Clarity
Agile Software Development
Sprint Efficiency
Software Quality Assurance (SQA)

1.0 INTRODUCTION

In the past few years, Agile methodologies have been the main approach in the software development area, and it is all due to the rapid and reliable software delivery, which was the demand. Agile methods primarily focus on flexibility, customer cooperation, and incremental delivery, which makes it possible for the teams to change but at the same time guarantee the delivery of the product. Such a sprint culture also calls for very clear, well-defined requirements, and especially software requirements. The requirements are the ground upon which the sprint's direct development and quality assurance work is built. If the inputs are unclear or incomplete, the sprint is quickly affected. Thus, the elicitation and validation of requirements must be very accurate and, at the same time, not only a matter of following the procedure but also being crucial to the success of projects and the quality of software in Agile environments [1][2].

While the iterative method and focus on Agile communication have been prioritized, the difficulty of requirement uncertainty is still shown by research. Normally, it is the case that developers are left to figure out the high-level business requirements that have not been properly detailed, documented, or verified by the stakeholders. This mismatching rolls in more assumptions, gaps in implementation, and more rework when the expectations fail. In Agile sprints, when time is short and the quality is to be maintained at all costs, uncertainty about the requirements directly leads to high defect rates, low sprint velocity, and low team morale [3]. Mature Agile teams typically experience changes in requirements during the sprint or right at the beginning that are hard to manage. The challenges eventually bring down the whole Quality Assurance (QA) process as they result in a more reactive than a proactive approach to defect detection [4]. The present paper undertakes a theoretical and reconnaissance-based investigation of the impact of requirement ambiguity on software quality and sprint productivity The study does not try to bring in any new empirical evidence; rather, it fuses the current ones to map out the link between unclear requirements and the performance of Agile teams. Besides, the article provides an overview of the (QA) practices, especially in the areas of requirements validation and test planning, which could possibly alleviate the problem. Agile practices promote teamwork and adaptability, although there is still an ongoing problem in that no formal procedures exist to check the clarity of requirements prior to the start of a sprint [5][6].

Primarily, this study focuses on three main goals. First, to examine the impact of requirement ambiguity on software quality in Agile sprints. Second, to explore the relationship between incomplete requirements and sprint results, particularly regarding defects and delays. Third, the Software Quality Assurance (SQA) activities at different organizational levels that can be applied to mitigate the adverse impacts of ambiguous requirements must be determined and analyzed. Aligning with these goals, this research addresses three research questions:

Research Question (RQ) 1: How does ambiguity in requirements influence software quality in Agile development?

Research Question (RQ) 2: How does requirement incompleteness relate to defect density or sprint interruption?

***CORRESPONDING AUTHOR | Syed Shabeeb Raza** | ✉ shabeeb.raza@iobm.edu.pk

Research Question (RQ) 3: Which assurance strategies work best in avoiding the effects of unclear or changing requirements in Agile projects?

Hence, this research investigates prevailing frameworks, like requirement verification methods and stakeholder-developer alignment models, and examines their potential to reduce these persistent problems. The remaining sections of the paper are organized as follows. The next section is focused on a literature review, followed by the theoretical foundation in section three. The conceptual framework design is described in section four, followed by the discussion and implications in section five. The conclusion is presented in the final section.

2.0 RELATED WORKS

It is known that the clarity of requirements is a significant contributing factor to success in software engineering projects. Ambiguities often cause rework, scope creep, and delivery delays. In Agile, sprint efficiency depends on how well user stories are defined; undefined requirements will disturb the team's velocity and sprint planning. This review highlights the need for such a framework to understand how requirement clarity affects sprint efficiency and the software's accuracy in Agile environments.

Agile Software Development literature emphasizes the importance of precise requirements, efficiently executed sprints, and high software quality. For example, a study in [7] demonstrated that ambiguity in requirements limits sprint planning, thereby limiting product quality. However, it focuses on requirement ambiguity as an independent single variable without studying cascading effects on team velocity and defect density in successive sprint cycles. Where the authors suggest mitigation strategies, they do not indicate any quantitative metrics to measure the improvement, thus limiting practical implementation guidance. While fast-moving Agile environments find effective integration of Quality Assurance into all Sprint activities challenging, researchers have so far considered only requirement clarity, sprint efficiency, and software quality in silos. Yet, an integrated model considering their interrelations is still lacking. The reviewed existing frameworks discuss continuous improvement and built-in quality, but do not pursue these dynamics coherently. This review identifies the need for a common conceptual framework to provide further insight into how these factors influence Agile success.

Requirement Engineering (RE) is the basis for major industrial-scale software projects that represent success [8]. Imprecise specifications are among the main reasons of exceeded budgets, later endings, and the whole project setbacks. Demand problems are sometimes caused by unclear requirements, the lack of good elicitation, poor traceability, and inconsistent documentation. Besides, the authors suggest improvements in processes and role changes. Nevertheless, this framework has not included metrics integration at the sprint level like velocity and defect density. Furthermore, the research does not explain how these RE problems become apparent specifically in Agile iterations, which are time-boxed, where fast feedback loops call for more dynamic requirement validation mechanisms than the ones supported by traditional waterfall methods. Previous research has provided only partial answers regarding the need for guidelines, frameworks, and role reassignments, however, there always existed a continuous lack of comprehensive process coverage The use of Agile methodologies, besides being flexible, still suffers a lot from the turmoil caused by changing requirements. In addition, the whole process is further complicated by technical interdependencies and poor management of non-functional requirements. The researcher points out that there is a crucial lack of a scalable RE framework validation that would be able to systematically and cost-effectively cope with the issues mentioned above, and consequently, this way, it would be an important contributor to the success of the large-scale software industry.

Likewise, the paper in [9] mentioned the increased importance of AI-based decision support systems (DSSs) in the Agile Software Project Management (ASPM) environment. The traditional methods and tools mentioned above for project management are incapable of properly assessing risk, effectively redistributing resources based on the dynamic needs, or performing the tasks according to the newly set work priorities. The authors back up their assertions with real-world cases by presenting that machine learning (ML) models can forecast sprint results with an accuracy rate of 78%. Nevertheless, their method is severely limited: it needs an enormous amount of historical data, a minimum of 20 sprints, which makes it impractical for new teams or projects. Besides that, the authors do not discuss how AI recommendations are integrated with human decision-making in sprint planning sessions, which may create friction between data-driven insights and team autonomy, one of the key Agile principles. Contemporary research utilizes machine learning and predictive analytics to improve sprint effectiveness, backlog, and defect management. Exploration on DSS has identified and discussed the gaps in proactive decision-making capabilities in contemporary Agile environments, particularly for distributed Agile teams that require real-time coordination. The present study mentions how AI-based DSS can bridge the gap because it joins quantitative data with qualitative insight in order to support real-time, data-driven decision-making in Agile settings. Agile software development stresses flexibility and working with customers, but can be hindered by several hurdles.

An investigation in [10] has identified barriers to overcome that are considered essential: resistance to change, lack of training and knowledge of Agile practices, lack of stakeholder engagement, or issues with scaling Agile models in a large organization. This study used a qualitative approach by interviewing 45 practitioners at 15 organizations. Although this will give rich contextual insights, such a study faces self-reporting bias, and generalizability is limited as all respondents came from North American companies. The paper also does not distinguish between challenges relating to initial adoption and issues encountered by mature Agile teams; it considers all obstacles equivalent hindrances. The study also articulates

key managerial actions, including supportive leadership, training, and structured transition planning to support Agile adoption. Agile frameworks (Scrum, Kanban, and SAFe) generally outperform traditional project management approaches in productivity and stakeholder satisfaction under certain conditions. However, typical success varies among industries; even within the same industry, a company's maturity should inform specific steps to influence changes to processes, leadership, stakeholder engagement, and continuous improvement, and fit for the organizational culture. Research indicates a common theme around contextualization, support from leadership, and some action plan related to the transition to Agile.

Agile Project Management (APM) and Scrum methodologies are becoming increasingly popular because of their ability to increase product development speed and improve adaptability [11]. APM supports incremental progress, is built around collaboration with stakeholders, and allows for changes in direction when market conditions change, to respond to market needs quickly. Scrum provides a framework for implementing incremental progress through time-constrained (short) and defined-scope (incremental) releases (sprints) that can be developed, iterated, and finalized to ensure timely delivery. Literature suggests that the joint application of APM and Scrum would facilitate time-to-market, team collaboration, and an end product meeting customer expectation. Comparative analysis, however, proves that the benefits are not universal. Companies with hierarchical culture and waterfall legacy systems resist implementation - 40% reported reduced productivity within the first six months of implementation. This study does not indicate a framework that could be applied to identify the appropriate organizational contexts where APM adoption is to be carried out, nor the transition strategies based on different levels of maturity of organizations. Researchers have also pointed out that fixed-duration iteration often does not allow for the complexity of the tasks and variations associated with projects. The authors, rooted in task complexity, suggest variable sprint durations: two weeks are, in principle, too little for architectural refactoring and too much for simple bug fixes. Their theoretically sound Multi-Attribute Utility Theory (MAUT)-based optimization model assumes perfect up-front information on task complexity- an unrealistic assumption in Agile settings, where uncertainty is inherent. Moreover, applying variable sprint durations contradicts the Agile principle of establishing a sustainable rhythm and thus may badly affect the stability of team velocity measurements and stakeholder expectations. Analysis has been done and the findings keep pointing to the fact that cross-functional collaboration, iterative cycles, and feedback loops are the key factors that make product launches successful. APM and Scrum bring several advantages, however, the whole industry is still struggling to adopt them in their structure or processes owing to the reasons like fear of change, lack of experience, and scarcity of Agile-Scrum expert resources having high demand.

Agile methods, with Scrum as the most widely used, make use of fixed-length sprints. Nevertheless, [12] has conducted a recent study which casts doubt on the inflexibility and limitations associated with the use of fixed timelines. Some researchers have suggested that fixed-duration iterations often cannot support the complexity associated with tasks and project variations. The authors suggest varying the duration of sprints based on the complexity of the tasks. In their opinion, a two-week sprint is too short for architectural refactoring and too long for bug fixes. Their underlying MAUT-based optimization model, which is theoretically valid, assumes perfect information in advance on the complexity of tasks, which is not the case in Agile environments characterized by uncertainty. Most importantly, the different lengths of sprints would break the Agile principle of creating a sustainable rhythm, which could lead to unreliable team velocity measurements and stakeholder expectations. Once more, researchers are unanimous that flexibility should be the focus rather than recommending variable sprint lengths since project workloads and the intensity of resources and support vary. Some studies have shown how mathematical techniques can be applied to find the optimal sprint length using cost and performance parameters of tasks, referring to MAUT. While widely practiced, only a few research efforts have proposed a formal model for dynamic sprint planning, paving the way to develop adaptive and data-driven frameworks toward better sprint efficiency. Table 1 presents the comparative analysis of existing frameworks and highlights the novelty of the proposed study.

Table 1. Comparative Analysis: Framework Contributions and Novelty

Study	Year	Primary Focus	Theoretical Base	Metrics Quantified	Scope / Level	Key Limitation
Hoy & Xu [5]	2023	Requirement Engineering Challenges in Agile	Systematic Literature Review; Process-oriented	qualitative mitigation practices	Team / Project level	Handles RE challenges qualitatively; short of quantitative integration of how requirement ambiguity cascades to defect density and velocity degradation; addresses RE as isolated problem
Moham mad & Kollama na [2]	2024	Causes and Mitigation Practices of Requirement Volatility	Empirical qualitative interviews; practitioner-centric	RV causes identified; mitigation practices described	Team level	Concentrates on RV as independent variable; no mathematical model for relating RV to downstream sprint performance; no

						composite quality measurement
Ilays et al. [3]	2024	Quality of Requirement and Testing Process in Agile	Empirical mixed-method study; QA-centric	Defect metrics, quality indicators, requirement quality	Team / Project level	Independent examination of requirement quality and testing; reactive strategy (after-sprint measurement); no predictive warning system; no cognitive or information processing viewpoint
Siewert et al. [40]	2025	Impact, advantages and limitations of implementing Agile in developing physical products	Exploratory empirical research; manufacturing/R&D interviews; implementation of agile principles in areas other than software	Portfolio-level metrics: Flow, Competency, Outcomes	Enterprise / Product-development level	Not entirely enterprise scale / portfolio level; fewer hard metrics around defect density, requirement clarity, or schedule variance; less sprint cycle focus as in software; might not generalize to pure software situations
Pham & Neuman n [33]	2024	How to Measure Performance in Agile Software Development?	Mixed-method study (interviews, focus groups); empirical repository of metrics	Velocity, Lead Time, Test Coverage, Custom Metrics; repository-based approach	Team / Project level	Identifies the performance measures; but without theoretical integration; regards metrics as autonomous enumeration; fails to explain causal dependencies; no aggregate scoring procedure; lacks information processing or cognitive load viewpoint
Our proposed study	2025	Interdependency Among Requirement Clarity, Sprint Efficiency, and Software Quality	Information Processing Theory; Cognitive Load Theory; Agile Principles; Systems Thinking	RVI (Rc/Rt) with risk bands; Sprint Velocity (Points/Sprints); Schedule Variance (EV–PV); Defect Density (Defects/KLOC) ; Composite Quality Index QI=(Ds×Rf)/Ut	Team-level sprint cycle (actionable at sprint planning and execution)	Addresses the gaps in all prior frameworks

3.0 THEORETICAL FOUNDATION

This research uses several theoretical frameworks to account for how requirement uncertainty affects SQA in Agile development. Theories like information processing theory, cognitive load theory, agile methodology principles, and systems thinking are used as analytical frames to examine stakeholder and developer misalignment and quality decline systematically. Information Processing Theory addresses how individuals process, store, and respond to information. In agile, there is fast and continuous communication; hence, IPT is very relevant in an agile environment. Due to incomplete or imprecise requirements, developers misunderstand the objectives, which causes defects and unwanted sprint outcomes. Mismatches in information flow reduce the team's ability to correctly extract and apply stakeholder expectations [13-15]. Cognitive Load Theory explains how the nature of information developers handle affects their mental workload. Under dynamic or uncertain conditions, the cognitive burden which is unnecessary but heavy results in the devolvement of decision-making abilities of the developers thereby causing not great but low effect of sprints. This burden leads to slower development and also creates an environment that is reactive to quality issues rather than proactive [16-18].

Agile Methodology Principles put into consideration the flexibility, iterative development, and cross-functional teamwork. On the one hand, Agile allows flexible communication; on the other hand, research indicates that Agile is not always able to get rid of requirement uncertainties. Agile teams do not employ very well-defined and common

requirement validation processes; as a result, the definition of user stories and sprint goals is often unclear and poorly articulated [19][20]. Systems Thinking adopts a holistic approach by indicating that not only small issues like miscommunication in requirements would become bigger with each iterative cycle but also all such problems that expand by misunderstanding would be handled together. When the requirements are not correctly understood, it would cause a delay in the production cycle, which would critically affect the quality of the product and the team cohesiveness in the long run [21][22].

4.0 CONCEPTUAL FRAMEWORK DESIGN

The proposed framework will comprehensively investigate the roles of the requirement clarity, the sprint effectiveness, and the software quality on the success of the Agile software development projects. The framework depicts quantifiable entities through the use of mathematical formulas derived from the most recent research, which can verify its strength and scientific practice in influencing and assessing such factors.

Requirement Clarity is identified as a major factor affecting sprint results and software quality in general. When the requirements are not clear or very much changing, it leads to misunderstandings, re-doing of work, and defects. The Requirement Volatility Index (RVI) metric is used to measure the stability of the requirements. This metric calculates the percentage of requirements that have changed during a sprint and is computed as follows:

$$RVI = \frac{R_c}{R_t} \tag{1}$$

The Rc denotes how many requirements have been altered during the sprint while at the same time depicting the number of requirements that were initially planned. RVI is elevated when the requirements are more unstable and less defined, which jeopardizes the success of the sprint and the quality of the software [23] [24].

In the RVI calculation, as illustrated in Figure 1, the process begins with sprint planning and an initial requirement count, then monitors the changes throughout the sprint cycle, and finally, applies the formula RVI = (RC/N) × 100%, where RC represents the changed requirements and N the total of the original requirements. The visualization groups the results of RVI into three risk levels: low volatility (< 20%) indicating the accuracy of the requirements and stability throughout the sprints; medium fluctuations (20-40%) indicating medium risk that needs to be monitored; high volatility (> 40%) indicating serious risks for successful sprinting and software quality that need immediate attention.

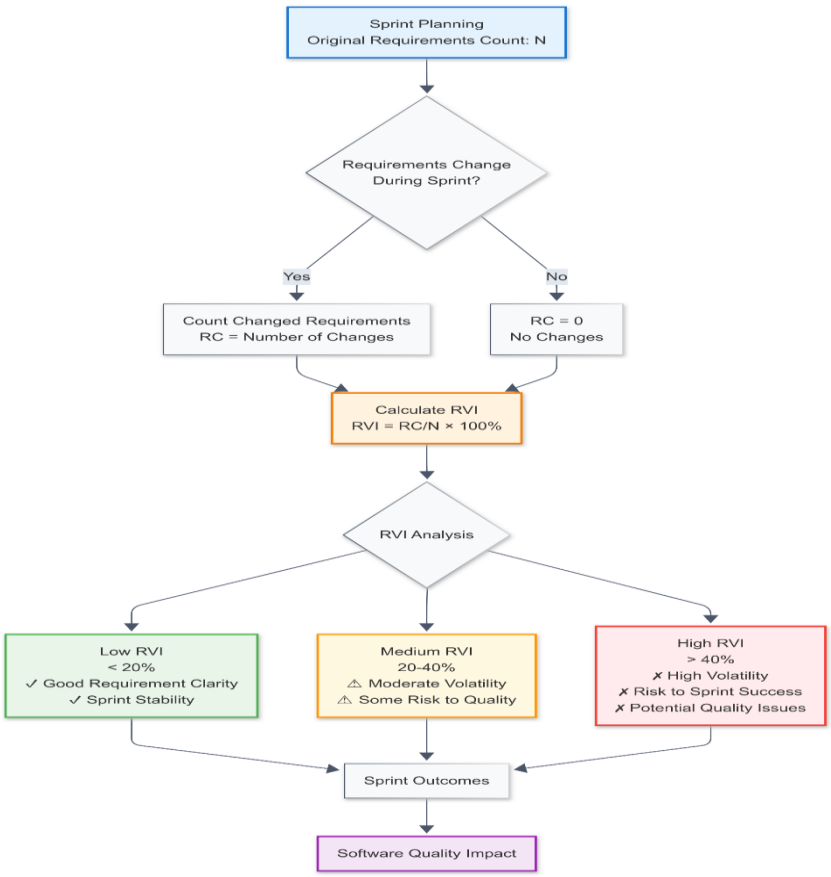

Figure 1. Requirement Volatility Index (RVI) Flow Diagram

Low Volatility (< 20%) indicates considering requirements and stable sprints carefully. Sprint Outcomes: delivery on time and very little rework, high morale, predictable velocity, and the level of software quality standards maintained.

Medium Volatility (20-40%) indicates medium risk that is requiring monitoring. Sprint Outcomes: there might be a delay in deliverables, moderate rework efforts, and the appearance of technical debt, and the team velocity might be oscillating, plus there is a need for stakeholders to be re-engaged regarding the clarity of requirements.

High Volatility (> 40%) indicates about serious risks to the success of the sprint and the quality of the software that require prompt action. Sprint Outcomes: a big disruption in the sprint with missed deadlines, heavy rework of 30-50% of sprint capacity, software quality degradation with high defect rates, team morale suffering along with the risk of burnout, scope creep jeopardizing the projects' viability, and an urgent need for interventions aimed at stabilizing requirements.

Transitioning to Sprint Efficiency is essential to measure how well the development team performs planned work in sprint cycles. One of the most used metrics is Sprint Velocity which is used to determine the average work done in a sprint with regard to story points. The calculation is as follows:

$$Velocity = \frac{Total\ Story\ Points\ Completed}{Number\ of\ Sprints} \tag{2}$$

The above formula helps project managers to assess the delivery capacity of the team and to spot trends that might signify a shift in productivity or the occurrence of a bottleneck [25][26].

Figure 2 shows how teams are tracking their Sprint Velocity, which is in fact the gauging of the amount of work that a team is able to accomplish in each sprint at a reasonable level. This is the practical way of agile that is based on its core value of learning from practice and not theorizing through planning.

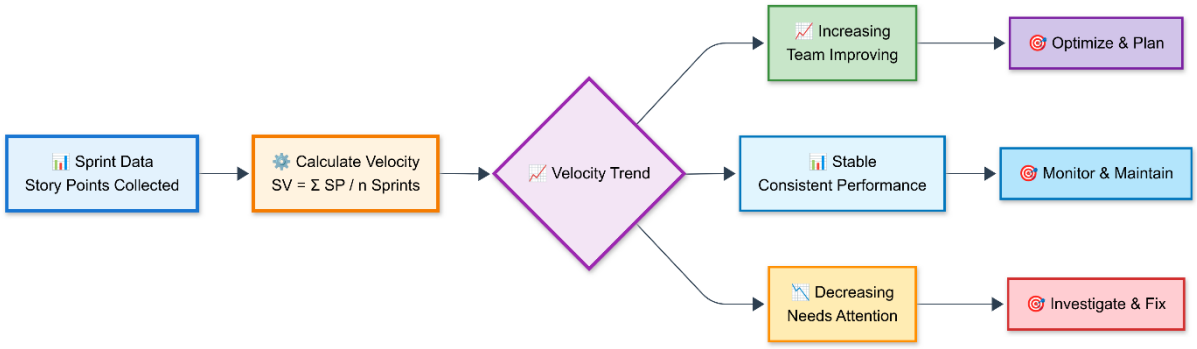

Figure 2. Sprint Velocity Measurement Process

What Sprint Velocity Really Measures: To put it simply, velocity just answers the question: "How much can we really get done?" The process starts off by collecting story points, those estimates that the teams allocate to user stories during their planning meetings—over a number of sprints. By applying the formula:

$$SV = \Sigma\ Story\ Points\ /\ Number\ of\ Sprints \tag{3}$$

Throughputs are then calculated by the teams in an average way. However, the point is that the specific number is not as significant as the trend which is being observed over time. A team with a velocity of 30 is not "better" than another with a velocity of 20 - what matters is whether that number is going up, staying the same, or going down.

4.1 Understanding Velocity Trends

Increasing Velocity: An increase in sprint velocity generally indicates improved team productivity and process efficiency. This may result from better team collaboration, improved familiarity with project requirements, and the removal of development bottlenecks. Consistently increasing velocity may enable teams to handle more complex tasks in future sprints.

Stable Velocity: Stable velocity is often considered an ideal condition in Agile development, as it reflects consistent team performance and sustainable workload management. This stability enables more accurate sprint planning and improves stakeholder confidence in project delivery timelines.

Slowing Velocity: A decline in sprint velocity may indicate underlying performance issues that require further investigation. Possible causes include technical debt, increased task complexity, resource limitations, unclear requirements, process inefficiencies, or dependency-related delays. Continuous monitoring of declining velocity is important to prevent further disruptions to project performance.

4.2 Linking Velocity to Quality: Why Defect Density is Important

Velocity alone may not accurately reflect overall team performance, as a team may deliver features rapidly while compromising software quality. Therefore, Defect Density is used to evaluate code quality by measuring the number of defects relative to the size of the developed code.

$$Defect\ Density\ =\ Total\ Defects\ /\ KLOC \tag{4}$$

This metric calculates the number of defects per thousand lines of code (KLOC) developed. Normalizing defects by code size enables meaningful comparisons across projects of different scales. For example, identifying 50 defects in 1,000 lines of code represents a more severe quality issue than identifying the same number of defects in 100,000 lines of code. Therefore, defect density provides a standardized measure of software quality.

In general, lower defect density indicates higher software quality. A defect density of fewer than 5 defects per KLOC may reflect effective testing practices, efficient code reviews, and well-managed development processes. In contrast, defect density exceeding 10 defects per KLOC may indicate inadequate testing, rushed development activities, or excessive code complexity that requires refactoring.

High velocity combined with low defect density reflects efficient software delivery while maintaining quality standards. High velocity with high defect density may indicate that development speed is being prioritized at the expense of quality. Low velocity with low defect density may suggest conservative development practices, while low velocity with high defect density reflects significant performance and quality concerns that require corrective actions.

The visualization shown in Figure 3 is the Quality Index (QI) model, which is a synthetic metric that combines various quality dimensions in agile development. It is an embodiment of continuous improvement theory and quality management and expresses quality assurance in terms of measurable indicators.

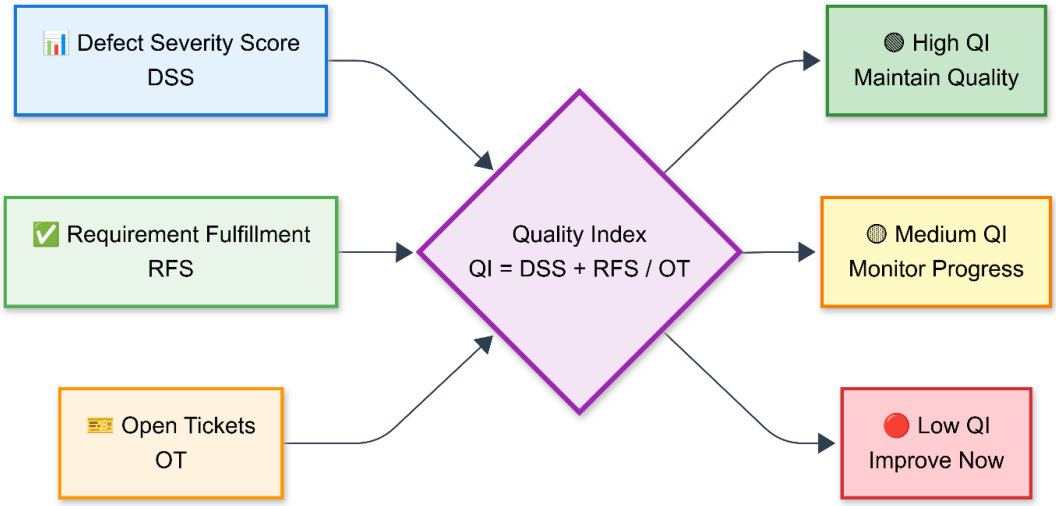

Figure 3. Defect Density Quality Measurement

The Quality Index combines three essential elements:

Assesses the impact of defects with weights reflecting severity levels, consistent with the notion that not all defects have the same influence on the quality of the product.

Requirement Fulfillment Score (RFS): Monitors the degree of compliance of delivered functionalities with the specified requirements, thereby corroborating the principle of agile that "the customer must always be satisfied" through the delivery of value in sync with customer requirements.

Open Tickets (OT): Captures outstanding issues, an inverse quality measure

The formula QI = (DSS + RFS) / OT provides a quality measure that is balanced and higher values indicate better quality. This is consistent with the Goal-Question-Metric (GQM) paradigm, which connects strategic quality objectives to measurable results.

Theoretical Foundation of Defect Density: While mere defect counts are very basic methods counting defects, defect density normalizes quality measurement by the size of the codebase (defects per KLOC - thousand lines of code), which allows for meaningful comparisons of projects with different sizes. The metric is based on software reliability theory, where the number of defects indicates the maturity of the code and the effectiveness of the development process. Industry benchmarks set limits:

High QI (Low defect density <5/KLOC): Mature code signifies maintenance level monitoring.

Medium QI (5-10 defects/KLOC): Moderate technical debt is the implication that progress should be monitored actively.

Low QI (>10 defects/KLOC): Severe quality issues with need for urgent addressing are the implication.

Sprint Management and Schedule Variance: Supporting quality measurements, monitoring schedule adherence also helps in identifying the deviations in velocity which can undermine the quality. The Schedule Variance formula:

$$SV = EV - PV \tag{5}$$

where EV (Earned Value) is the work done and PV (Planned Value) is the planned work, is able to issue early warning signs. This is in line to Earned Value Management (EVM) theory that covers scope, schedule, and cost factors. A positive SV signifies that the sprint is ahead of schedule while a negative SV indicates that there are delays that might lead to quality being compromised through hastiness in development or reduced testing time [27][28].

Velocity Trends Integration: The various metrics when grouped together create an end-to-end quality-schedule feedback loop. A decline in velocity along with the rise in defect density typically indicates the buildup of technical debt or problems with the capacity of the team these are the theoretical constructs of agile empiricism that concentrate on the iterations of the inspect-and-adapt loop. Tracking these trends over iterations gives teams the chances to differentiate short-term noise from the decline of the quality that is systematic, thereby triggering preventive measures that are in line with continuous improvement.

The diagram of states in Figure 4 presents the process of monitoring Schedule Variance as a deterministic method for early detection of sprint management delays. It shows the transition from data acquisition to the deduction of Schedule Variance using the formula SV = EV - PV, with EV denoting Earned Value and PV denoting Planned Value. The procedure splits into three outcome states: positive SV indicating that the project is ahead of schedule and thus, optimizing actions will be initiated, zero SV representing that the project is on schedule and therefore, maintenance activities will be performed, and finally, negative SV signifying that the project has fallen behind schedule and corrective measures will be taken which may include identifying bottlenecks, redistributing resources and modifying scope to avoid risks related to delivery and quality.

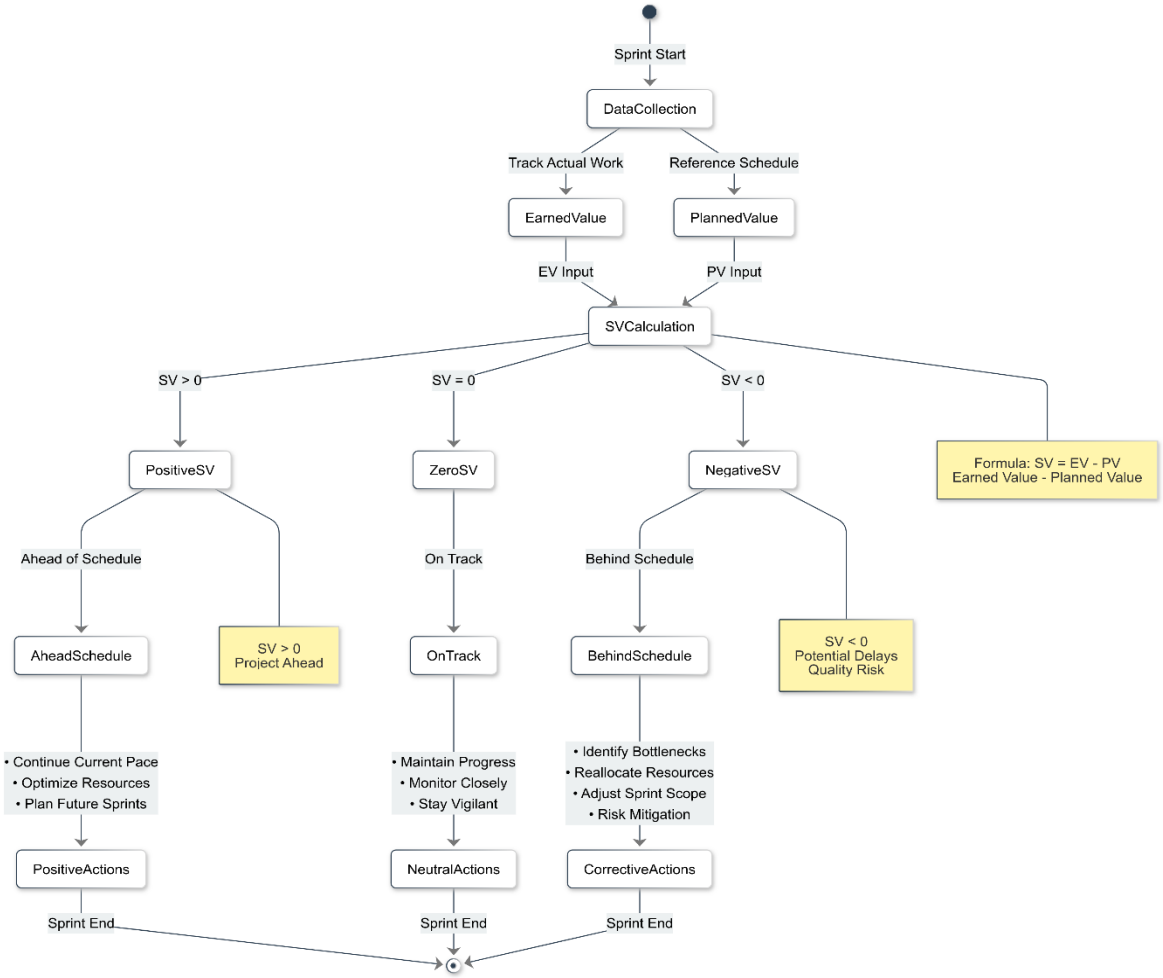

Figure 4. Schedule Variance (SV) Monitoring System

Finally, as an overall quality picture, a Quality Index can be derived from various quality determinants that are together represented in one composite score:

$$QI = \frac{D_s \times R_f}{U_t} \tag{6}$$

D_s is the Defect Severity Score, Rf is the Requirement Fulfillment Score, which indicates requirements' satisfaction, and Ut is the number of open tickets. This index supports tracking defect impact and requirement satisfaction, informing quality improvement activities [29][30].

This flowchart in Figure 5 introduces the Quality Index as an all-encompassing measure of quality that integrates several quality-associated variables into one composite score. Three input elements feed into the middle calculation in the diagram: DSS, capturing weighted defect effect; RFS, capturing requirement levels of satisfaction; and OT, capturing open issues. The formula QI = (DSS + RFS) / OT presents a combined measurement quality score which is then categorized into three levels of high QI scores as being excellent which requires less frequent maintenance, medium QI scores as being tolerable and needing further monitoring, and low QI scores as being unacceptable hence the need for immediate drastic action.

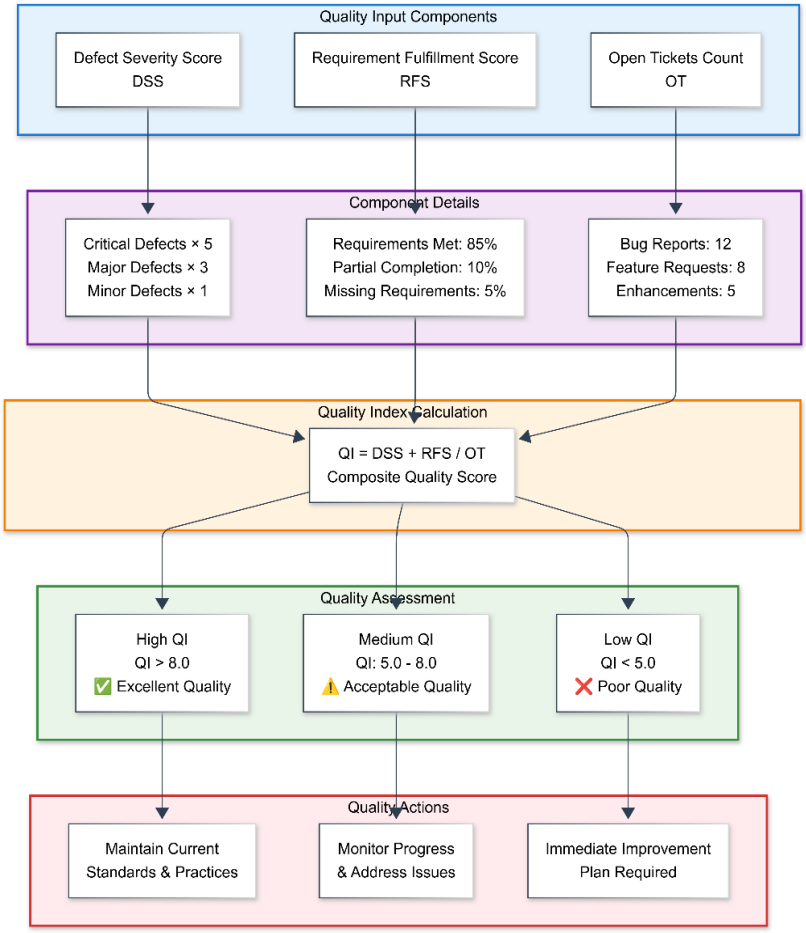

Figure 5. Quality Index (QI) Composite Measurement

Figure 6 illustrates the proposed conceptual framework for Agile software development quality metrics. Consequently, this framework is built on the premise of a four-layer approach, with each layer reinforcing the previous one in the pursuit of measurable quality assurance in agile projects. Theoretical Foundation Layer is the lowest level of the framework, which consists of four significant viewpoints: Information Processing Theory gives cognitive models to understand the development processes; Cognitive Load Theory provides guidance on how to handle mental workload in team situations; Agile Methodology reveals the basic values and iterative methods of agile work; and Systems Thinking encourages comprehensive, interactive methods to solve complicated software problems. The Core Concept Layer, under this umbrella, summarizes and represents the operational ideas from the mentioned foundations: Requirement Clarity, the customer needs definition and comprehension; Sprint Efficiency, the agile team's productivity and iteration capacity; and Software Quality, a standard of deliverables. The depicted interactions imply that the process of clarifying requirements benefits sprint productivity, which again benefits the software quality and the rate of defects.

All these core concepts are indicated in the Metrics Layer, where for each concept quantitative indicators are defined: Defect Density (Defects per KLOC) for software quality, Velocity (Points per Sprint) for team throughput, Schedule Variance (EV–PV) for temporal adherence, Requirement Volatility Index (RVI; Rc/Rt) for stability of requirements, and Quality Index (Ds×Rf/Ut) for holistic quality evaluation. These metrics are direct indicators of the level of implementation and the outcome of the core concepts. Below this is the Implementation Layer where the required concepts and measurements appear in the shape of agile processes: Sprint Planning, Development Execution, Quality Assurance, and Continuous Improvement, etc. The feedback and guidance arrows in the diagram represent the dynamic iterative improvement cycle between the areas of implementation, metrics, and core concepts and the adaptation and empirical nature of agile practice.

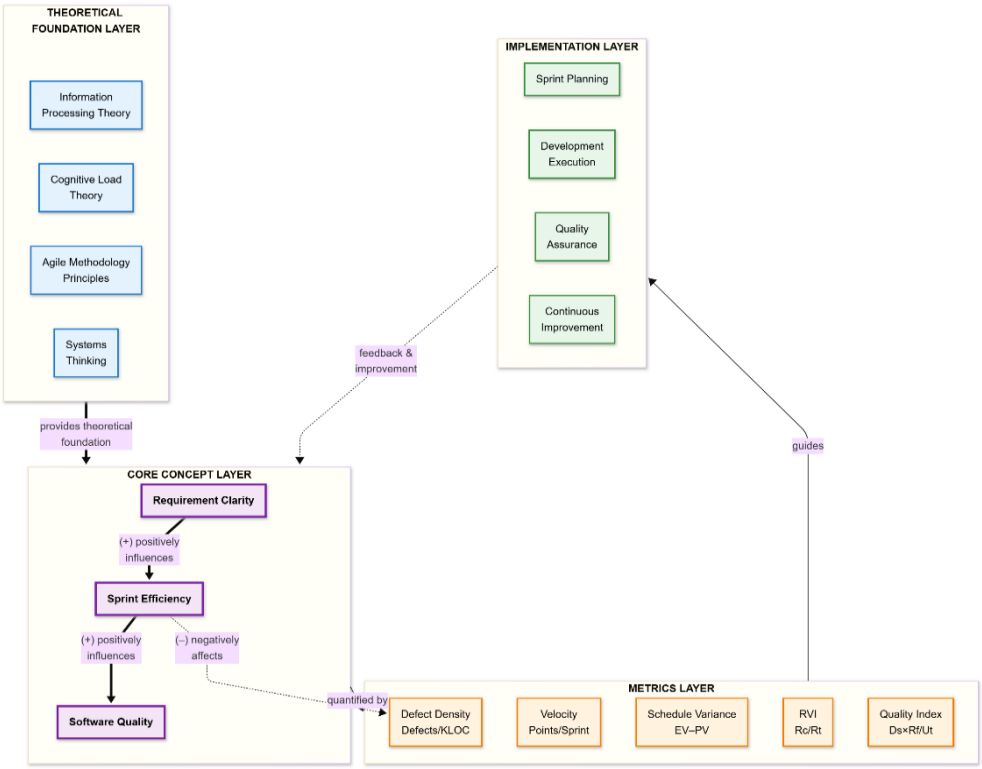

Figure 6: Conceptual Framework for Agile Software Development Quality Metrics

5.0 DISCUSSION AND IMPLICATIONS

This part of the text presents the theoretical and practical implications of the clarity of requirements and the performance of sprints in the practice of Agile development. The reviewed literature includes studies that have undergone a peer-review process and that present metrics, comparisons, and guidelines for software quality, delivery time, and requirements handling. The following three comparison tables provide a structured framework for viewing findings. Table 2. highlights metrics applied to measure the impact of requirement clarity on the project's predictability and sprint efficiency in Agile. Table 3 reviews frequently utilized sprint and quality metrics that indicate the efficiency of Agile teams' operations and output quality. Table 4 reviews metrics that support early identification of ambiguity and promote better requirement elicitation practices.

The comparative metrics reveal a clear trend across the three dimensions: First, requirement clarity defined operationally as RVI <20%, Stakeholder Alignment Index >75%, and Change Request Frequency <2 per sprint has become a strategic factor rather than just a procedural compliance for project success. Moreover, the predictability of the sprint is directly impacted by the clarity of requirements, as shown by the increasing Sprint Predictability Ratios from 0.65-0.75 (unclear requirements) to 0.85-0.95 (clear requirements), which signifies a 15-30% improvement in delivery precision. Finally, the ripple effect shows that the best teams, with clarity in the requirements as characterized by the above metrics, are able to reduce the number of defects by 40-60%, experience 50-65% fewer disruptions during the sprint, and increase their speed of delivery by 15-25% compared to their baseline performance.

The metrics that are being reviewed not only provide quantifiable evidence but also show that the organized planning and cooperation through improvements during the sprint planning stages can have an impact on three quality measurable outcomes: (1) quality of code proportionate defect density reduction from 8-12 defects/KLOC to 3-5 defects/KLOC; (2) delivery speed increment of story completion rate from 65-75% to 85-95%; and (3) team efficiency velocity fluctuation kept in the range of 10% for consecutive sprints. This shift allows project managers and quality assurance personnel to recognize gaps early, respond appropriately, and develop continuous improvement cycles that are based on empirical data rather than subjective evaluation.

Table 2. Metrics Reflecting the Impact of Requirement Clarity

Metrics	Description	Insight
RVI (Requirement Volatility Index) [31]	Measures requirement volatility. $RVI = \frac{R_c}{R_t}$	High RVI is associated with inconsistent sprint results.
Requirement Understanding Score [32]	Qualitative score from interviews with stakeholders	Qualitative score from interviews with stakeholders.
Sprint Predictability Ratio [33]	Completed versus committed stories ratio	Greater clarity enhances delivery precision.
Change Request Frequency [34]	Frequency of mid-sprint changes requested	Lower frequency is associated with original clarity.
Stakeholder Alignment Index [35]	Level of agreement during requirement reviews	A high index increases sprint planning reliability.

Table 3: Sprint Performance and Quality Metrics

Metrics	Description	Insights
Velocity [32]	Number of Sprints	Increases with repeated cycles and well-defined goals.
Defect Density [36]	Defects per KLOC	Decreases when requirements are understood better.
Code Rework Ratio [34]	Percentage of code reworked after the sprint	A large ratio suggests unstable or ambiguous specs.
Story Completion Rate [37]	% of stories with status "done" at sprint completion	The measure of correctness of sprint estimation.
Defect Removal Efficiency [38]	Defects removed before release / total defects	Linked to accuracy in acceptance criteria.

Table 4: Metrics for Managing Ambiguity and Improving Requirement Engineering

Metrics	Description	Insights
QI (Quality Index) [39]	a list of defect severity and requirement fit	Composite measure for quality and clarity tracking.
UAT Success Rate [32]	User Acceptance Test pass% %	Strong measure for dev output alignment with user needs
Feedback Cycle Time [33]	Time from feedback request to response	Shorter cycles improve planning flexibility.
Ambiguity Detection Count [40]	# of unclear or conflicting requirements	A handy measure for training and knowledge sharing.
Use Case Coverage [36]	% of use cases defined in the sprint covered	Directly relates to product acceptance.

6.0 PRACTICAL IMPLICATION

For Agile teams, sprint-based performance measures, particularly velocity, rate of story completion, and defect density, offer a tangible means for monitoring ongoing improvement. Project managers can make RVI or QI the top priority during sprint planning. At the same time, QA engineers can use defect removal efficiency and UAT pass rate to verify test alignment with business requirements.

Based on the theories cited in the conceptual framework design section, the research questions above have been answered below.

RQ 1: How does ambiguity in requirements influence software quality in Agile development?

Ambiguity in requirements significantly impacts software quality in Agile environments primarily because it indirectly interferes with understanding, planning, and execution. Agile development relies on low documentation and fast iterations, so any uncertainty at the beginning of the process significantly increases problems as long as the project is in development. According to Information Processing Theory, when requirements are ambiguous, the team has difficulty interpreting stakeholder needs correctly, resulting in increased error rates and misinterpreted deliverables [13]. This is supported by the theory of Cognitive Load, which implies that changing requirements mentally burdens developers and lowers their cognitive capability to perform a task optimally [14] [16].

On the other hand, there is also regular empirical evidence showing that imprecise requirements increase the likelihood of defects during and after development. Authors of [7] determined that the vagueness of requirements hinders sprint planning and deteriorates product quality. The usual measurements in such cases, the Defect Density and Requirement Volatility Index (RVI), are generally high, indicative of low-quality control and reactive defect management [32] [33]. Additionally, Systems Thinking allows us to see how small miscommunications add up exponentially from sprint to sprint, striking at the heart of testing, release management, and long-term reliability [21] [22]. The result is a product that can hit its marks but lacks strength and client satisfaction.

RQ 2: How does requirement incompleteness relate to defect density or sprint interruption?

Incompleteness results in high defect density and high interruption rates in sprints. Incomplete requirements can be incomplete user stories, incomplete acceptance criteria, or volatile backlog items, which all introduce uncertainty within the sprint cycle. According to studies such as [8] and [3], missing requirements are the primary cause of rework, cost, and deadline blowouts. Such interferences are generally measured in key metrics. For example, a high Code Rework Ratio and a high frequency of Change Requests within a sprint signal that the requirements are poorly documented [36]. The Defect Density metric typically goes up when missing inputs trigger the creation of incorrect code that cannot satisfy user needs [36].

The Sprint Predictability Ratio- a measure of completed vs. planned stories- is lower, with the growing incompleteness of requirements [35]. Team velocity and team morale also reflect this. According to [4], reacting to uncertainty instead of working in certainty. Procedures such as the Ambiguity Detection Count and the Stakeholder Alignment Index can head off such disruptions, and organizations can take steps toward enhanced flow and better quality in a sprint's output [35][40].

RQ 3: Which assurance strategies work best in avoiding the effects of unclear or changing requirements in Agile projects?

Various SQA approaches are helpful in an Agile context to reduce the impact of uncertain or variable requirements. They are all based on principles of early verification, collaboration with stakeholders, and feedback loops. Works like [5] and [19] have indicated the significance of requirement verification techniques, especially when integrated into the sprint planning process right from the outset. For example, the UAT Success Rate and the Requirement Fulfillment Score are reliable indicators of whether the initial requirements were adequately translated into deliverables [32].

Organizational levels call for better alignment of developers and stakeholders. The Stakeholder Alignment Index has verified that better alignment signifies sprint success and reduced rework [35]. Employing ambiguity detection mechanisms in collaborating sessions can also significantly reduce the risk of misinterpretation [41]. Besides, the quality index measures defect severity, requirement satisfaction, and open tickets, enabling a balanced focus on quality and readability throughout the sprint lifecycle [39]. Finally, integrating Agile with Systems Thinking develops a better understanding of how near-requirements problems may escalate into system-wide failures, as recommended by [22].

Its practicality for real-world usage is underpinned by direct application in Agile software development environments beset by persistent sprint inefficiencies and issues of requirement volatility. By embedding quantifiable measures such as the Requirement Volatility Index (RVI), Sprint Velocity, Defect Density, Schedule Variance, and the Quality Index (QI), this model provides the mechanisms for project managers and Agile practitioners to monitor project health continuously based on quantifiable metrics. For example, high RVI triggers teams immediately to recognize unstable or uncertain requirements, thus providing a starting point for stakeholder reviews or backlog adjustments. Similarly, reducing sprint velocity or increasing defect density offers actionable feedback on team workload balance and code quality assessment. The metrics presented provide real-time performance measurement and facilitate data-driven retrospectives whereby decisions are based on empirical evidence rather than subjective judgment. Moreover, the model promotes predictive agility, representing a team's capability to anticipate risks related to ambiguous requirements and to fix those problems well before they manifest into schedule delays or quality loss. This makes Agile project management a proactive, analytical discipline underpinned by principles of continuous improvement and quantified outcomes.

7.0 CONCLUSION

This research offers a comprehensive conceptual framework that establishes the relationship between theoretical concepts and real-life problems in Agile software development, with particular reference to inter-linkages among the clarity of requirements, efficiency of sprints, and quality of software. The research depicts how the more unclear the requirements are, the larger the adverse effect on software quality and sprint execution efficiency. It shows how theories (Information Processing, Cognitive Load, Agile Principles, and Systems Thinking) combine to make sense of this relationship. Such metrics, RVI, Sprint Velocity, Defect Density, Schedule Variance, and QI, measure the health of Agile projects to bring some quantifiable means of assessing and improving Agile project outcomes. It is established that any requirement validation, engagement of stakeholders, and setting up of quality reviews early will lead to less rework, fewer defects, and the maintenance of high morale among team members. A necessary contribution of this framework is that it provides an avenue for making Agile practitioners, project managers, and quality engineers more proactive, strategically work out in determining requirement-related issues, and encourage a work culture that embraces continuous improvements for the timely delivery of high-quality software products.

ACKNOWLEDGEMENT

This study was not supported by any grants from funding bodies in the public, private, or not-for-profit sectors.

AUTHORS CONTRIBUTION

Syed Shabeeb Raza (Conceptualization; Methodology; Data curation; Writing - original draft; Resources)

Khalid Mahboob (Formal analysis; Visualisation; Writing - review & editing; Supervision)

Mustafa Ahmed Khan (Validation; Investigation; Writing - review & editing)

CONFLICT OF INTEREST

The authors declare no conflicts of interest.

REFERENCES

[1] M. R. Ara, "Improving clarity and completeness in user stories: Insights from a multi-domain analysis with developer feedback," vol. 2, no. ICEIS, pp. 272–279, 2025, doi:10.5220/0013363500003929.

[2] A. Mohammad and J. M. Kollamana, "Causes and mitigation practices of requirement volatility in agile software development," Informatics, vol. 11, no. 1, 2024, doi:10.3390/informatics11010012.

[3] I. Ilays et al., "Towards improving the quality of requirement and testing process in agile software development: An empirical study," Computers, Materials & Continua, vol. 80, no. 3, pp. 3761–3784, 2024, doi:10.32604/cmc.2024.053830.

[4] J. Meckenstock, "Shedding light on the dark side: A systematic literature review of the issues in agile software development methodology use," Journal of Systems and Software, vol. 211, p. 111966, 2024, doi: 10.1016/j.jss.2024.111966.

[5] Z. Hoy and M. Xu, "Agile software requirements engineering challenges-solutions: A conceptual framework from systematic literature review," Information, vol. 14, no. 6, pp. 1–19, 2023, doi:10.3390/info14060322.

[6] E. Bjarnason, M. Unterkalmsteiner, M. Borg, and E. Engström, "A multi-case study of agile requirements engineering and the use of test cases as requirements," Information and Software Technology, vol. 77, pp. 61–79, 2016, doi: 10.1016/j.infsof.2016.03.008.

[7] M. Asif, Z. U. Din, M. I. Khan, and S. Ullah, "Optimization of software quality framework in agile software development," *International Journal of Computational Intelligence in Control*, vol. 14, no. 1, pp. 494–515, Jun. 2022.

[8] H. S. Dar and A. Rahman, "A conceptual framework for reducing requirement engineering challenges in industrial-scale software projects," *IJIST Journal*, Jun. 2024.

[9] S. S. Almalki, "AI-driven decision support systems in agile software project management: Enhancing risk mitigation and resource allocation," *Systems*, vol. 13, no. 3, pp. 1–23, 2025, doi:10.3390/systems13030208.

[10] L. S. Geetha et al., "Challenges and solutions in agile software development: A managerial perspective on implementation practices," *International Journal of Advanced Computer Science and Applications*, vol. 16, no. 3, pp. 748–758, 2025, doi:10.14569/IJACSA.2025.0160374.

[11] E. Dosumu, O. O. George, and C. O. Makata, "Advancing product launch efficiency: A conceptual model integrating agile project management and scrum methodologies," vol. 5, no. 2, pp. 1233–1238, 2025.

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

[13] V. S. Lapshin, Y. I. Rogozov, and S. A. Kucherov, "Method for building an information model specification based on a sensemaking approach to user involvement in the development process," *Journal of King Saud University – Computer and Information Sciences*, vol. 34, no. 7, pp. 4644–4658, 2022, doi:10.1016/j.jksuci.2021.04.016.

[14] D. Helgesson, "Exploring cognitive waste and cognitive load in software development: A grounded theory," pp. 60–73, 2023.

[15] G. Sriraman and S. Raghunathan, "A systems thinking approach to improve sustainability in software engineering: A grounded capability maturity framework," *Sustainability*, vol. 15, no. 11, 2023, doi:10.3390/su15118766.

[16] D. Helgesson, D. Appelquist, and P. Runeson, *A grounded theory of cognitive load drivers in novice agile software development teams*. Association for Computing Machinery, 2021.

[17] P. Belém, R. De Mello, A. S. Vivacqua, and A. Neves De Souza, "Investigating the cognitive load drivers of software evolution activities," *ACM International Conference Proceedings Series*, pp. 342–347, 2023, doi:10.1145/3613372.3613377.

[18] W. Ajayi, O. W. Olakayode, F. Temilade, and M. Gbadegesin, "Implementing quality assurance in an agile software development process," *International Journal of Innovative Science and Research Technology*, vol. 7, no. 9, pp. 1951–1958, 2022.

[19] I. Atoum et al., "Challenges of software requirements quality assurance and validation: A systematic literature review," *IEEE Access*, vol. 9, pp. 137613–137634, 2021, doi:10.1109/ACCESS.2021.3117989.

[20] L. S. Ferreira and F. S. Nobre, "Agile project management under the perspective of dynamic capabilities," *Gestão & Produção*, vol. 29, pp. 1–24, 2022, doi:10.1590/1806-9649-2022V29E3122.

[21] G. L. Dugarte-Peña et al., "Using system dynamics to teach about dependencies, correlation and systemic thinking on software process workflows," *IET Software*, vol. 15, no. 6, pp. 351–364, 2021, doi:10.1049/sfw2.12031.

[22] D. I. Silva and L. K. B. Siriwardana, "Comparative analysis of software quality assurance approaches in development models," 2023.

[23] K. Kharchenko et al., "Forecasting software development costs in scrum iterations using ordinary least squares method," *Technology Audit and Production Reserves*, vol. 4, no. 2, pp. 30–33, 2024, doi:10.15587/2706-5448.2024.310411.

[24] M. Poženel et al., "Agile effort estimation: Comparing the accuracy and efficiency of planning poker, bucket system, and affinity estimation methods," *International Journal of Software Engineering and Knowledge Engineering*, vol. 33, no. 11–12, pp. 1923–1950, 2023, doi:10.1142/S021819402350064X.

[25] R. U. Koralage, "Novel approach to estimate velocity for agile scrum sprint planning," 2023, doi:10.13140/RG.2.2.17595.18721.

[26] S. Sharma, D. Kumar, and M. E. Fayad, "On the sprint length estimation technique in agile software development using planning poker," *International Journal of Agile Systems and Management*, vol. 16, no. 2, pp. 205–225, 2023, doi:10.1504/IJASM.2023.130837.

[27] P. A. Whigham, C. A. Owen, and S. G. MacDonell, "A baseline model for software effort estimation," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 3, 2015, doi:10.1145/2738037.

[28] V. Tawosi, R. Moussa, and F. Sarro, "Agile effort estimation: Have we solved the problem yet? Insights from a replication study," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2677–2697, 2023, doi:10.1109/TSE.2022.3228739.

[29] Y. J. Pérez Castillo et al., "Sprint management in agile approach: Progress and velocity evaluation applying machine learning," *Information*, vol. 15, no. 11, 2024, doi:10.3390/info15110726.

[30] E. Kula et al., "Dynamic prediction of delays in software projects using delay patterns and Bayesian modeling," *Proceedings of ESEC/FSE 2023*, pp. 1012–1023, 2023, doi:10.1145/3611643.3616328.

[31] K. P. Pham and M. Neumann, "How to measure performance in agile software development? A mixed-method study," *Proceedings of SEAA*, pp. 1–8, 2024, doi:10.1109/SEAA64295.2024.00074.

[32] F. Almeida and P. Carneiro, "Performance metrics in scrum software engineering companies," *International Journal of Agile Systems and Management*, vol. 14, no. 2, pp. 205–223, 2021, doi:10.1504/IJASM.2021.118061.

[33] M. Huss et al., "Comparing measured agile software development metrics using an agile model-based software engineering approach versus scrum only," *Software*, vol. 2, no. 3, pp. 310–331, 2023, doi:10.3390/software2030015.

[34] S. Jain, "Comparative analysis of agile frameworks: Evaluating their impact on software development lifecycle efficiency," *Intelligent Systems and Applications*, vol. 2, no. 1, pp. 3126–3137, 2024.

[35] F. Uzomah et al., "A comparative analysis of agile and traditional project management methodologies," vol. 30, no. 1, pp. 5737–5746, 2024, doi:10.53555/kuey.v30i1.9237.

[36] R. Menezes et al., "Metrics in large-scale agile software development: A multivocal literature review," *CIbSE 2024*, pp. 106–120, 2024, doi:10.5753/cibse.2024.28442.

[37] 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.

[38] B. Falah, "Evaluating agile methodologies for software requirements and construction: A SWEBOK-based analysis," vol. 13, no. 2, pp. 174–194, 2025.

[39] 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.

[40] P. Malla, "Analyzing the impact of agile methodologies on software quality and delivery speed: A comparative study," vol. 25, no. 1, pp. 1207–1216, 2025.

[41] L. Siewert, K. Gericke, N. Siller, J. Rusin, and D. Göhlich. Impact, benefits and challenges of agile development: an explorative study on physical products. *Proceedings of the Design Society*. 2025; 5:2879–2888. doi:10.1017/pds.2025.295.