**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 interdependence among requirement clarity, sprint efficiency, and software quality: A conceptual framework approach

**Syed Shabeeb Raza, Khalid Mahboob*, Mustafa Ahmed Khan**

Department of Computer Science, Institute of Business Management, Karachi, Pakistan

**Abstract -**. This study 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 a major issue that negatively impacts both sprint performance and product quality. Grounded in information processing theory, cognitive load theory, and systems theory, this study 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 for agile success. Overall, this study 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*

## 1. Introduction

In recent years, agile methodologies have become the main approach in software development, largely because of the demand for rapid and reliable software delivery. Agile methods focus on flexibility, customer collaboration, and incremental delivery, enabling teams to adapt while ensuring product delivery. This sprint culture also calls for clear, well-defined requirements, especially software requirements. Requirements form the foundation for the sprint's development and quality assurance. If the inputs are unclear or incomplete, the sprint is quickly affected. Thus, requirements elicitation and validation must be accurate and, in agile environments, not only a matter of following procedure but also crucial to project success and software quality [1]–[2]. Although iterative methods and a focus on agile communication have been prioritised, research still shows the difficulty of requirement uncertainty. Often, developers are left to interpret high-level business requirements that stakeholders have not properly detailed, documented, or verified. This mismatch introduces more assumptions, implementation gaps, and rework when expectations are not met. In Agile sprints, when time is short and quality must be maintained at all costs, uncertainty about requirements directly leads to high defect rates, low sprint velocity, and low team morale [3]. Mature Agile teams typically face requirement changes during the sprint or at the start that are difficult to manage. These challenges eventually undermine the entire Quality Assurance (QA) process, resulting in a more reactive than proactive approach to defect detection [4]. This study undertakes a theoretical and reconnaissance-based investigation of how requirement ambiguity affects software quality and sprint productivity. This study does not introduce any new empirical evidence; rather, it synthesises existing evidence to map the link between unclear requirements and agile team performance. In addition, this article provides an overview of practices, especially in requirements validation and test planning, that could help alleviate the problem. Agile practices promote teamwork and adaptability, but a problem remains: no formal procedures exist to check requirement clarity before a sprint begins [5]–[6]. This study primarily focuses on three main goals. First, we examined the impact of requirement ambiguity on software quality in agile sprints. Second, to explore the relationship between incomplete requirements and sprint results, particularly in terms of defects and delays. Third, we identified and analysed Software Quality Assurance (SQA) activities at different organisational levels that can mitigate the adverse impacts of ambiguous requirements. Aligned with these goals, this study addresses the following 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 and sprint interruption?
- Research Question (RQ) 3: Which assurance strategies work best in avoiding the effects of unclear or changing requirements in Agile projects?

Hence, this study investigates prevailing frameworks, such as requirement verification methods and stakeholder-developer alignment models, and examines their potential to reduce these persistent problems. The remainder of this paper is organised as follows. The next section focuses on the literature review, followed by the theoretical foundation in section three. Section four describes the conceptual framework design, followed by the discussion and implications in section five. The conclusion is presented in the final section of this paper.

***CORRESPONDING AUTHOR | Khalid Mahboob** | ✉ khalid.mahboob@iobm.edu.pk
© 2026 The Author(s). Published by Universiti Malaysia Pahang Al-Sultan Abdullah Press. This is an open access article under the CC BY-NC 4.0 license

## 2. Related Works

Clear requirements are a major factor in the success of 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 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 software accuracy in agile environments. The Agile Software Development literature emphasises the importance of precise requirements, efficiently executed sprints, and high software quality. For example, a study in [7] showed that ambiguous requirements limit sprint planning, thereby reducing product quality. However, it treats requirement ambiguity as an independent variable without examining its cascading effects on team velocity and defect density across successive sprint cycles. Although the authors suggest mitigation strategies, they do not provide quantitative metrics to measure improvement, limiting practical implementation guidance. While fast-moving agile environments find integrating Quality Assurance into all Sprint activities challenging, researchers have considered only requirement clarity, sprint efficiency, and software quality in silos. However, an integrated model that considers 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 better understand how these factors influence agile success.

Requirements Engineering (RE) underpins major industrial-scale software projects and is central to their success [8]. Imprecise specifications are among the main reasons for exceeding budgets, late endings, and project setbacks. Demand problems can stem from unclear requirements, poor elicitation, poor traceability, and inconsistent documentation. In addition, the authors suggest process improvements and role changes. However, this framework does not integrate sprint-level metrics, such as velocity and defect density. Furthermore, the research does not explain how these RE problems become apparent in Agile iterations, which are time-boxed and rely on fast feedback loops that call for more dynamic requirement-validation mechanisms than traditional waterfall methods support. Previous research has provided only partial answers on the need for guidelines, frameworks, and role reassignments; however, the literature has long lacked comprehensive process coverage. Although Agile methodologies are flexible, they still suffer significantly from the turmoil caused by changing requirements. In addition, technical interdependencies and poor management of non-functional requirements further complicate the process. The researcher points out a crucial lack of scalable RE framework validation that can systematically and cost-effectively address the issues mentioned above; consequently, 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 cannot properly assess risks, effectively redistribute resources based on dynamic needs, or perform tasks according to newly set work priorities. The authors support their assertions with real-world cases, showing that machine learning (ML) models can predict sprint results with 78% accuracy. Nevertheless, their method is severely limited: it requires a large amount of historical data and at least 20 sprints, making it impractical for new teams or projects. In addition, the authors do not discuss how AI recommendations integrate with human decision-making in sprint planning sessions, which may create friction between data-driven insights and team autonomy, a key agile principle. Contemporary research uses machine learning and predictive analytics to improve sprint effectiveness, backlog management, and defect management. Research on DSS has identified gaps in proactive decision-making in contemporary agile environments, particularly for distributed agile teams that require real-time coordination. This study explains how an AI-based DSS can bridge this gap by combining quantitative data with qualitative insights to support real-time, data-driven decision-making in agile settings. Agile software development stresses flexibility and customer collaboration but can be hindered by several hurdles.

An investigation in [10] identified the following essential barriers to overcome resistance to change, lack of training and knowledge of agile practices, lack of stakeholder engagement, and issues scaling agile models in a large organisation. This study used a qualitative approach by interviewing 45 practitioners from 15 organisations. Although this provides rich contextual insights, the study faces self-reporting bias, and generalisability is limited because all respondents were from North American companies. This study also does not distinguish between challenges related to initial adoption and issues encountered by mature agile teams; it treats all obstacles as equivalent hindrances. This 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 terms of productivity and stakeholder satisfaction under certain conditions. However, success varies across industries; even within the same industry, a company's maturity should inform specific steps to influence changes in processes, leadership, stakeholder engagement, and continuous improvement, and to fit the organisational culture. Research indicates a common theme of contextualisation, leadership support, and action plans related to the transition to Agile.

Agile Project Management (APM) and Scrum methodologies are becoming increasingly popular because they can increase product development speed and improve adaptability [11]. APM supports incremental progress, is built around stakeholder collaboration, and allows changes in direction when market conditions shift to respond quickly to market needs. Scrum provides a framework for implementing incremental progress through time-constrained (short) and defined-scope (incremental) releases (sprints) that can be developed, iterated, and finalised to ensure timely delivery. The literature suggests that applying APM and Scrum together improves time-to-market, team collaboration, and alignment with customer expectations. However, comparative analysis proves that these benefits are not universal. Companies with hierarchical cultures and waterfall legacy systems resist implementation; 40% reported reduced productivity within the first six months. This study does not provide a framework to identify appropriate organisational contexts for APM

adoption, nor transition strategies based on organisations' different maturity levels. Researchers have also pointed out that fixed-duration iterations often do not accommodate the complexity of tasks and the variations associated with projects. Rooted in task complexity, the authors 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 optimisation model assumes perfect up-front information on task complexity, an unrealistic assumption in agile settings, where uncertainty is inherent. Moreover, variable sprint durations contradict the agile principle of establishing a sustainable rhythm and may adversely affect the stability of team velocity measurements and stakeholder expectations. Analysis shows that cross-functional collaboration, iterative cycles, and feedback loops are key factors in successful product launches. APM and Scrum offer several advantages; however, the industry still struggles to adopt them in its structure or processes because of fear of change, lack of experience, and a scarcity of Agile-Scrum experts amid high demand.

Table 1. Comparative analysis: Framework contributions and novelty

| Reference | Primary focus | Theoretical base | Metrics quantified | Scope / level | Key limitation |
|---|---|---|---|---|---|
| Hoy & Xu [5] | 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 |
| Mohammad & Kollamana [2] | 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] | 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] | 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 & Neumann [33] | 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 | 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 |

Agile methods, with Scrum as the most widely used, utilise fixed-length sprints. Nevertheless, [12] conducted a recent study that casts doubt on the inflexibility and limitations associated with fixed timelines. Some researchers have suggested that fixed-duration iterations often cannot support the complexity associated with task and project variations. The authors suggested varying the sprint duration based on task complexity. In their opinion, a two-week sprint is too short for

architectural refactoring and too long for fixing bugs. Their underlying MAUT-based optimisation model, while theoretically valid, assumes perfect advance knowledge of task complexity, which does not hold in agile environments characterised by uncertainty. Most importantly, different sprint lengths would break the agile principle of creating a sustainable rhythm, which could lead to unreliable team velocity measurements and stakeholder expectations. Researchers are unanimous that flexibility should be the focus rather than recommending variable sprint lengths because 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 task cost and performance parameters, referring to MAUT. Although widely practised, few research efforts have proposed a formal model for dynamic sprint planning, paving the way for adaptive, data-driven frameworks that improve sprint efficiency. Table 1 presents a comparative analysis of existing frameworks and highlights the novelty of the proposed study.

## 3. Theoretical Foundation

This study uses several theoretical frameworks to account for how requirement uncertainty affects SQA in Agile development. Theories such as information processing theory, cognitive load theory, agile methodology principles, and systems thinking are used as analytical frames to systematically examine stakeholder and developer misalignment and quality decline. Information Processing Theory addresses how individuals process, store, and respond to information. In agile, communication is fast and continuous; hence, IPT is highly relevant in agile environments. Because requirements are incomplete or imprecise, developers misunderstand the objectives, causing defects and unwanted outcomes. Mismatches in information flow reduce a 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, unnecessary but heavy cognitive burdens reduce developers' decision-making ability, thereby causing a low impact of sprints. This burden slows development and creates an environment that reacts to quality issues rather than proactively preventing them [16]–[18]. Agile methodology principles emphasise flexibility, iterative development, and cross-functional teamwork. On the one hand, Agile enables flexible communication; on the other, research indicates that Agile cannot always eliminate requirement uncertainties. Agile teams do not use well-defined, shared requirement-validation processes; as a result, user stories and sprint goals are often unclear and poorly articulated [19][20]. Systems Thinking adopts a holistic approach by indicating that not only do small issues, such as miscommunication in requirements, become bigger with each iterative cycle, but also that all such problems that expand through misunderstanding are handled together. When requirements are not correctly understood, it can delay the production cycle, which can critically affect product quality and team cohesiveness in the long run [21]–[22].

## 4. Conceptual Framework Design

The proposed framework comprehensively investigates the roles of requirement clarity, sprint effectiveness, and software quality in the success of agile software development projects. The framework depicts quantifiable entities through mathematical formulas derived from the most recent research, which can verify its strength and scientific rigor in influencing and assessing these factors. Requirement Clarity is identified as a major factor affecting sprint results and overall software quality. When requirements are unclear or frequently change, misunderstandings, rework, and defects occur. The Requirement Volatility Index (RVI) metric measures the stability of requirements. This metric calculates the percentage of requirements that have changed during a sprint and is computed as follows:

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

where $R_c$ denotes the number of requirements that have been altered during the sprint while simultaneously depicting the number of requirements that were initially planned. RVI is elevated when requirements are more unstable and less defined, which jeopardises sprint success and software quality [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 number of original requirements. The visualisation grouped 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 a medium risk that needs to be monitored; and high volatility (> 40%), indicating serious risks for successful sprinting and software quality that need immediate attention.

Low Volatility (< 20%) indicates that the team considers requirements carefully and sprints are stable. 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 a medium risk that requires monitoring. Sprint Outcomes: There may be delays in deliverables, moderate rework, the emergence of technical debt, and fluctuating team velocity, plus a need to re-engage stakeholders on requirement clarity. High Volatility (> 40%) indicates serious risks to sprint success and software quality that require prompt action. Sprint Outcomes: a major 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 jeopardising the project's viability, and an urgent need for interventions aimed at stabilising requirements. Transitioning to Sprint Efficiency is essential for measuring how well the development team performs planned work in sprint cycles. One of the most used metrics is Sprint Velocity, which determines the average work done in a sprint in story points. The calculation is as follows.

$$Velocity = \frac{Total\ Story\ Points\ Completed}{Number\ of\ Sprints} \quad (2)$$

Eq. (2) helps project managers to assess the delivery capacity of the team and spot trends that might signify a shift in productivity or the occurrence of a bottleneck [25]–[26]. Figure 2 shows how teams track their Sprint Velocity, which gauges the amount of work a team can accomplish in each sprint at a reasonable level. This is a practical way of working in agile, based on its core value of learning from practice and not theorising through planning.

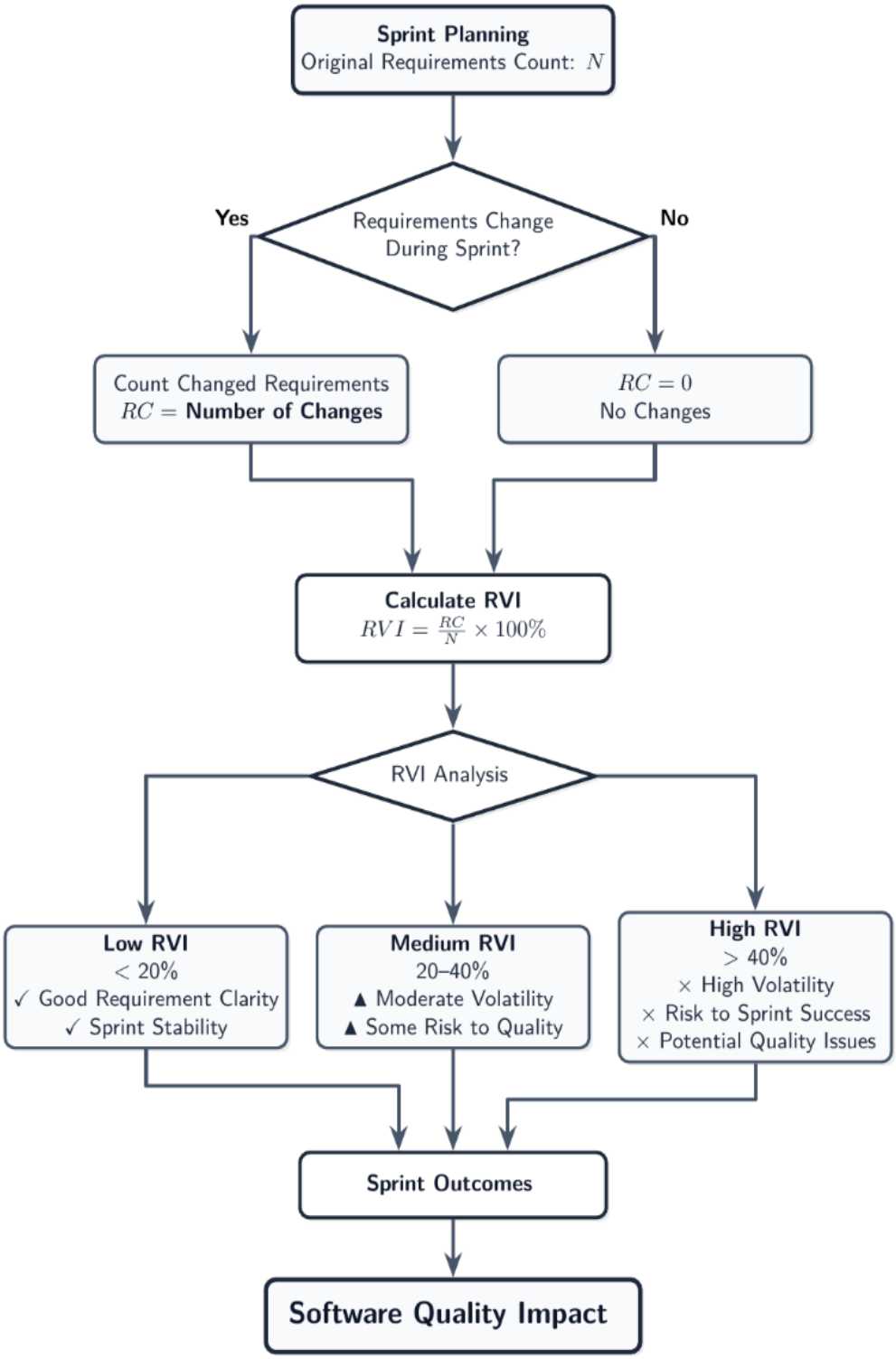



Figure 1. Requirement volatility index flow diagram

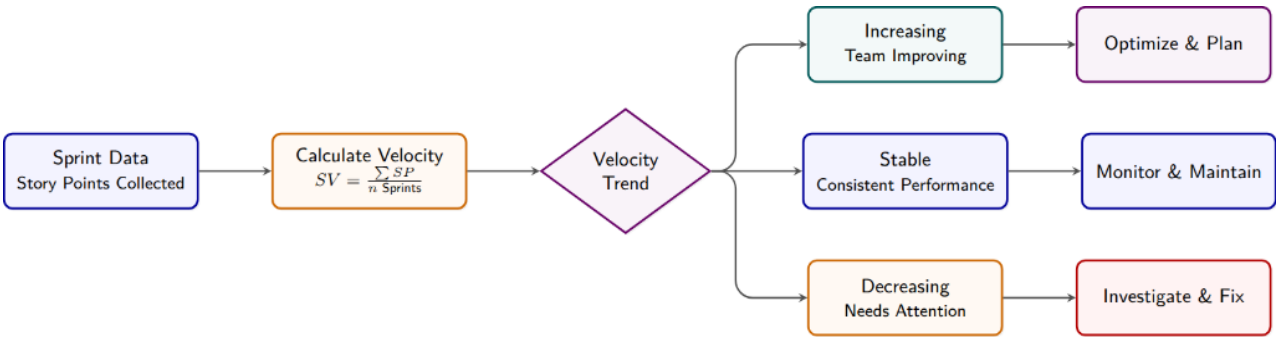



Figure 2. Sprint velocity measurement process

To put it simply, velocity just answers the question: "How much can we really get done?" The process starts by collecting story points, the estimates teams allocate to user stories during planning meetings—over several sprints. The following formula was applied:

$$SV = \sum \left( \frac{Story\ points}{Number\ of\ springs} \right) \tag{3}$$

The teams then calculate throughput as an average. However, the specific number is less significant than the trend 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 increasing, staying the same, or decreasing.

### 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, greater familiarity with project requirements, and fewer development bottlenecks. Consistently increasing velocity may enable teams to handle more complex tasks in future sprint cycles. 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 timeframes. A decline in sprint velocity may indicate underlying performance issues that require further investigation to determine the cause. Possible causes include technical debt, increased task complexity, resource limitations, unclear requirements, process inefficiencies, and dependency-related delays. Continuous monitoring of declining velocity helps prevent further disruptions to project performance.

### 4.2 Linking Velocity to Quality

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 the code quality by measuring the number of defects relative to the size of the developed code.

$$Defect\ dencisy\ = \left( \frac{Total\ defects}{KLOC} \right) \tag{4}$$

This metric calculates defects per thousand lines of code (KLOC). Normalising 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. Therefore, the defect density provides a standardised measure of software quality. A lower defect density indicates a higher software quality. A defect density of fewer than five defects per KLOC may reflect effective testing practices, efficient code reviews, and well-managed development processes. In contrast, a 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 the quality standards. High velocity with high defect density may indicate that development speed is prioritised over quality. Low velocity with low defect density may suggest conservative development practices, whereas low velocity with high defect density reflects significant performance and quality concerns that require corrective actions. The visualisation in Figure 3 shows the Quality Index (QI) model, a synthetic metric that combines various quality dimensions in agile development. It embodies continuous improvement theory and quality management, expressing quality assurance in measurable indicators.

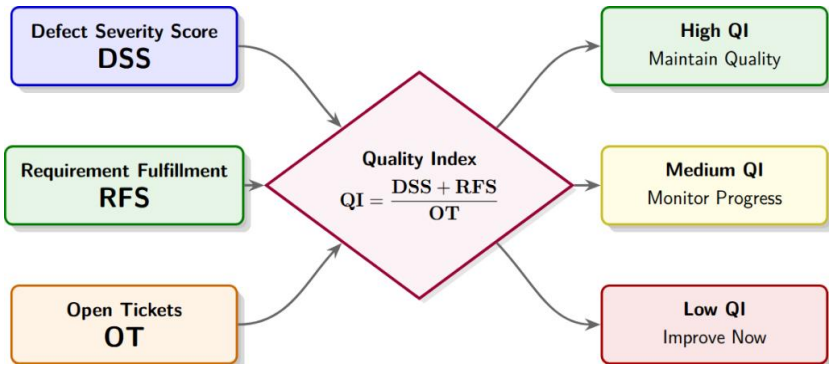



Figure 3. Defect density quality measurement

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 product quality.

Requirement Fulfilment Score (RFS):
Monitors the degree of compliance of delivered functionalities with the specified requirements, thereby corroborating the agile principle 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 balanced quality measure, with higher values indicating 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 defect counts are a very basic method for counting defects, defect density normalises 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 code maturity and the effectiveness of the development process.

Industry benchmarks set limits on the following:
A high QI (low defect density <5/KLOC) signifies maintenance-level monitoring.
Medium QI (5-10 defects/KLOC): Moderate technical debt implies that progress should be actively monitored.
Low QI (>10 defects/KLOC): Severe quality issues that require urgent addressing.

Sprint Management and Schedule Variance:
Supporting quality measurements and monitoring schedule adherence also help identify velocity deviations that can undermine quality. The Schedule Variance formula is as follows:

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

where Earned Value (EV) is the work done and Planned Value (PV) is the planned work, which can issue early warning signs. This aligns with Earned Value Management (EVM) theory, which covers scope, schedule, and cost factors. A positive SV means the sprint is ahead of schedule, whereas a negative SV indicates delays that may compromise quality through rushed development or reduced testing time [27]–[28].

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

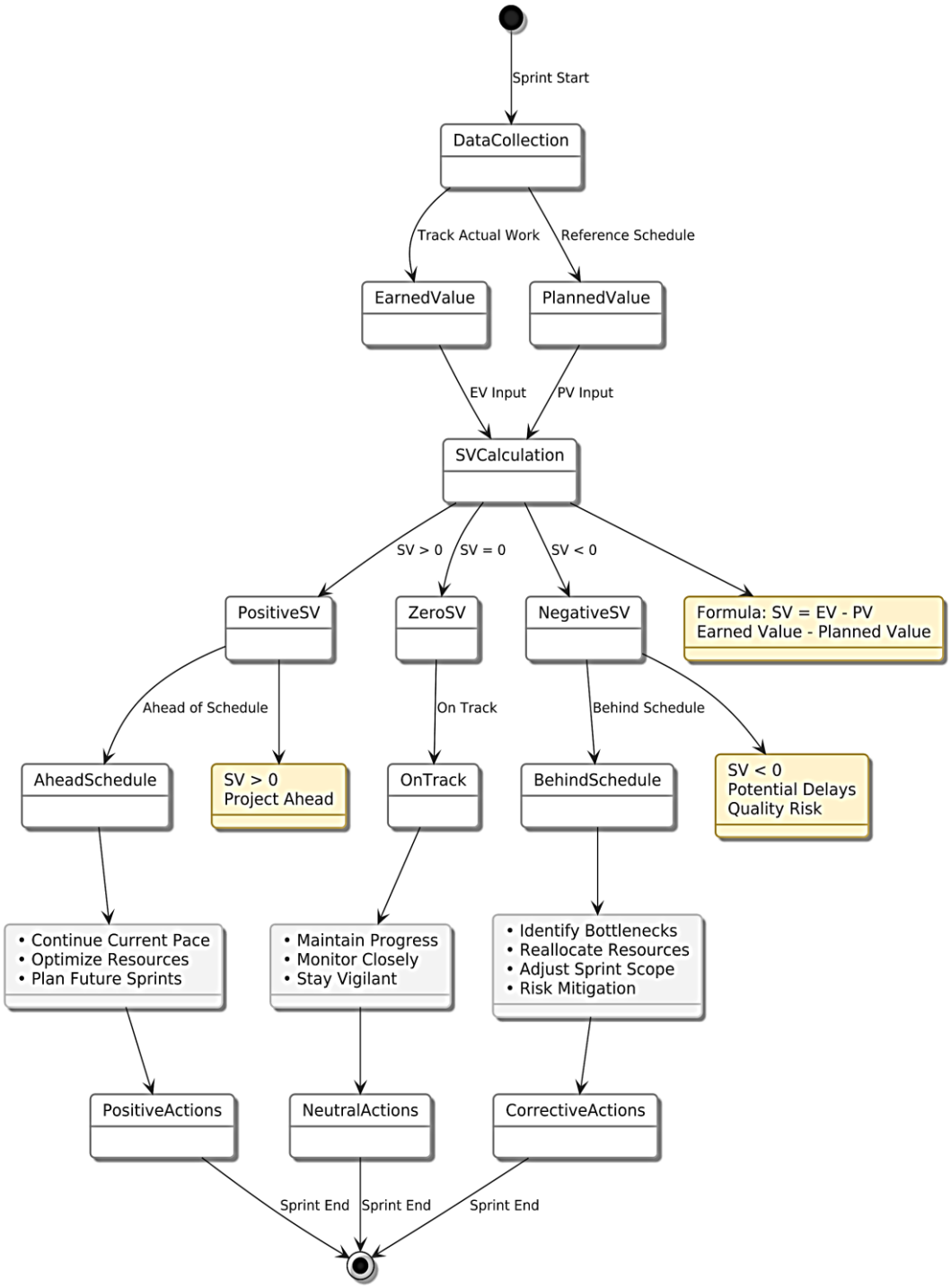



Figure 4. Schedule variance monitoring system

The state diagram in Figure 4 presents the process of monitoring the Schedule Variance as a deterministic method for the 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 is split into three outcome states: positive SV indicating that the project is ahead of schedule and thus, optimising 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.

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

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

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

The flowchart in Figure 5 introduces the Quality Index as an all-encompassing measure of quality that integrates several quality-associated variables into a composite score. Three input elements feed into the middle calculation in the diagram: DSS, which captures the weighted defect effect; RFS, which captures the required satisfaction levels; and OT, which captures open issues. The formula QI = (DSS + RFS) / OT presents a combined measurement quality score, which is then categorised 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 and drastic action.

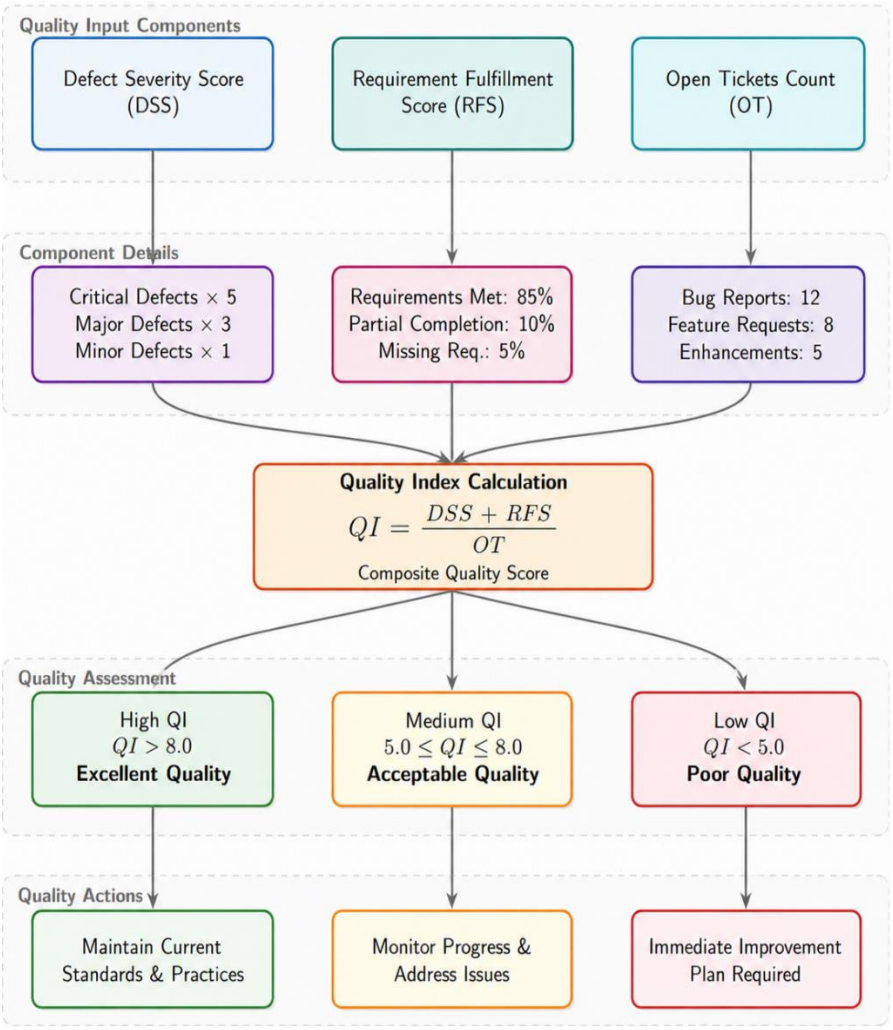



Figure 5. Quality index composite measurement

Figure 6 illustrates the proposed conceptual framework for agile software development quality metrics. Consequently, this framework is built on a four-layer approach, with each layer reinforcing the previous one to achieve 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 provides 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, summarises the operational ideas from these foundations: Requirement Clarity, the definition and understanding of customer needs; Sprint Efficiency, the agile team's productivity and iteration capacity; and Software Quality, a standard for deliverables. The depicted interactions imply that clarifying requirements improves sprint productivity, which, in turn, improves software quality and the defect rate. All these core concepts are indicated in the Metrics Layer, where quantitative indicators are defined for each concept: 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 directly indicate the level of implementation and outcomes of core concepts. Below this is the Implementation Layer, where the required concepts and measurements appear as agile processes: Sprint Planning, Development Execution, Quality Assurance, and Continuous Improvement. The feedback and guidance arrows in the diagram represent

the dynamic, iterative improvement cycle among implementation areas, metrics, and core concepts, reflecting the adaptive and empirical nature of agile practice.

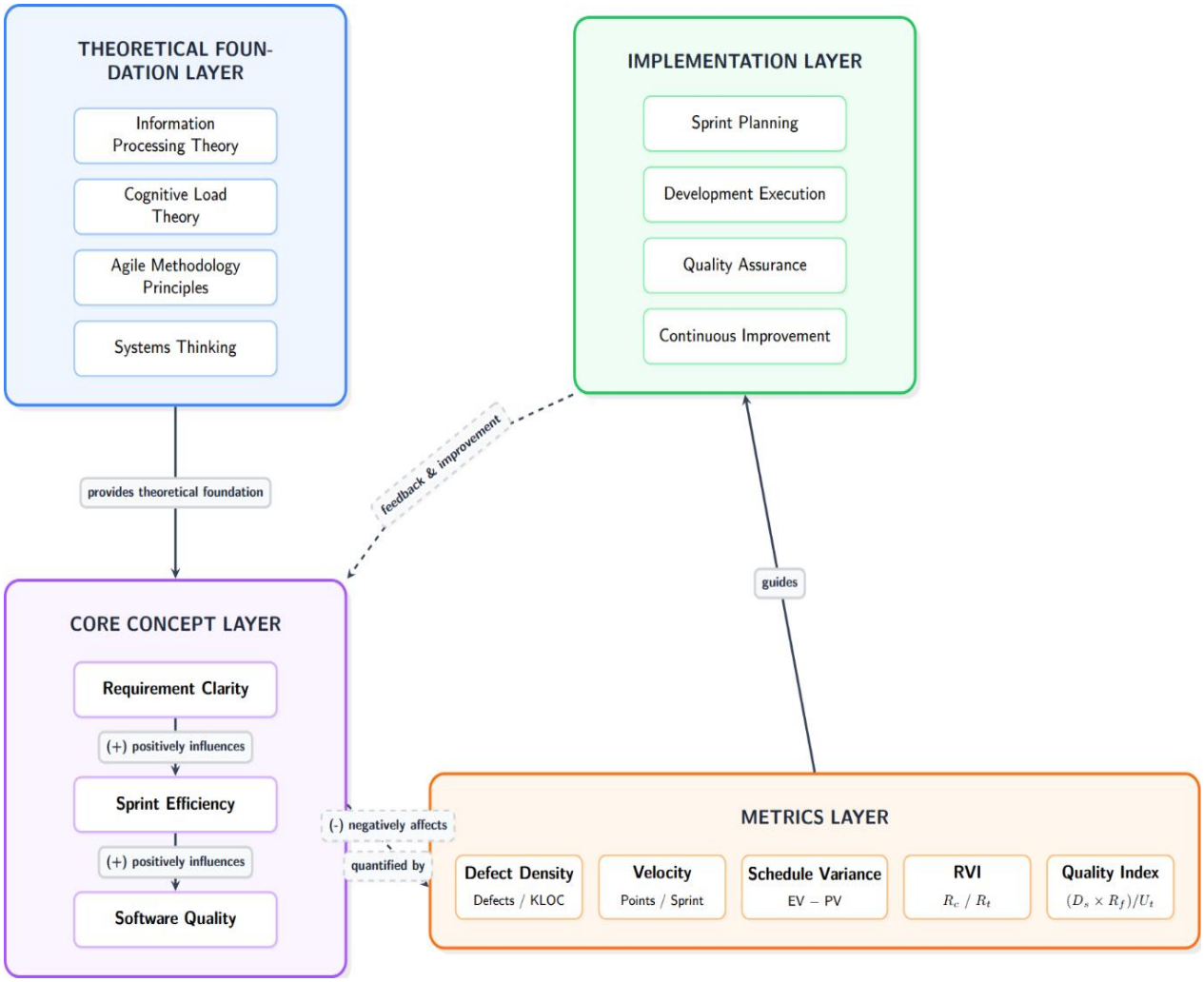



Figure 6. Conceptual framework for agile software development quality metrics

## 5. Discussion

This section presents the theoretical and practical implications of requirements clarity and sprint performance in agile development practice. The reviewed literature includes peer-reviewed studies that present metrics, comparisons, and guidelines for software quality, delivery time, and requirement handling. The following three comparison tables provide a structured framework for viewing the findings. Table 2 highlights the metrics used to measure the impact of requirement clarity on project predictability and sprint efficiency in Agile. Table 3 reviews the frequently utilised sprint and quality metrics that indicate the efficiency of agile teams' operations and output quality. Table 4 reviews the metrics that support the early identification of ambiguity and promote better requirement elicitation practices. The comparative metrics revealed 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 procedural compliance for project success. Moreover, sprint predictability is directly impacted by requirement clarity, as shown by the increase in Sprint Predictability Ratios from 0.65-0.75 (unclear requirements) to 0.85-0.95 (clear requirements), signifying a 15-30% improvement in delivery precision. Finally, the ripple effect shows that the best teams, with clarity in the requirements as characterised 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 organised 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 recognise gaps early, respond appropriately, and develop continuous improvement cycles based on empirical data rather than subjective evaluations.

Table 2. Metrics reflecting the impact of requirement clarity

| Metrics | Description | Insight |
|---|---|---|
| RVI (Requirement volatility index) [31] | It measures the requirement volatility. $RVI = \frac{R_c}{R_t}$ | A high RVI was associated with inconsistent sprint results. |
| Requirement Understanding score [32] | Qualitative score from interviews with stakeholders | Qualitative scores from interviews with stakeholders. |
| Sprint predictability ratio [33] | Completed versus committed stories ratio | Greater clarity enhances the precision of delivery. |
| Change request frequency [34] | Frequency of mid-sprint changes requested | A lower frequency is associated with the original clarity. |
| Stakeholder alignment index [35] | Level of agreement during requirement reviews | A high index increases the reliability of sprint planning. |

Table 3. Sprint performance and quality metrics

| Metrics | Description | Insights |
|---|---|---|
| Velocity [32] | Number of Sprints | Increases with repeated cycles and well-defined objectives. |
| Defect density [36] | Defects per KLOC | Decreases when the requirements are better understood. |
| Code rework ratio [34] | Percentage of code reworked after the sprint | A large ratio suggests unstable or ambiguous specifications. |
| Story completion rate [37] | % of stories with status "done" at sprint completion | The measure of the correctness of the sprint estimation. |
| Defect removal efficiency [38] | Defects removed before release / total defects | This is linked to the accuracy of the 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 the flexibility of planning. |
| 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 | It is directly related to product acceptance. |

## 6. Practical Implication

For Agile teams, sprint-based performance measures, particularly velocity, rate of story completion, and defect density, offer a tangible means of monitoring ongoing improvement. Project managers can prioritise RVI or QI during sprint planning. Simultaneously, QA engineers can use the defect removal efficiency and UAT pass rate to verify the test alignment with business requirements. Based on the theories cited in the conceptual framework design section, the research questions were answered as follows.

RQ 1: How does ambiguity in requirements influence the 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 minimal documentation and fast iterations; therefore, uncertainty at the start of the process can create problems throughout development. According to Information Processing Theory, when requirements are ambiguous, the team struggles to interpret stakeholder needs correctly, increasing error rates and leading to misinterpreted deliverables [13]. This is supported by Cognitive Load theory, which suggests that changing requirements mentally burden developers and reduce their cognitive capacity to perform tasks optimally [14], [16]. However, empirical evidence also shows that imprecise requirements increase the likelihood of defects during and after development. The authors of [7] found that vague requirements hinder sprint planning and degrade product quality. In such cases, common measures like Defect Density and RVI are often high, indicating weak quality control and reactive defect management [32]–[33]. Additionally, Systems Thinking shows 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 satisfaction.

RQ 2: How does requirement incompleteness relate to defect density and sprint interruption?
Incompleteness results in high defect density and high sprint interruption rates. Incomplete requirements can include incomplete user stories, acceptance criteria, or volatile backlog items, all of which introduce uncertainty within the sprint cycle. Studies such as [8] and [3] identify missing requirements as the primary cause of rework, cost, and deadline overrun. Such interferences are generally measured using key metrics. For example, a high Code Rework Ratio and a high frequency of Change Requests within a sprint signal poorly documented requirement [36]. The Defect Density metric typically increases when missing inputs trigger incorrect code that cannot satisfy user needs [36]. The Sprint Predictability Ratio, a measure of completed versus planned stories, declines as requirements become increasingly incomplete [35]. Team velocity and morale also reflect this. According to [4], teams react 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 organisations can take steps toward improved flow and better sprint output quality [35], [40].

RQ 3: Which assurance strategies are most effective in avoiding the effects of unclear or changing requirements in agile projects?
Various SQA approaches help reduce the impact of uncertain or variable requirements in agile contexts. They rely on early verification, stakeholder collaboration, and feedback loops. Works such as [5] and [19] have highlighted the importance of requirement verification techniques, especially when integrated into sprint planning from the outset. For example, the UAT Success Rate and the Requirement Fulfilment Score reliably indicate whether the initial requirements were adequately translated into deliverables [32]. Organisational levels call for better alignment between developers and stakeholders. The Stakeholder Alignment Index has verified that better alignment signifies sprint success and reduced rework [35]. Employing ambiguity detection mechanisms in collaborative sessions can also significantly reduce the risk of misinterpretation [41]. In addition, 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 improves understanding of how near-requirements problems can escalate into system-wide failures, as recommended by Boehm and Turner [22]. Its practicality for real-world use is supported by its direct application in agile software development environments marked by persistent sprint inefficiencies and 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 gives project managers and agile practitioners mechanisms to continuously monitor project health using metrics. For example, a high RVI immediately signals unstable or uncertain requirements, providing a starting point for stakeholder reviews or backlog adjustments. Similarly, reduced sprint velocity or increased defect density provides actionable feedback on workload balance and code quality. These metrics provide real-time performance measurements and support data-driven retrospectives, so decisions are based on empirical evidence rather than subjective judgment. Moreover, the model promotes predictive agility, reflecting a team's ability to anticipate risks from ambiguous requirements and address them well before they cause schedule delays or quality loss. This makes agile project management a proactive, analytical discipline underpinned by continuous improvement and quantified outcomes.

## 7. Conclusions

This study offers a comprehensive conceptual framework that links theoretical concepts to real-world problems in agile software development, with reference to interlinkages among the clarity of requirements, sprint efficiency, and software quality. The research shows that the more unclear the requirements are, the greater 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 explain this relationship. Metrics such as RVI, Sprint Velocity, Defect Density, Schedule Variance, and QI measure agile project health and provide a quantifiable way to assess and improve outcomes. Requirement validation, stakeholder engagement, and early quality reviews reduce rework, defects, and improve team morale. A key contribution of this framework is that it gives agile practitioners, project managers, and quality engineers a way to be more proactive, strategically address requirement-related issues, and foster a culture of continuous improvement to deliver high-quality software on time.

**Acknowledgements**
The authors thank the Department of Computer Science, Institute of Business Management, Karachi, Pakistan, for providing the infrastructure needed to carry out this research.

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

**Declaration of Competing Interest**
The authors declare no conflict of interest.

**Credit Authorship Contribution Statement**
Syed Shabeeb Raza: Conceptualization; Methodology; Data curation; Writing - original draft; Resources
Khalid Mahboob: Formal analysis; Visualisation; Writing - review & revise; Supervision
Mustafa Ahmed Khan: Validation; Investigation; Writing - review & revise

**Availability of Data and Materials**
The data supporting the findings of this study are available upon request from the corresponding author.

**Ethics Declarations**
This study did not involve human participants or animal subjects. Therefore, ethical approval was not required.

**Generative Artificial Intelligence Declarations**
No generative artificial intelligence tools were used to prepare this manuscript. All intellectual content is original and is solely attributable to the authors.

**References**

[1] M. R. A. Souza and T. Conte, "Improving clarity and completeness in user stories: Insights from a multi-domain analysis with developer feedback," in *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS)*, vol. 2, Porto, Portugal, 2025, pp. 272–279, https://doi.org/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, p. 12, 2024, https://doi.org/10.3390/informatics11010012

[3] I. Ilays, Y. Hafeez, N. Almashfi, *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, https://doi.org/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, https://doi.org/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, p. 322, 2023, https://doi.org/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, https://doi.org/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, 2022.

[8] H. S. Dar and A. Rahman, "A conceptual framework for reducing requirement engineering challenges in industrial-scale software projects," *International Journal of Innovations in Science & Technology*, vol. 6, no. 2, pp. 698–712, 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, p. 208, 2025, https://doi.org/10.3390/systems13030208

[10] L. S. Geetha, Y. A. B. El-Ebiary, B. S. Rao, *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, https://doi.org/10.14569/IJACSA.2025.0160374

[11] R. E. Dosumu, O. O. George, and C. O. Makata, "Advancing product launch efficiency: A conceptual model integrating agile project management and scrum methodologies," *International Journal of Advanced Multidisciplinary Research and Studies*, vol. 5, no. 2, pp. 1233–1238, 2025, https://doi.org/10.62225/2583049X.2025.5.2.3971

[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, https://doi.org/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, https://doi.org/10.1016/j.jksuci.2021.04.016

[14] D. Helgesson, "Exploring cognitive waste and cognitive load in software development: A grounded theory," in *Proceedings of the 34th Annual Workshop of the Psychology of Programming Interest Group (PPIG)*, 2023, pp. 60–73.

[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, p. 8766, 2023, https://doi.org/10.3390/su15118766

[16] D. Helgesson, D. Appelquist, and P. Runeson, "A grounded theory of cognitive load drivers in novice agile software development teams," in *Proceedings of the 33rd Annual Workshop of the Psychology of Programming Interest Group (PPIG)*, 2022, pp. 197–215.

[17] P. Belém, R. de Mello, A. S. Vivacqua, and A. Neves de Souza, "Investigating the cognitive load drivers of software evolution activities," in *Proceedings of the ACM International Conference Proceedings Series*, 2023, pp. 342–347, https://doi.org/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, M. K. Baklizi, I. Alsmadi, *et al.*, "Challenges of software requirements quality assurance and validation: A systematic literature review," *IEEE Access*, vol. 9, pp. 137613–137634, 2021, https://doi.org/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, https://doi.org/10.1590/1806-9649-2022V29E3122

[21] G.-L. Dugarte-Peña, M.-I. Sánchez-Segura, A. de Amescua, F. Medina-Domínguez, and S. Armenia, "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, https://doi.org/10.1049/sfw2.12031

[22] D. I. Silva and L. K. B. Siriwardana, "Comparative analysis of software quality assurance approaches in development models," *Research Square*, preprint, 2023, https://doi.org/10.21203/rs.3.rs-3458412/v1

[23] K. Kharchenko, O. Beznosyk, B. Bulakh, B. Kyriusha, and V. Yaremenko, "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, https://doi.org/10.15587/2706-5448.2024.310411

[24] M. Poženel, L. Fürst, D. Vavpotič, and T. Hovelja, "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, https://doi.org/10.1142/S021819402350064X

[25] R. U. Koralage, "Novel approach to estimate velocity for agile scrum sprint planning," preprint, 2023, https://doi.org/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, https://doi.org/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, pp. 20:1–20:11, 2015, https://doi.org/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, https://doi.org/10.1109/TSE.2022.3228739

[29] Y. J. Pérez Castillo, S. D. Orantes Jiménez, and P. O. Letelier Torres, "Sprint management in agile approach: Progress and velocity evaluation applying machine learning," *Information*, vol. 15, no. 11, p. 726, 2024, https://doi.org/10.3390/info15110726

[30] E. Kula, E. Greuter, A. van Deursen, and G. Gousios, "Dynamic prediction of delays in software projects using delay patterns and Bayesian modeling," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, San Francisco, CA, USA, 2023, pp. 1012–1023, https://doi.org/10.1145/3611643.3616328

[31] K. P. Pham and M. Neumann, "How to measure performance in agile software development? A mixed-method study," in *Proceedings of the 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2024, pp. 443–450, https://doi.org/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, https://doi.org/10.1504/IJASM.2021.118061

[33] M. Huss, D. R. Herber, and J. M. Borky, "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, https://doi.org/10.3390/software2030015

[34] S. Jain, V. Jotwani, and S. Agrawal, "Comparative analysis of agile frameworks: Evaluating their impact on software development lifecycle efficiency," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 23s, pp. 3126–3137, 2024.

[35] F. Uzomah, J. Agbana, and A. J. Alu, "A comparative analysis of agile and traditional project management methodologies: Impact on project success metrics in the IT sector amid rapid technological evolution," *Educational Administration: Theory and Practice*, vol. 30, no. 1, pp. 5737–5746, 2024, https://doi.org/10.53555/kuey.v30i1.9237

[36] R. Menezes, M. Marinho, and S. Sampaio, "Metrics in large-scale agile software development: A multivocal literature review," in *Proceedings of the Congresso Ibero-Americano em Engenharia de Software (CIbSE)*, 2024, pp. 106–120, https://doi.org/10.5753/cibse.2024.28442

[37] L. López, X. Burgués, S. Martínez-Fernández, *et al.*, "Quality measurement in agile and rapid software development: A systematic mapping," *Journal of Systems and Software*, vol. 186, p. 111187, 2022, https://doi.org/10.1016/j.jss.2021.111187

[38] B. Falah, S. Abufardeh, and O. Tawose, "Evaluating agile methodologies for software requirements and construction: A SWEBOK-based analysis," *Transactions on Engineering and Computing Sciences*, 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, https://doi.org/10.1007/s13198-023-01958-5

[40] P. Malla, "Analyzing the impact of agile methodologies on software quality and delivery speed: A comparative study," *World Journal of Advanced Research and Reviews*, vol. 25, no. 1, pp. 1207–1216, 2025, https://doi.org/10.30574/wjarr.2025.25.1.0184

[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*, vol. 5, pp. 2063–2072, 2025, https://doi.org/10.1017/pds.2025.10220