
Scalable Quality Assurance Model for Distributed Agile Software Engineering Teams

Abstract

Distributed agile software engineering enables organizations to coordinate teams across locations, but quality assurance becomes difficult when communication, ownership, testing practices, and integration responsibilities vary. This study proposes a Distributed Agile Quality Coordination Model, called DAQCM, to maintain software quality as team numbers increase. The model combines local quality cells, exposure-based assurance routes, cross-team quality clusters, shared evidence records, asynchronous handover boards, and selective product-level escalation. It was evaluated across five scaling stages involving three to fifteen distributed teams and compared with independent team QA, centralized review, and shared checklist-based assurance. DAQCM limited escaped integration defects to 7.8 per 100 stories at the fifteen-team level, compared with 19.6, 14.7, and 11.8 for the comparison approaches. It also improved evidence completeness and reduced cross-team response time. These findings show that federated quality ownership and standardized evidence can support scalable assurance without creating excessive coordination or review bottlenecks in distributed agile programmes.

Keywords: distributed agile, quality assurance, software teams, integration defects, quality coordination

1. Introduction

Distributed agile software engineering allows organizations to combine specialists from different regions while maintaining short development cycles and frequent product delivery. Teams can work across locations, time zones, languages, and organizational units using shared repositories, communication platforms, continuous integration services, and agile planning tools. However, physical separation can reduce informal communication and delay the identification of quality problems. Trust is especially important in distributed agile teams because members must rely on work completed by colleagues whom they may rarely meet directly [1]. A scalable quality assurance process must therefore support both technical verification and reliable collaboration.

Quality activities are often implemented differently by separate agile teams. One team may emphasize automated testing, while another may depend strongly on manual review or exploratory testing. This variation becomes difficult to manage when several teams contribute to the same platform. Empirical evidence from remote agile projects indicates that team success depends on technical infrastructure, organizational support, communication, coordination, and suitable working practices [2]. Without shared quality rules, defects may pass between services, ownership may become unclear, and product-level risks may remain hidden until integration or release.

Distributed development also creates delays when quality decisions depend on central reviewers or a small specialist group. A management model for distributed agile teams must balance team autonomy with sufficient visibility and coordination across the wider project [3]. Excessive central control can slow delivery, but complete independence can produce inconsistent test coverage, incompatible acceptance criteria, and repeated defects. Quality assurance should therefore use common evidence and escalation rules while allowing each team to select practices suited to its component and sprint workload.

Human behaviour also affects whether quality risks are reported early. Team members may avoid raising concerns when they expect blame, conflict, or negative judgement. Psychological safety has been associated with quality-related behaviours such as admitting mistakes, sharing concerns, helping colleagues, and solving problems collectively [4]. A scalable assurance model should create visible technical controls while encouraging teams to communicate incomplete work, uncertain requirements, failed tests, and emerging risks. This study proposes a Distributed Agile Quality Coordination Model, called DAQCM, to maintain consistent software quality as the number of participating teams increases.

2. Methodology

DAQCM was designed as a federated quality assurance structure in which every agile team remained responsible for its own component quality while contributing evidence to a shared product-level quality

system. The model contained three operational levels: team quality cells, cross-team quality clusters, and a product assurance council. A team quality cell managed local acceptance criteria, automated tests, review completion, and defect correction. Cross-team clusters handled shared interfaces and integration risks, while the council examined only high-impact issues that could affect several product areas. This arrangement prevented routine testing decisions from becoming centralized bottlenecks.

At the start of each sprint, every user story received a Quality Exposure Profile. The profile recorded functional criticality, affected services, interface count, data sensitivity, release dependency, expected test effort, and historical defect density. Stories with low exposure remained within the local team process. Medium-exposure stories required evidence from at least one dependent team, while high-exposure stories were added to the cross-team quality cluster. Continuous integration can improve cooperation and quality assurance, although it can also create technical and process difficulties when teams use pipelines inconsistently [5]. DAQCM therefore assigned pipeline controls according to change exposure rather than applying the same workflow to every story.

The model created a Shared Quality Evidence Record for each completed change. This record included acceptance-test results, unit and integration coverage, review status, static-analysis findings, unresolved defects, interface compatibility results, and deployment readiness. Evidence was collected automatically from repositories, build systems, test services, and issue trackers. Modern code review supports product quality and process improvement, but review effectiveness also depends on workflow design and practitioner needs [6]. DAQCM used reviewer selection rules based on component ownership, dependency impact, and available expertise instead of requiring approval from a permanent central reviewer.

A Quality Synchronization Index was calculated for each team at the end of every sprint. The index combined evidence completeness, build stability, test reliability, review turnaround, cross-team dependency status, and defect reopening frequency. The score was not used to rank individual developers. It identified teams that required support, additional integration testing, or temporary quality assistance. Large-scale evidence from software projects shows that testing, code review, static analysis, and build automation are not always applied together with the same intensity [7]. The index therefore measured the combined quality process rather than treating one activity as sufficient.

Cross-team quality clusters met asynchronously through a structured Quality Handover Board. Each board entry contained the changed interface, dependent teams, expected behaviour, evidence link, unresolved risk, required response, and completion deadline. Shared patterns and recurring defects were transferred to a Quality Practice Library. Communities of practice can support knowledge exchange and coordinated learning in large-scale agile development [8]. DAQCM used this mechanism to distribute testing methods, defect-prevention rules, and reusable test assets without making every team attend additional formal meetings.

The model applied adaptive escalation when evidence indicated a product-level risk. A change was escalated when it failed a critical interface test, affected several teams, contained an unresolved severe defect, or caused repeated build instability. Lower-level warnings were returned to the local team with a correction deadline. Agile teams often manage information from multiple development sources, and data integration, automation, and data quality can become major challenges [9]. DAQCM therefore normalized evidence fields and used one common identifier to connect stories, commits, builds, tests, defects, and releases. The five coordination mechanisms are summarized in Table 1.

Table 1. Federated quality coordination mechanisms used by DAQCM

Coordination mechanism	Primary responsibility	Evidence collected	Activation rule	Quality outcome
Team quality cell	Local agile team	Acceptance, unit-test, review, and defect evidence	Applied to every sprint change	Maintains component-level quality
Exposure-based assurance route	Story owner and quality lead	Criticality, dependencies, data impact, and test effort	Selected when a story enters sprint planning	Matches assurance effort to change risk
Cross-team quality cluster	Representatives of dependent teams	Interface tests, contract results, and shared-risk status	Activated for medium- or high-exposure changes	Reduces integration defects
Quality handover board	Producing and receiving teams	Interface change, evidence link, response, and deadline	Used when work crosses ownership boundaries	Improves asynchronous coordination
Product assurance escalation	Product assurance council	Severe findings, repeated failures, and release impact	Triggered only by product-level risk	Prevents major defects from reaching release

The evaluation simulated a distributed enterprise software programme containing fifteen agile teams located across five time zones. Each team delivered services to a common digital operations platform through two-week sprints. The model was compared with independent team-level QA, centralized quality review, and a shared checklist-based approach. Team count was increased through five experimental stages containing three, six, nine, twelve, and fifteen teams. Every method received equivalent requirements, defect injections, interfaces, sprint durations, and computing resources. The primary measure was escaped integration defects per 100 completed stories, while supporting measures included review delay, failed build recovery time, evidence completeness, duplicate defect rate, and cross-team response time.

3. Results and Discussion

All four quality assurance approaches produced relatively similar results when only three teams were involved. Independent team-level QA recorded 7.4 escaped integration defects per 100 stories, centralized review recorded 6.8, the shared checklist approach recorded 6.3, and DAQCM recorded 5.9.

At this scale, communication paths were limited and team members could resolve most interface questions directly. The advantage of the proposed model became clearer as additional teams and dependencies were introduced.

Figure 1 presents the escaped integration defect rate across increasing distributed team counts. Under independent QA, the rate increased from 7.4 defects with three teams to 19.6 defects with fifteen teams. Centralized review limited some defects but increased from 6.8 to 14.7 because the review group became overloaded. The checklist approach reached 11.8 defects at the highest scale. DAQCM maintained the lowest rate and increased only from 5.9 to 7.8 defects per 100 stories.

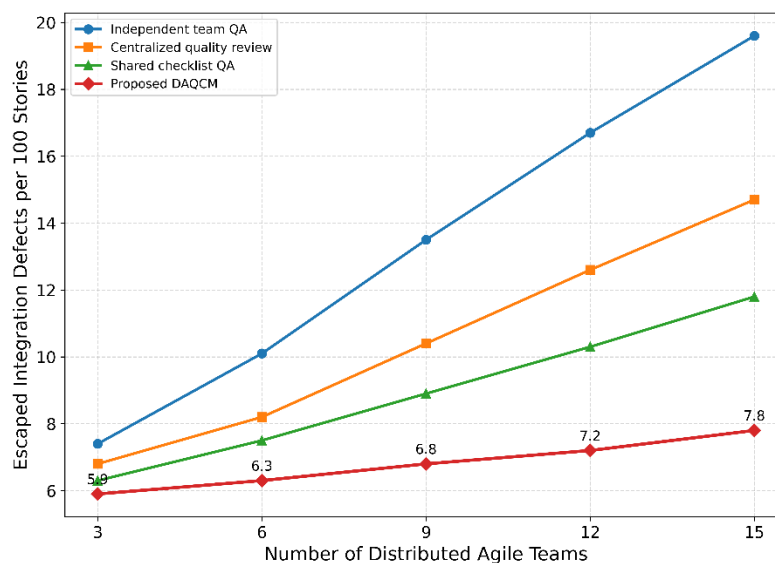


Figure 1. Escaped integration defect rate across increasing numbers of distributed agile teams

The proposed model performed better because it did not depend entirely on local autonomy or centralized approval. Exposure profiles identified which stories required cross-team evidence, while routine low-risk changes remained within the team workflow. The Quality Handover Board reduced uncertainty when interfaces moved between ownership areas. Shared identifiers also allowed teams to trace a requirement through commits, builds, tests, defects, and release evidence. This reduced repeated requests for information and allowed quality problems to be addressed before final integration.

DAQCM also reduced the average cross-team quality response time from 13.2 hours under independent QA and 10.6 hours under checklist-based QA to 6.1 hours. Centralized review required 17.5 hours at the fifteen-team level because work accumulated around the central reviewers. Evidence completeness reached 95% under DAQCM, while the comparison methods ranged from 68% to 84%. The results indicate that quality assurance can scale more effectively when responsibilities are federated, evidence is standardized, and escalation is reserved for risks that genuinely affect the wider product.

4. Conclusion

This study proposed DAQCM, a scalable quality assurance model for distributed agile software engineering teams. The model combined local team ownership with exposure-based assurance, shared quality evidence, cross-team quality clusters, asynchronous handover, and selective product-level escalation. It also introduced a Quality Synchronization Index to identify incomplete evidence, unstable builds, slow reviews, and repeated defects without using the score to evaluate individual developers. The approach maintained agile team autonomy while creating consistent quality visibility across a distributed programme.

The evaluation showed that DAQCM limited the increase in escaped integration defects as the number of teams grew from three to fifteen. It also improved evidence completeness and reduced cross-team response time compared with independent, centralized, and checklist-based quality assurance. The model can support large enterprise programmes that require frequent delivery from geographically separated teams. Further work should evaluate DAQCM using production project histories, larger team networks, different agile scaling methods, and long-term measurements of product maintainability and team workload.

References

1. Tyagi, S., Sibal, R., & Suri, B. (2022). Empirically developed framework for building trust in distributed agile teams. *Information and Software Technology*, 145, 106828.
2. Adzgauskaite, M., Tam, C., & Martins, R. (2025). What helps Agile remote teams to be successful in developing software? Empirical evidence. *Information and Software Technology*, 177, 107593.
3. Montalvão Junior, D., Batista, T., & Cavalcante, E. (2023, September). An Agile Management Model for Distributed Software Development Teams. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (pp. 122-131).
4. Alami, A., Zahedi, M., & Krancher, O. (2024). The role of psychological safety in promoting software quality in agile teams. *Empirical Software Engineering*, 29(5), 119.
5. Soares, E., Sizilio, G., Santos, J., Da Costa, D. A., & Kulesza, U. (2022). The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, 27(3), 78.
6. Badampudi, D., Unterkalmsteiner, M., & Britto, R. (2023). Modern code reviews—Survey of literature and practice. *ACM Transactions on Software Engineering and Methodology*, 32(4), 1-61.
7. Khatami, A., & Zaidman, A. (2024). State-of-the-practice in quality assurance in Java-based open source software development. *Software: Practice and Experience*, 54(8), 1408-1446.

8. Tobisch, F., Schmidt, J., & Matthes, F. (2024, May). Investigating communities of practice in large-scale agile software development: An interview study. In *International Conference on Agile Software Development* (pp. 3-19). Cham: Springer Nature Switzerland.
9. Fawzy, A., Tahir, A., Galster, M., & Liang, P. (2025). Exploring data management challenges and solutions in agile software development: a literature review and practitioner survey. *Empirical Software Engineering*, 30(3), 77.