
Risk Aware Test Planning Model for Large Scale Agile Software Development Projects

Abstract

Large-scale agile projects require flexible test planning because requirement changes, code complexity, business importance, defect history, and sprint dependencies vary across development cycles. This study presents a risk-aware test planning model that converts these factors into risk scores for user stories and features. The scores guide test strategy selection, priority setting, tester allocation, automation support, and environment use. The model was evaluated across eight agile sprints and compared with uniform and history-based planning. Results showed that the proposed approach maintained higher test coverage in low-, medium-, and high-risk conditions, with the greatest gains during complex and critical sprints. It also improved the use of limited testing resources by directing deeper testing toward unstable, defect-prone, and business-critical items. The model supports more reliable sprint planning, stronger cross-team coordination, earlier defect detection, and lower overall release risk in large-scale agile software projects.

Keywords: Risk-aware testing, Agile software development, Test planning, Test coverage, Resource allocation

1. Introduction

Large-scale agile software projects involve several teams, frequent releases, shared components, and changing business requirements. These conditions make test planning difficult because all features cannot receive the same level of testing within a limited sprint period. Team capability, customer involvement, and coordination have a strong effect on the success of agile software projects [1]. A suitable test plan must therefore consider both technical risk and the working conditions of the project.

Agile teams commonly update requirements during backlog refinement, sprint planning, and development. Frequent requirement changes may affect completed features, test cases, system interfaces, and delivery schedules. Requirement volatility can increase rework and reduce planning accuracy when changes are not identified and controlled early [2]. Testing effort should therefore be adjusted whenever the stability or importance of a requirement changes.

Large projects also contain features with different levels of business importance and technical difficulty. A failure in a payment, security, or customer-data function may cause greater damage than a failure in a low-priority internal feature. Critical success factors in agile projects include clear requirements, team communication, management support, and effective planning [3]. Risk-aware test planning uses these conditions to identify where deeper testing and additional resources are required.

Uniform test planning may assign similar effort to all sprint items even when their failure probability and impact are different. Risk-based testing gives more attention to functions with greater business impact, technical complexity, or defect likelihood. Large-scale agile environments also require coordination between teams so that shared risks are not assessed separately or ignored [4]. This study proposes a risk-aware test planning model that updates testing priorities and resource decisions during successive agile sprints.

2. Methodology

The proposed methodology included sprint-data collection, risk identification, risk-score calculation, test-strategy selection, resource allocation, and planning evaluation. Data was collected from product backlogs, sprint boards, source-code repositories, test-management systems, and defect records. Each user story was linked with its affected components, planned effort, earlier defects, business priority, and dependent sprint items. The information was updated before every sprint planning meeting.

Five main risk factors were used in the model: requirement volatility, code complexity, business criticality, defect history, and sprint dependency. Each factor was converted into a normalized score between zero and one. A risk-based testing framework for distributed agile development can improve test selection by linking software risks with suitable testing priorities [5]. The proposed model combined the five factor scores to calculate one risk value for every user story or feature.

Requirement volatility was measured using the number and importance of changes made during recent sprints. Code complexity was obtained from structural metrics, affected-module size, and integration difficulty. Business criticality represented financial, operational, security, and customer impact, while defect history described the frequency and severity of earlier failures. Sprint dependency measured how strongly one item depended on work completed by other teams or components.

Table 1 presents the risk factors and the related test-planning decisions. Low-risk items received focused functional and smoke testing, while medium-risk items received broader integration and regression testing. High-risk items received deeper functional, integration, security, performance, and exploratory testing. Test case selection and prioritization can improve regression efficiency by combining coverage, change information, and failure history [6]. The model therefore increased testing depth when several high-risk indicators appeared together.

Table 1. Agile Risk Factors, Sprint Characteristics, Test Planning Priority, and Resource Allocation Decisions

Risk factor	Measured project indicator	Assigned risk level	Planned testing strategy	Allocated testing resources
Requirement volatility	Number, frequency, and impact of requirement changes	Low, medium, or high	Change-focused functional and regression testing	Additional analyst and tester time for unstable stories
Code complexity	Complexity, module size, coupling, and integration difficulty	Low, medium, or high	Unit, integration, boundary, and exploratory testing	Senior tester and automation support for complex modules
Business criticality	Financial, security, customer, and operational impact	Low, medium, or high	End-to-end, security, performance, and recovery testing	Priority environment access and specialist reviewers
Defect history	Earlier defect count, severity, and reopening rate	Low, medium, or high	Defect-focused regression and negative testing	Increased regression coverage and review effort
Sprint dependency	Number of dependent teams, services, and backlog items	Low, medium, or high	Interface, contract, integration, and workflow testing	Shared test environment and cross-team coordination time

The final risk score was calculated using weighted values assigned to the five factors. Business criticality and severe defect history received greater weights because failures in these areas could produce larger project damage. Many-criteria regression test selection can balance fault detection, test cost, and practical project constraints [7]. Risk weights were reviewed during sprint planning so that product owners, developers, and testers could adjust them when project priorities changed.

Testing resources were assigned according to risk level and available sprint capacity. High-risk items received earlier test design, more experienced testers, additional automation, and wider environment coverage. Medium-risk items received standard functional and integration testing, while low-risk items received essential validation and selected regression tests. Machine learning test-prioritization studies

show that test value can change across development cycles as new execution evidence becomes available [8]. The proposed model therefore updated risk values after every completed sprint.

The model was evaluated across successive sprints using test coverage, early defect detection, escaped defects, planning accuracy, and testing effort. It was compared with uniform test planning and history-based planning under the same sprint capacity. Contextual information can improve test prioritization by connecting earlier test outcomes with the current development cycle [9]. The evaluation therefore considered both historical project evidence and the current sprint risk profile.

3. Results and Discussion

The proposed risk-aware test planning model was evaluated across eight successive agile sprints. Test coverage was measured under low-risk, medium-risk, and high-risk development conditions. The proposed model was compared with uniform test planning and history-based test planning using the same sprint capacity. The results showed that risk-aware planning maintained stronger coverage because testing effort was adjusted according to requirement volatility, code complexity, business importance, defect history, and sprint dependency.

The largest improvement appeared during medium-risk and high-risk sprints. Figure 1 shows the test coverage achieved by the three planning approaches across the eight sprint cycles. The proposed model increased testing depth when several risk indicators became high and reduced unnecessary effort when the risk level was lower. This adaptive allocation allowed the team to maintain broad coverage without applying the same testing intensity to every backlog item.

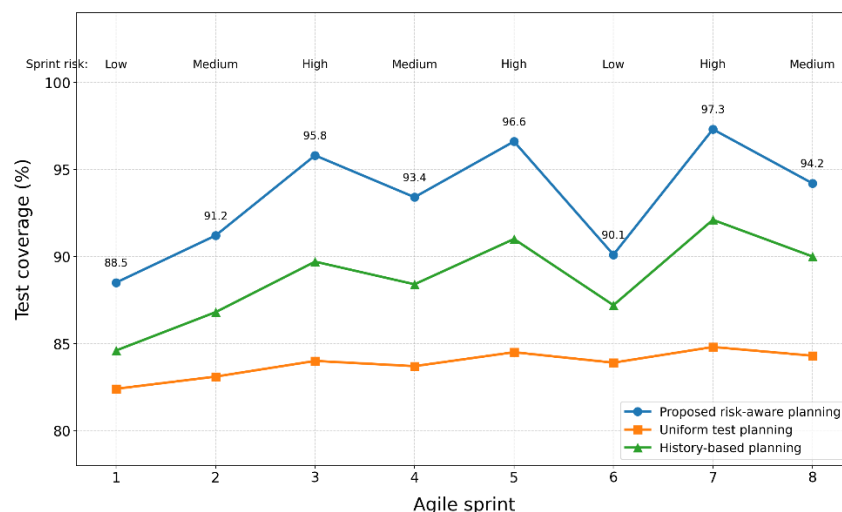


Figure 1. Test Coverage Achieved Across Successive Agile Sprints Under Different Risk Levels

Uniform test planning produced a stable but lower coverage pattern because similar testing effort was assigned to all sprint items. This method did not respond well when critical or complex features entered the backlog. History-based planning performed better because it used earlier defects and test outcomes,

but it reacted more slowly to new requirement changes and cross-team dependencies. The proposed model produced a more balanced response by combining current sprint risk with historical project evidence.

The coverage difference became clearer during high-risk sprints that contained unstable requirements, complex code, critical business functions, and several dependencies. The proposed model assigned more experienced testers, broader test environments, and additional automation to these items. It also preserved essential testing for low-risk work so that less critical features were not ignored. These results show that risk-aware planning can improve test coverage and resource use across changing agile sprint conditions.

4. Conclusion

This study presented a risk-aware test planning model for large-scale agile software development projects. The model used requirement volatility, code complexity, business criticality, defect history, and sprint dependency to estimate the risk of each user story or feature. These risk values were then used to select testing strategies, set planning priorities, and allocate testers, automation support, and test environments. The approach connected technical risk with sprint capacity and allowed the test plan to change as project conditions evolved.

The results showed that the proposed model maintained higher test coverage than uniform and history-based planning across successive agile sprints. It provided the greatest benefit during medium-risk and high-risk conditions because it directed more testing effort toward complex and important features. The model can help large agile teams use limited testing resources more effectively, improve coordination across dependent teams, and reduce the chance that critical defects escape into later releases.

References

1. Barros, L., Tam, C., & Varajão, J. (2024). Agile software development projects—Unveiling the human-related critical success factors. *Information and Software Technology*, 170, 107432.
2. Mohammad, A., & Kollamana, J. M. (2024, March). Causes and mitigation practices of requirement volatility in agile software development. In *Informatics* (Vol. 11, No. 1, p. 12). MDPI.
3. Binboga, B., & Gumussoy, C. A. (2024). Factors affecting agile software project success. *IEEE Access*, 12, 95613-95633.
4. Ahmad, M. O., Ghanbari, H., & Gustavsson, T. (2022). Team performance and large scale agile software development. *arXiv preprint arXiv:2209.07878*.

5. Khanna, E., Popli, R., & Chauhan, N. (2025). A Novel Risk-Based Testing Framework for Distributed Agile Software Development. *International Arab Journal of Information Technology (IAJIT)*, 22(6).
6. Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2021). Test case selection and prioritization using machine learning: a systematic literature review. *arXiv preprint arXiv:2106.13891*.
7. Dobslaw, F., Wan, R., & Hao, Y. (2023). Generic and industrial scale many-criteria regression test selection. *Journal of Systems and Software*, 205, 111802.
8. Marijan, D. (2023). Comparative study of machine learning test case prioritization for continuous integration testing. *Software Quality Journal*, 31(4), 1415-1438.
9. da Roza, E. A., do Prado Lima, J. A., & Vergilio, S. R. (2024). On the use of contextual information for machine learning based test case prioritization in continuous integration development. *Information and Software Technology*, 171, 107444.