Universal Library of Engineering Technology Research Article

ISSN: 3064-996X | Volume 3, Issue 2

Open Access | PP: 26-30

DOI: https://doi.org/10.70315/uloap.ulete.2026.0302005

The Role of Test Architecture Restructuring in Enabling Shared Quality Ownership Between QA Engineers and Developers

Vitali Skadorva

Test Automation Developer Specialist, Intact Financial Corporation, Gatineau, Canada.

Abstract

This article examines the role of test architecture restructuring as a condition for the transition to shared quality ownership between QA engineers and developers in an environment of accelerating release cycles and increasing requirements for software system reliability. The study's relevance stems from the contradiction between the need for high delivery speed and the team's reliance on monolithic E2E checks that introduce delays, regression fragility, and bottlenecks in the verification process. The aim of the study is to substantiate multi-level test decomposition as a mechanism for redistributing responsibility for quality and increasing engineering resilience. The article's scientific novelty lies in integrating a technical model of layered test coverage decomposition with an organizational model of QA maturity. It is shown that shifting checks to the component, UI integration, and API levels transforms the structure of team participation in quality assurance. Empirical data indicate a reduction in regression time, decreased flakiness, and the release of labor resources. It is concluded that the architectural restructuring of tests creates a foundation for the Quality Engineering model and consolidates quality as a team-wide practice. The article will be useful for software engineering researchers, QA leads, test architects, developers, and product team managers.

Keywords: *Quality Assurance, Test Architecture Restructuring, Shared Responsibility, Shift-Left, Component Testing, Test Decomposition, Agile Development.*

INTRODUCTION

The software development industry is undergoing a transformation driven by the need to accelerate release cycles without compromising system reliability (Mylsamy & Mehrotra, 2025). In this context, the traditional approach to quality assurance, based on the sequential execution of development and testing phases, is shown to be inadequate. The core problem lies in the architectural overload of verification processes, where the main burden of checks is placed on end-to-end E2E scenarios executed at the end of the development lifecycle (Elsayed et al., 2023). This gives rise to a bottleneck, in which the test engineering team becomes a bottleneck for product delivery.

The relevance of this study is driven by the need to move from the declarative appeal to shared quality ownership to specific technical and architectural mechanisms that ensure this process. Shared responsibility for quality requires a change in organizational structure. It also requires restructuring of the test infrastructure. Without a technical base that allows developers to obtain rapid feedback, responsibility for quality remains nominal.

The research problem is a contradiction between the need for high development speed and the fragility of monolithic test architectures, which impede developer involvement in quality assurance processes. The purpose of the study is to substantiate the role of multi-level decomposition of test architecture as an important factor that enables the distribution of responsibility among team members and increases operational efficiency.

To achieve this purpose, the following objectives are addressed.

1. Analysis of the evolutionary spectrum of quality assurance maturity from the control model to engineering assurance.

2. Mathematical and logical substantiation of the decomposition of monolithic E2E tests into component, integration, and system levels.

3. Assessment of the impact of the new architecture on the distribution of labor costs and roles.

Citation: Vitali Skadorva, "The Role of Test Architecture Restructuring in Enabling Shared Quality Ownership Between QA Engineers and Developers", Universal Library of Engineering Technology, 2026; 3(2): 26-30. DOI: https://doi.org/10.70315/uloap.ulete.2026.0302005.

4. Validation of the methodology on empirical data from six organizations with a detailed analysis of performance indicators in the insurance sector.

The scientific novelty of the study lies in the synthesis of a technical test decomposition model with an organizational QA maturity model. Specific cases demonstrate how the architectural decision to isolate UI components and mock the API layer directly affects developers' psychological ownership of code and the staffing resilience of QA departments.

In accordance with the stated purpose, the study tests the following hypotheses.

Hypothesis 1. Multi-level decomposition of test architecture reduces the duration of regression cycles, decreases the instability of automated checks, and lowers the labor costs of test maintenance.

Hypothesis 2. Shifting test coverage to the component, UI integration, and API levels creates conditions for shared responsibility for quality between developers and QA engineers and consolidates the Quality Engineering model in team practice.

MATERIALS AND METHODOLOGY

The methodological basis of the study is built on a combination of qualitative and quantitative approaches to the analysis of engineering processes. The study uses the following methods.

A comparative analysis of architectural patterns was conducted by comparing the traditional testing pyramid with the proposed multi-level decomposition model. Differences in execution speed, signal reliability, and maintenance cost at each level are analyzed.

The case study was conducted by examining the implementation of the methodology across six anonymized organizations in North America and Canada during the period from 2016 to 2025. The research objects include companies from the insurance sector (Organization A), e-commerce and automotive manufacturing (Organization B), digital security (Organization C), enterprise artificial intelligence (Organization D), B2B AI (Organization E), and educational platforms (Organization F). In the analysis of Organization A (insurance), data from 26 regression cycles were examined, ensuring the statistical significance of the conclusions.

RESULTS AND DISCUSSION

The restructuring of test architecture involves reconfiguring processes that change the team's interaction paradigm. The main results of the study are structured around three axes: the evolution of the maturity model, the technical mechanism of decomposition, and the organizational redistribution of responsibility.

The transition to shared responsibility for quality cannot be achieved within the traditional Quality Control model. As the data show, in the QC model testing is concentrated in the hands of an isolated team, performed after development is completed, and leads to delivery delays (Bernardi et al., 2021). Three maturity levels can be distinguished through which organizations progress as they transform their processes. This is shown in Table 1.

Table 1. Quality assurance maturity spectrum in an organization

Characteristic	Quality Control, QC	Quality Assistance	Quality Engineering, QE
Primary Focus	Defect detection	Support in testing	Defect prevention
Responsibility	Isolated QA team	Developers + QA, consulting	Entire cross-functional team
Testing Timing	After development, Post-Dev	In parallel with development	Integrated into SDLC, Shift-Left
Tooling	Manual tests + heavy E2E	Sprint-level automation	Multi-layer decomposition + AI
Impact on Architecture	None	Limited	Full, design for testability

Architectural restructuring acts as the primary instrument for moving the organization from the right side of the spectrum to the left. The Quality Engineering model assumes that automated testing becomes an integral part of the code-writing process, whereas in other approaches it remains only a final check. Within this paradigm, test engineers evolve into quality architects who design an environment in which defects are either prevented at the design stage or detected immediately.

The inefficiency of monolithic E2E tests can be substantiated as follows. In modern web applications, user interface components exhibit high reusability. For example, a single component, such as an input field or navigation element, may be used on 32 different pages. With 5 brands and support for 2 languages, the number of unique combinations requiring verification across end-to-end scenarios reaches 320.

In traditional architecture, the QA team is forced to maintain all these scenarios, leading to a combinatorial explosion. Restructuring presupposes a layered decomposition algorithm that collapses this verification space, as shown in Figure 1.

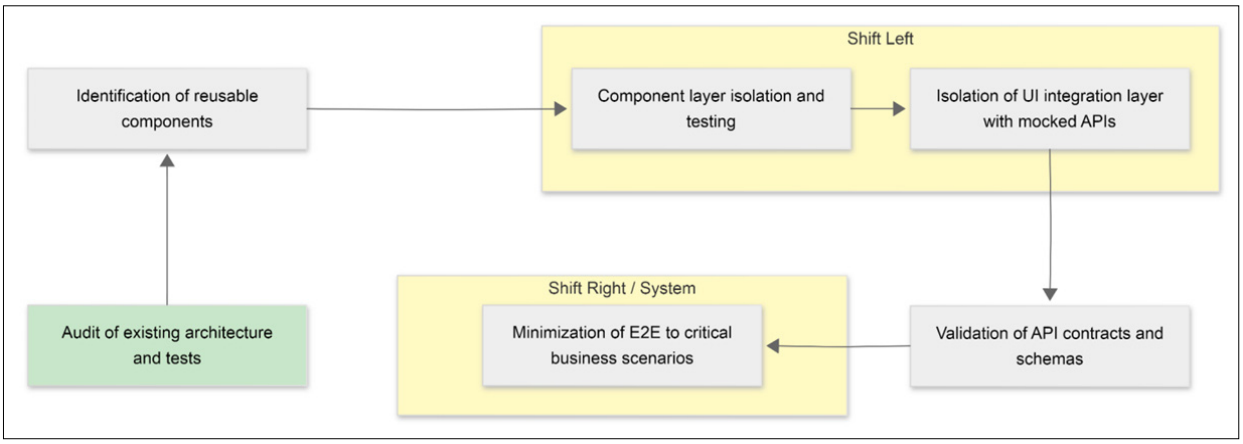

Fig. 1. Algorithm for restructuring test architecture to ensure scalability

According to this algorithm, the architecture is rebuilt using a multi-level approach. At the component level, the behavior of reusable elements is moved into isolated tests, for example, using Cypress Component Testing. Such tests mount the component in memory and do not require launching the entire application. As a result, 320 end-to-end checks can be reduced to a single suite of component tests.

At the UI integration level, each page is tested as a component integration using a mock API server. This approach eliminates instability caused by network delays and backend state. As a result, the test environment becomes more controlled and reproducible.

At the API level, verification of contracts, schemas, and parameter variations is carried out independently of the user interface. This enables reliable checks of integration boundaries and early detection of discrepancies between interacting parts of the system.

At the E2E level, end-to-end tests are retained only for complete user scenarios. This concerns cases such as the end-to-end process of filing an insurance claim, which cannot be verified at lower levels. This distribution of checks makes the test architecture more resilient and more rational.

Architectural restructuring creates a natural technological basis for shared responsibility. In the QE model, the question of who writes tests is resolved through proximity to the code and the task context.

Low-level component tests are written by developers. Such tests require deep knowledge of the implementation and are executed in the same development environment as the functional code. Including test writing in the developer's definition of done increases psychological ownership of the product. Studies show that developers are more apt to make their systems testable when they are responsible for writing the verification test themselves (Rahman et al., 2024).

QA engineers and Software Development Engineers in Test (SDETs) focus more on the UI integration and E2E layers, work on the system overall, analyze complex scenarios, and support the testing infrastructure. This allows the QA team to avoid becoming a department for fixing failing tests and to direct their expertise toward high-level risks.

To consolidate this model, a trigger system is introduced into the CI/CD pipeline that determines when feedback is obtained.

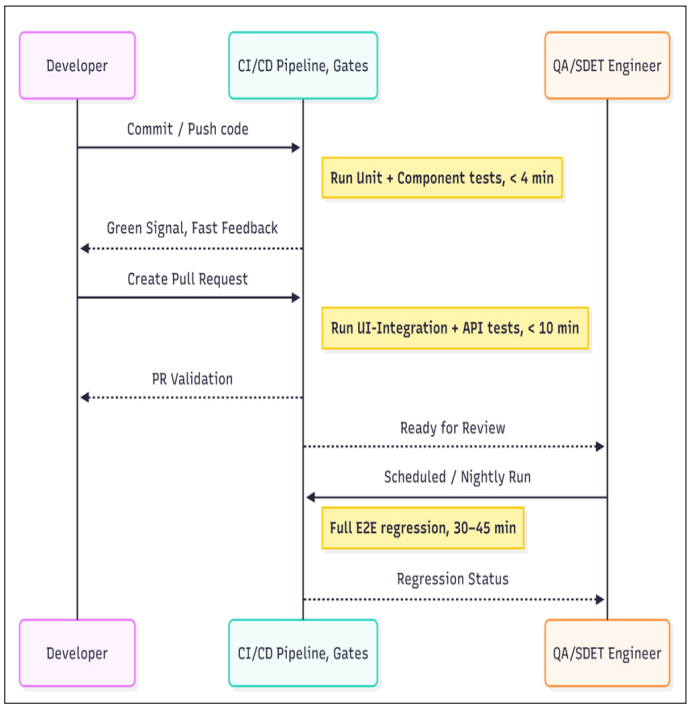

Fig. 2. A model for distributing responsibility through triggers in a CI/CD pipeline

This model implements the shift-left principle at the architectural constraint level. A developer cannot physically pass the code further if component tests have not passed. This reduces defect correction cycles and prevents defects from reaching the system testing phase, where corrections are more costly.

The most demonstrative restructuring results were obtained in Organization A. Before the transformation began, the company used a monolithic suite of 621 E2E tests. The regression process was highly inefficient. Execution took more than 3.5 hours. The level of fragility was 25%, which meant that one quarter of the tests could fail for reasons

unrelated to real bugs, that is, due to network failures and timeouts.

After applying the multi-level decomposition methodology, the architecture was changed. Instead of 621 heavy tests, 1257 fast component tests, 360 UI integration tests, and only 80 targeted E2E scenarios were created.

The testing model based on end-to-end coverage included 621 E2E tests, whereas the layered model used 80 tests. This shift reduced the number of E2E tests by 87.1%. Execution time per cycle decreased from 3 hours 41 minutes 57 seconds to 20 minutes 30 seconds, which corresponds to a 90.7% reduction. Average flakiness fell from 0.25 to 0.02, with a decrease of 92.0%.

Annual maintenance time declined from 221 hours to 39 hours, which equals a reduction of 82.4%. Regression time across 26 cycles decreased from 96 hours 10 minutes 48 seconds to 8 hours 53 minutes, which represents a 90.8% reduction. The new structure also yielded 269.3 hours of team time saved per year. These values indicate substantial gains in test efficiency, stability, and maintenance effort.

This confirms that architectural restructuring leads to explosive productivity growth. The reduction in regression time frees the equivalent of several working weeks for the entire team. The decrease in flakiness from 25% to 2% restored developers' trust in automation results, which became a factor in the transition to shared responsibility. A developer in Organization A now sees test results within 4 minutes at the commit stage, instead of waiting 4 hours until the end of the day.

Despite differences in technological stacks such as Cypress, Playwright, Java, and REST Assured, the results of methodology validation across other industries demonstrate recurrent patterns.

In Organization B, operating in e-commerce and the automotive sector, implementing this approach reduced the time to deliver new features by 60%. This result was achieved by eliminating the need to wait for long UI tests to complete.

In Organization C, belonging to the field of digital security, the transition to component testing ensured 100% coverage of critical authentication scenarios. For an industry with high security requirements, such a change has fundamental significance.

Organizations D and E operate in the Enterprise and B2B AI segments respectively. Additional constraints are applied due to the higher complexity of the developed algorithms. At the same time, the final restructuring of tests made it easier to separate the user interface from the backend-heavy computations, speeding up frontend development.

Release stability was a meaningful benefit for Organization F in the education sector, where platform content constantly changes and is rapidly evolving.

The main effect across all of the cases was to find a stable QA staffing model; developers were doing over half the automation at the component level. As a result, companies did not need to ramp up tester staffing with the number of their product teams.

Conversely, those numbers show how many barriers must be dealt with in order to restructure effectively.

Developers may perceive the obligation to write tests as an additional burden that reduces their pure productivity (Straubinger & Fraser, 2023). To overcome this risk, focus on the fact that a sound test architecture reduces debugging time and eliminates the need to return to old code several days later when QA discovers a defect in it.

The transition to the QE model requires technical skills from test engineers, including an understanding of frontend architecture and working with CI/CD tools. A shortage of such specialists may slow implementation. Maintenance of a complex system of triggers and isolated environments requires investment in DevOps practices. Insufficient CI server capacity may result in fast tests being delayed in the queue (Bournassenko, 2025).

The methodology is most effective in applications with a component-oriented architecture. In legacy monoliths, extracting isolated components may require deep refactoring of the application code (Wei et al., 2025).

CONCLUSION

The conducted study confirms that test architecture restructuring plays a decisive role in the transition to a model of shared responsibility for quality. The principal conclusion of the study is that organizational changes are impossible without technical decomposition of test coverage.

The key result of the study was a technological solution to the problem of combinatorial growth in test scenarios. It is shown that transitioning from monolithic end-to-end tests to component and UI integration levels enables reducing the volume of redundant checks by an order of magnitude. This means that the testing system becomes more manageable, while verification of interface behavior and application logic becomes more precise and rational.

The results of the study confirm the first hypothesis. For Organization A, a reduction in regression execution time was observed. The results show that the multi-level decomposition of test architecture improves test stability and efficiency. The move of checks from the component, UI integrated, and API layers changes the distribution of tasks within the team, which confirms the second hypothesis. The developer owned a test area (component tests) and got quick feedback, while the test engineers were mainly focusing on system testing, integration tests and test infrastructure,

which mainly ran at later stages. While establishing quality as a shared engineering practice, this model also set the stage for organizations adopting Quality Engineering.

The approach was validated in the case of Organization A by reducing test execution time and instability. The annual savings in team-working time exceeded 260 hours. Such a result corresponds to a substantial operational effect and a pronounced financial gain, since the reduction of time costs directly affects the productivity of engineering processes.

Particular significance belongs to the architectural consolidation of responsibility within the CI/CD pipeline. The proposed trigger model forms clear boundaries of quality ownership at different stages of development. Developers control component state at the commit stage. QA engineers ensure system integrity at the Pull Request level and within release cycles. Such a distribution of roles strengthens process predictability and makes the quality control system more coherent.

The study also shows the transformation of QA specialists' professional roles. The restructuring of the test model reduces dependence on routine maintenance of fragile scenarios. As a result, QA engineers have the opportunity to move into the roles of quality architects and consultants. This change increases the overall resilience of engineering culture and contributes to the formation of a more mature model of interaction within the team.

The practical significance of the study lies in the possibility of applying the described decomposition algorithm in companies of any scale. The transition to Quality Engineering through architectural restructuring is a strategic imperative for organizations seeking to preserve competitiveness amid accelerating digital transformation. Despite barriers in the form of the need for personnel training and initial infrastructure investment, the long-term benefits in the form of high delivery speed and product stability make this path the only valid one for the modern software engineering industry.

REFERENCES

1. Bernardi, S., Gentile, U., Marrone, S., Merseguer, J., & Nardone, R. (2021). Security modelling and formal verification of survivability properties: Application to cyber–physical systems. *Journal of Systems and Software*, *171*, 110746. https://doi.org/10.1016/j.jss.2020.110746
2. Bournassenko, G. (2025). On Queueing Theory for Large-Scale CI/CD Pipelines Optimization. *ArXiv*. https://doi.org/10.48550/arXiv.2504.18705
3. Elsayed, A., Cerny, T., Salazar, J. Y., Lehman, A., Hunter, J., Bickham, A., & Taibi, D. (2023). End-to-End Test Coverage Metrics in Microservice Systems: An Automated Approach. *ArXiv*. https://doi.org/10.48550/arXiv.2308.09257
4. Mylsamy, S., & Mehrotra, Dr. T. (2025). Continuous Integration and Continuous Delivery (CI/CD) Pipelines. *Journal of Quantum Science and Technology*, *2*(2). https://doi.org/10.63345/jqst.v2i2.284
5. Rahman, S., Saha, A. K., Chakraborty, U., Sujana, H. T., & Abdullah, M. (2024). Evaluating the impact of Test-Driven Development on Software Quality Enhancement. *International Journal of Mathematical Sciences and Computing*, *10*(3), 51–76. https://doi.org/10.5815/ijmsc.2024.03.05
6. Straubinger, P., & Fraser, G. (2023). A Survey on What Developers Think About Testing. *ArXiv*. https://doi.org/10.48550/arXiv.2309.01154
7. Wei, X., Li, J., He, X., Peng, W., Zhu, Y., Gu, R., Zhu, Y., & Huang, J. (2025). Extracting microservices from monolithic applications using a consistent graph enhanced Graph Transformer. *Journal of Systems and Software*, *222*, 112345. https://doi.org/10.1016/j.jss.2025.112345