Universal Library of Engineering Technology Research Article

ISSN: 3064-996X

Open Access | PP: 23-26

DOI: https://doi.org/10.70315/uloap.ulete.2023.004

Methodologies for Accelerated Implementation of Quality Assurance (QA) Processes in Software Projects with Legacy Code and Missing Test Infrastructure

Khudenko Daniil

Quality Assurance Engineer.

Abstract

In the context of rapid digital transformation, legacy code is one of the obstacles to the implementation of innovations and maintaining competitiveness. The study aims to develop and systematize a comprehensive methodology for the accelerated integration of Quality Assurance (QA) processes in software projects with a significant amount of legacy code and no original test infrastructure. The purpose of the work is to analyze the features of methodologies for the accelerated implementation of quality assurance processes in software projects with legacy code and no test infrastructure. The methodological base of the study is formed on the basis of the analysis and synthesis of the main scientific publications in recent years devoted to refactoring, automated testing and analysis of legacy systems. As a result, a model was proposed that includes a preliminary exploratory audit stage with risk-oriented prioritization, formation of a basic test framework (scaffolding), subsequent iterative build-up of test coverage and automation, as well as final integration into the CI/CD pipeline with the organization of continuous monitoring. The results obtained will be of interest to heads of development teams, QA engineers, software architects and other researchers specializing in the modernization and maintenance of complex software systems.

Keywords: *Legacy Code, Quality Assurance, Accelerated Implementation, Test Automation, Refactoring, Risk Analysis, Technical Debt, CI/CD, Test Infrastructure, Software Modernization.*

INTRODUCTION

In the modern digital economy, the ability of organizations to bring their solutions to market quickly is one of the key factors of competitiveness. At the same time, a significant share of the global IT infrastructure continues to rely on outdated, legacy systems. According to a Gartner study, by 2023, expenditures on the modernization of obsolete systems will exceed half of the budgets of major IT companies, with the majority of these costs going not to the development of new features but to the maintenance and integration of existing components [1]. Such systems, often created decades ago in archaic programming languages and deployed on obsolete hardware, are characterized by high architectural complexity, scarce or missing documentation, and, most critically, the near absence of established quality assurance processes. Any change in such code carries the risk of regression failures capable of disrupting business operations [8].

The purpose of this study is to analyze the features of methodologies for the accelerated implementation of quality assurance processes in software projects with legacy code and no testing infrastructure.

The scientific noveltyof the research lies in a hybrid, risk-oriented approach that combines static code analysis to identify vulnerable areas, characterization testing to capture current system behavior, and iterative automation. This approach enables measurable improvement in product reliability within limited time frames.

The author's hypothesis is that step-by-step implementation of the proposed methodology can reduce the time required to deploy an initial set of automated regression tests compared with traditional full-coverage strategies, while simultaneously minimizing the likelihood of critical defects reaching the production environment.

MATERIALS AND METHODS

The issue of accelerated implementation of quality assurance (QA) processes in projects with legacy code and no testing infrastructure is closely related both to global trends in cloud technology investment and DevOps, and to strategies for modernizing applications themselves. Forecast reports demonstrate the growing importance of cloud platforms and DevOps practices. According to Gartner, global end-user spending on public cloud services is expected to reach $597.3 billion in 2023 [1]. Research by Google Cloud and DORA highlights a direct correlation between DevOps maturity and the speed of release delivery while simultaneously reducing the number of defects [8]. SonarSource, in turn, notes that leading software vendors increasingly adopt automated code checks as part of the CI/CD pipeline [10].

Modernization strategies for legacy systems include both organizational and methodological, as well as technological approaches. Ponnusamy S. and Eswararaj D. [2] propose a multi-level modernization model that combines preliminary audits, gradual decomposition of monolithic applications into services, and phased implementation of test infrastructure based on containerization and orchestration. Mishra S. and Tripathi A. R. [12] focus on identifying "technical debt" and its effect on business value, proposing a method for quantitative risk assessment during legacy system upgrades and migration to hybrid cloud architectures.

Regarding the refactoring of legacy code to improve security, Almogahed A., Omar M., and Zakaria N. H. [3] demonstrate the practice of automated vulnerability analysis and incremental application of secure coding patterns. They note that integrating security tests (SAST/DAST) at the refactoring stage makes it possible to detect up to 60% of hidden vulnerabilities without a fundamental architectural redesign.

Machine learning is actively applied to analyze and predict code quality and defects. Khalid A. et al. [4] examine several classifiers - SVM, decision trees, and neural networks - for predicting defect probability at the package level, achieving up to 85% accuracy when trained on historical commit and bug tracker metrics. Alaswad F. and Poovammal E. [5] apply ensemble methods to predict quality metrics such as coverage and code smells, emphasizing the need to enrich training datasets with test coverage data to increase accuracy. Studer S. et al. [6] describe the use of generative models (GPT-like) for automated test case generation and test data synthesis. Fawzy A. H., Wassif K., and Moussa H. [11] propose a framework for online monitoring of DevOps pipelines using anomaly detection algorithms, which enables the identification of regression-related issues immediately after a commit.

Containerization as a means of "isolated" testing for legacy systems is examined in the work of Chippagiri S. and Ravula P. [7], where the authors describe best practices for cloud-native development, including the use of Kubernetes for dynamic deployment of testing environments and integration with CI/CD tools. Watada J. et al. [9] focus on using containers to ensure test environment reproducibility, which is essential when decomposing monolithic systems and in the absence of a unified testing server.

Therefore, the literature reveals a wide range of methods - from strategic and financial justifications for adopting cloud and DevOps technologies to specific techniques involving machine learning and containerization for accelerated quality assurance. However, a gap persists between strategic recommendations and practical tool-based solutions: integration patterns that enable a smooth transition from legacy code auditing to automated testing remain insufficiently described. With respect to ML-based methods, some authors emphasize the high predictive accuracy achieved on historical datasets [4], while others point out weak correlation with real test coverage in production [5]. The scalability of generative approaches to test data [6] in large monolithic codebases and the automatic updating of test containers during architectural changes [7, 9] remain underexplored. Moreover, limited attention has been paid to integrating anomaly detection mechanisms into CI/CD pipelines for legacy systems lacking full test coverage [11].

RESULTS AND DISCUSSION

Based on the analysis of existing approaches and identified gaps, a hybrid methodology for accelerated implementation of QA processes is proposed. It consists of four consecutive yet partially overlapping stages.

The first stage is Reconnaissance and Risk-Based Prioritization. When working with legacy code, the primary task is not to write tests but to gather information and assess risks. Without this stage, efforts are scattered and ineffective. The main tool here is static code analysis using solutions such as SonarQube or CodeScene [2, 4, 10]. The objectives of this stage include:

Identification of "hot spots": modules with high cyclomatic complexity and frequent change history (Git-based analysis).

Technical debt assessment: quantitative estimation of the time required to fix problems (code duplication, vulnerabilities).

Dependency mapping: visualization of component relationships to understand the impact of changes.

Technical metrics should be correlated with the business criticality of functionality. For this purpose, a risk matrix is developed jointly with product owners (see Table 1.) Modules that fall into the quadrant of high business criticality and high technical risk become the top priority for QA implementation.

Table 1. Example of a risk-oriented prioritization matrix for modules (compiled by the author based on [2, 4, 9, 10]).

System Module	Business Criticality (1–5)	Technical Risk (1–5)	Integral Rank	QA Priority
BillingEngine.dll	5 (Highest)	5 (High complexity, frequent edits)	25	1(Highest)
UserAuthentication.svc	5 (Highest)	2 (Stable, simple code)	10	3 (Medium)
ReportingModule.jar	3 (Medium)	4 (Complex SQL queries)	12	2 (High)
LegacyDataExporter.exe	2 (Low)	5 (Unsupported framework)	10	4 (Low)
Localization.lib	1 (Minimal)	1 (Simple, unchanged)	1	5 (Deferred)

Once the highest-risk areas have been identified, the next step is to build a "safety net" (Stage 2). Developing unit tests for tightly coupled legacy components often fails because of excessive interdependencies. Therefore, attention shifts to a higher level of abstraction -characterization testing.

Integration or end-to-end (E2E) scenarios are created to cover key business processes (for example, "order creation and payment" in the BillingEngine.dll module) and to record the final system state (database records, generated reports, API responses). Their task is not to verify correctness but to capture the current behavior of the system. Any unintended change in logic is immediately detected by such tests, forming a regression baseline. The choice of tools depends on architecture: Selenium or Playwright for UI, Postman or REST-assured for API testing [5, 7].

The third stage involves iterative expansion of coverage and automation. The presence of this "safety framework" enables safe refactoring and gradual bottom-up growth of test coverage. This stage proceeds iteratively:

Refactoring for testability: For prioritized modules, the "Boy Scout rule" applies — leave the code cleaner than it was. Developers decouple tight dependencies, for instance, by moving business logic out of interface handlers into separate, easily testable classes [3].

Coverage of new code with tests: All new or refactored code must be covered by unit and integration tests.

Adapted testing pyramid: The classical pyramid (unit tests at the base, E2E tests at the top) is initially inapplicable to legacy projects [5, 6]. Instead, an adapted scheme is used, illustrated in Figure 1.

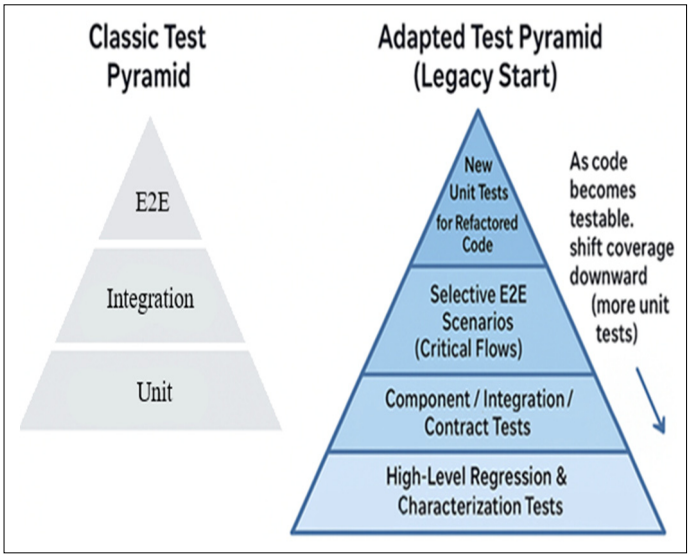

Fig.1. Adapted testing pyramid for legacy systems (compiled by the author based on [5, 6]).

The classical testing pyramid, where fast unit tests form the base and slow E2E tests the top, does not apply to legacy systems at the initial stage. The reason is simple: such code lacks isolated, testable modules.

Therefore, an adapted approach is proposed:

Initial phase (left side): Testing is based on a limited number of broad characterization (E2E) tests. They create a safety framework but are numerous and slow, effectively inverting the pyramid.

Evolution phase (right side): As refactoring progresses and new code is added, faster unit and integration tests emerge. They form a solid foundation, allowing a reduction in the number of slow E2E tests since their logic is partially covered by lower-level tests. Ultimately, the structure evolves toward a classical, stable testing pyramid, significantly accelerating feedback cycles.

Using tools such as Jenkins, GitLab CI, or GitHub Actions, a process is configured to automatically perform the following tasks on every commit:

- static code analysis (immediate verification);

- build execution;

- running fast unit and integration tests [4, 6].

Full runs of long E2E tests are delegated to nightly builds to avoid slowing down development. In parallel, monitoring systems (Prometheus, Sentry) and log analysis tools (ELK Stack) are implemented to detect anomalies in the production environment, providing an additional layer of protection [11].

The overall structure of the proposed methodology is shown in Figure 2.

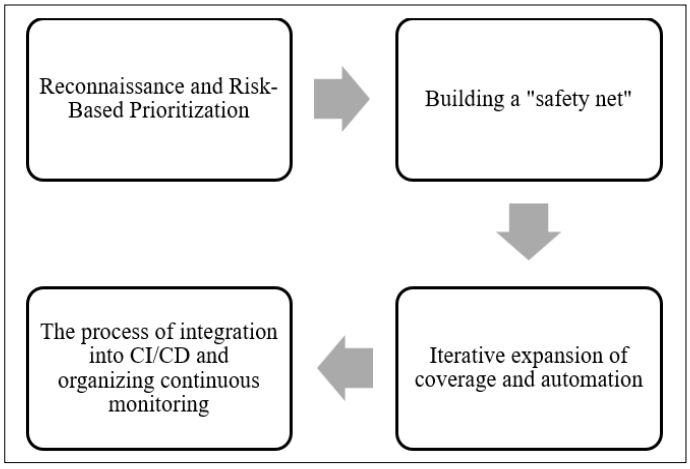

Fig.2. Four-stage methodology for accelerated QA implementation (compiled by the author based on [12])

High-precision analysis of predictive data confirms the feasibility of reducing the time required to implement a full-scale QA process. This effect is achieved by abandoning the "100% coverage" paradigm in favor of a risk-oriented, iterative approach that maximizes the return on quality investments at every stage.

CONCLUSION

Integrating quality assurance processes into projects with legacy code remains one of the most complex and pressing

challenges in modern software engineering. The lack of both testing infrastructure and comprehensive documentation, combined with the architectural complexity and critical nature of legacy systems, creates serious barriers to their maintenance and evolution, while also leading to significant financial and reputational risks.

This study, based on the analysis of prior research in the field, identified several gaps in existing approaches, which are often fragmented and fail to provide a systematic, accelerated solution to the problem. In response, a hybrid four-stage methodology was proposed to transition a project from a state of chaos and uncertainty to a structured model of continuous quality control. The key features of the methodology include: synergy between static analysis and business priorities to optimize the focus of QA efforts; the use of characterization tests as an operational "safety framework"; an adapted testing pyramid reflecting the actual evolution of a legacy project; and the staged integration of QA processes into CI/CD pipelines.

Thus, the goal of the study has been achieved and the hypothesis confirmed. The practical significance of the research lies in providing engineers and project managers with a detailed step-by-step algorithm for addressing one of the most urgent challenges in the IT industry.

REFERENCES

1. Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023. [Electronic resource]. – Access mode: https://www.gartner.com/en/newsroom/press-releases/2023-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023 (Accessed on 06/15/2023).
2. Ponnusamy S., Eswararaj D. Navigating the modernization of legacy applications and data: Effective strategies and best practices //Asian Journal of Research in Computer Science. – 2023. – Vol. 16 (4). – pp. 239-256. DOI: 10.9734/AJRCOS/2023/v16i4386.
3. Almogahed A., Omar M., Zakaria N. H. Refactoring codes to improve software security requirements //Procedia Computer Science. – 2022. – Vol. 204. – pp. 108-115. DOI: 10.1016/j.procs.2022.08.013.
4. Khalid A. et al. Software defect prediction analysis using machine learning techniques //Sustainability. – 2023. – Vol. 15 (6). – pp. 1-17. DOI: 10.3390/su15065517.
5. Alaswad F., Poovammal E. Software quality prediction using machine learning //Materials Today: Proceedings. – 2022. – Vol. 62. – pp. 4714-4720. DOI: 10.1016/j.matpr.2022.03.165.
6. Studer S. et al. Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology //Machine learning and knowledge extraction. – 2021. – Vol. 3 (2). – pp. 392-413.
7. Chippagiri S., Ravula P. Cloud-Native Development: Review of Best Practices and Frameworks for Scalable and Resilient Web Applications //Int. J. New Media Studie. – 2021. – Vol. 8. – pp. 13-21.
8. 2023 State of DevOps Report. Google Cloud & DORA. [electronic resource]. URL: https://cloud.google.com/devops/state-of-devops (accessed on: 12/12/2023).
9. Watada J. et al. Emerging trends, techniques and open issues of containerization: A review //IEEE Access. – 2019. – Vol. 7. – pp. 152443-152472.
10. Khan M. S. et al. Critical challenges to adopt DevOps culture in software organizations: A systematic review //Ieee Access. – 2022. – Vol. 10. – pp. 14339-14349.
11. Fawzy A. H., Wassif K., Moussa H. Framework for automatic detection of anomalies in DevOps //Journal of King Saud University-Computer and Information Sciences. – 2023. – Vol. 35 (3). – pp. 8-19. DOI: 10.1016/j.jksuci.2023.02.010.
12. Mishra S. and Tripathi A. R. AI business model: an integrative business approach //Journal of Innovation and Entrepreneurship. – 2021. – Vol. 10 (1). - pp.1-21.

Citation: Khudenko Daniil, "Methodologies for Accelerated Implementation of Quality Assurance (QA) Processes in Software Projects with Legacy Code and Missing Test Infrastructure", Universal Library of Engineering Technology, 2023; 23-26. DOI: https://doi.org/10.70315/uloap.ulete.2023.004.