International Journal of Computing and Artificial Intelligence

E-ISSN: 2707-658X
P-ISSN: 2707-6571
Impact Factor (RJIF): 5.57
www.computersciencejournals.com/ijcai
IJCAI 2025; 6(2): 164-168
Received: 01-06-2025
Accepted: 06-07-2025

Drogunova Yulia
Bachelor's Degree, Dostoevsky Omsk State University, Omsk, Russian Federation

Corresponding Author:
Drogunova Yulia
Bachelor's Degree, Dostoevsky Omsk State University, Omsk, Russian Federation

Economic assessment of the impact of QA automation in a GitLab-based DevOps environment on the development and maintenance lifecycle of mobile applications

Drogunova Yulia

DOI: https://doi.org/10.33545/27076571.2025.v6.i2c.195

Abstract
This article examines the economic impact of quality assurance automation within a GitLab-based DevOps environment on the development and maintenance of mobile applications. It analyzes the structure of CI/CD pipelines, including the use of containerization, parallel testing, static code analysis, and dependency caching. Emphasis is placed on the reduction of total cost of ownership, increase in return on investment, and savings in labor expressed in full-time equivalents. Based on case studies of Airbus and Ally Financial, the implementation of automated testing processes is shown to significantly accelerate release cycles and reduce costs associated with testing and maintenance. The study concludes that integrating such solutions is strategically important for improving the predictability and efficiency of mobile development processes.

Keywords: DevOps, test automation, GitLab, CI/CD, mobile applications, economic efficiency, development lifecycle

Introduction

In the context of rapid innovation in mobile technologies and intense competition for the market of mobile applications, companies are increasingly being pressed to hasten the release of high-quality mobile applications. The DevOps approach, which facilitates continuous delivery and integration (CI/CD), has emerged as a central component of the modern IT production pipeline. Quality assurance (QA), however, is perhaps the most labor-intensive and error-prone phase of development. Use of automated testing on CI/CD pipelines-particularly by means of the GitLab toolset-significantly reduces release time, operational costs, and the likelihood of production defects, enhancing system stability and lowering support costs.

The aim of this study is to provide an economic assessment of the efficiency of QA automation in the GitLab environment for the development and maintenance of mobile applications.

Main part. Automated testing infrastructure in GitLab CI/CD for mobile applications

A modern DevOps infrastructure for mobile development requires tight integration of testing tools with CI/CD pipelines. GitLab provides a flexible and scalable framework for building such pipelines, allowing for stage configuration, parallel task execution, use of both self-hosted and cloud-based runners, and automated dependency management. According to GitLab reports, as of 2025, the platform is used by over 50 million registered users and more than 50% of Fortune 100 companies.

A typical CI/CD pipeline for mobile projects in GitLab consists of logically separated stages, each corresponding to a specific phase in the application lifecycle-from build to deployment-with integrated quality control and automated testing (fig. 1).

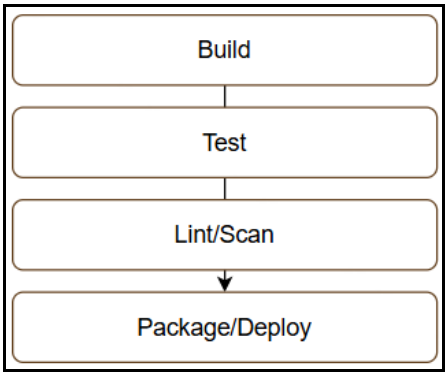

Fig 1: Stages of a typical CI/CD pipeline

Each stage constitutes a separate job within the CI/CD pipeline and is closely linked to a specific set of tools, configurations, and technological solutions. Stability and scalability of the infrastructure are ensured through the use of containerization, parallel runners, caching mechanisms, and integrated static code analysis (table 1).

Table 1: CI/CD pipeline components, tools, and example commands for Android and iOS [1]

Stage/component	Tools & technologies	Example commands (Android)	Example commands (iOS)
Build	Gradle, Xcode, CocoaPods, Swift Package Manager.	./gradlew assembleDebug	xcodebuild-scheme MyApp-configuration Debug
Testing	JUnit, Espresso, XCTest, XCUITest, Appium, Firebase Test Lab, Bitrise.	./gradlew test ./gradlew connectedAndroidTest	xcodebuild test-scheme MyApp-destination 'platform=iOS Simulator,name=iPhone 14'
Static analysis (lint/scan)	ktlint, detekt, SwiftLint, SonarQube.	./gradlew ktlintCheck Detect	Swiftlint sonar-scanner
Publishing/deployment	Firebase App Distribution, GitLab Pages, TestFlight, Google Play Console.	./gradlew publishBundle	xcrun altool--upload-app fastlane deliver
Containerization	Docker, Podman, GitLab Runners.	image: gradle:8.0-jdk17	image: macos-xcode
Scalability	GitLab parallel jobs, test sharding, cloud runners.	parallel: 4	parallel: 3
Caching	Gradle cache, CocoaPods cache.	cache:\n key: gradle\n paths:\n-.gradle/wrapper/\n-.gradle/caches/	cache:\n key: cocoapods\n paths:\n-Pods/\n-~/Library/Caches/CocoaPods/

In addition to the stages described above, smooth pipeline operation also requires a stable execution environment and infrastructure-level automation. To ensure consistency and reproducibility of all pipeline stages, containerization technologies such as Docker or Podman are used. GitLab Runners can be deployed using both containerized and bare-metal virtual machine deployment, depending on the specific needs of the project [2]. For overall performance optimization, dependency and artifact caching mechanisms are utilized (e.g., Gradle cache, CocoaPods cache) that avoid duplicate downloads of libraries and significantly accelerate CI execution.

Key characteristics of an effective testing infrastructure include parallelization and scalability. In GitLab, automated test execution can be organized with support for parallel processing and dynamic task scaling. Features such as test sharding and parallel execution across multiple runners reduce overall pipeline execution time significantly. This is particularly beneficial for large-scale UI testing, where a single test suite may be run in parallel on multiple emulators or real devices, including distributed cloud setups.

In addition to dynamic testing, static code analysis is also included in the pipeline to enable early defect and vulnerability detection. Apart from internal linters (for example, detekt for Kotlin, SwiftLint for Swift), SonarQube is utilized to complement quality metrics such as test coverage, technical debt, cyclomatic complexity, and so forth. Such checks prevent code integration that does not meet predefined quality standards, thereby maintaining a high level of code integrity throughout the entire development process.

Thus, the automated testing infrastructure of GitLab CI/CD is a scalable and modular one, which is a part of the whole delivery pipeline of the mobile application. It offers high QA automation rates, facilitates process transparency, raises predictability of releases, and reduces the operational costs by minimizing the number of defects that reach production.

Economic impact of QA automation

The implementation of automated testing within the CI/CD pipeline for mobile development using the GitLab platform has a direct impact on key economic indicators of a project. These include a reduction in Total Cost of Ownership (TCO), an increase in Return on Investment (ROI), and the optimization of personnel costs expressed in terms of Full-Time Equivalent (FTE)-table 2.

Table 2: Economic efficiency metrics for QA automation [3, 4]

Metric	Definition/interpretation	Calculation formula	Application in QA automation
TCO	Total cost of implementing, operating, and maintaining automated testing infrastructure.	TCO = CAPEX + OPEX.	Includes setup costs, infrastructure, licensing, test maintenance, and team training.
ROI	Ratio of financial gain from automation to the cost of investment.	ROI = (automation benefits-implementation costs)/implementation costs × 100%.	Evaluates the profitability of automation; a positive ROI indicates that investment is justified.
FTE	Full-time equivalent hours saved due to automation.	FTE = saved man-hours/average monthly load per specialist.	Measures how much QA capacity is freed up for reallocation to higher-value activities.

The above statistics provide a quantitative basis on which ROI in QA automation in the context of mobile DevOps can be evaluated. However, a comprehensive evaluation requires consideration of not only economic indicators but also qualitative differences between automated and manual processes also need to be considered.

Manual testing is traditionally viewed as a loose and undefined practice, particularly at the start of development or when testing non-compliant user cases. However, once a project expands and the frequency of releases increases, this type of method turns out to be resource-intensive and more challenging to anticipate. Test automation-especially if part of a GitLab CI/CD pipeline-delivers greater stability, speed, and reproducibility in QA cycles. A comparative overview of the two approaches across several critical parameters is presented below (table 3).

Table 3: Manual vs automated testing in CI/CD [5, 6]

Criterion	Manual testing	Automated testing (GitLab CI/CD)
Test coverage	Limited by time and QA team size.	Can reach 80-90% code coverage, depending on test strategy and type (unit, integration, UI).
Reproducibility	Human-dependent, may vary.	Fully reproducible in isolated environments (e.g., containers, runners).
Regression testing time	Typically 1-3 days, depending on scope and manual capacity	30-90 minutes with parallel execution, depending on infrastructure and number of test suites.
Human error risk	High-manual input, missed steps.	Minimized through standardized and repeatable test scenarios.
Team workload	High and repetitive.	Reduced via automated test runs on each commit or merge event.
Initial setup cost	Low.	Requires upfront investment in tooling, infrastructure, and test development.
Long-term maintenance cost	Grows with number of releases.	Varies depending on test complexity; may decrease with reusable test suites.
Release cycle speed	Slower due to manual checks.	Faster-supports daily or multiple daily releases in mature CI/CD environments.
Code change response	Slower adaptation	Immediate feedback via CI-triggered test execution (e.g., via GitLab pipelines).

It should be remembered that the characteristics depicted in the table are wide trends generalized from the behavior of experienced DevOps teams and studies. Actual performance metrics-such as test run duration, extent of coverage, or support cost-may vary depending on project size, system architecture, degree of automation, and tools employed. But the distinctions outlined above are a benchmark against which to compare the two approaches and evaluate the probable impact of test automation within a CI/CD system.

Overall, the presented facts confirm that the implementation of automated testing in mobile development CI/CD pipelines makes sense both from a technical and an economic aspect. Provided that the pipelines are fine-tuned and resource consumption is optimally controlled, automation can actually reduce the overall cost, enhance ROI, and free up human resources for more critical tasks. These advantages position automated testing as a sustainable and strategically sound solution for organizations focused on frequent releases, product quality, and operational efficiency.

Assessment of the impact of QA automation on the cost of the mobile application lifecycle

Test automation has a multifaceted impact on the cost structure throughout the entire lifecycle of a mobile application. The integration of automated checks into a GitLab-based DevOps infrastructure contributes to the reduction of both direct operational expenses and indirect costs associated with rework, bug fixes, and ongoing support. The resulting economic benefits manifest across development, release, maintenance, and update phases of the product lifecycle (table 4).

Table 4: Economic impact of QA automation across the mobile application lifecycle

Lifecycle stage	Impact of automation	Economic effect
Development	Elimination of repetitive tasks (e.g., regression and acceptance testing); redistribution of team roles; fewer rework cycles.	Reduction in manual workload for developers and QA engineers; potentially shorter development cycles.
Release	Improved predictability of releases; automated stabilization checks enhance release consistency.	Lower coordination overhead and increased deployment frequency in CI/CD environments.
Maintenance	Earlier detection of defects reduces the volume of incidents and support tickets.	Decreased post-release support costs; reduction in time and resources required for bug fixing in production stages.

A notable example of the successful use of GitLab CI/CD to accelerate software delivery is the case of Airbus Intelligence, where automation of development and testing processes led to a significant improvement in operational efficiency [7]. Operating in a distributed team environment with a previously fragmented toolchain, GitLab became a unified solution that consolidated deployment scripting, automated testing, and security controls within a single

pipeline. As a result, manual steps for environment setup and testing were eliminated, reducing the release time for a single application from 24 hours to 10 minutes. Currently, 98% of releases are delivered on time, with the remaining ones delayed by no more than a few hours-an outcome that was previously unattainable. The company now runs 17 applications managed through GitLab, and over five years, the speed of feature delivery has increased by a factor of 144, underscoring the strategic impact of integrating test automation and CI/CD practices into the production cycle.

Another indicator of the effectiveness of GitLab CI/CD implementation is the experience of Ally Financial Inc., the largest digital bank in the United States [8]. In a fully online service environment, the timeliness, stability, and security of releases are critical for maintaining and expanding the customer base. Prior to adopting GitLab, each update was accompanied by significant downtime, with developers losing up to 100 hours per month collectively-directly contributing to deployment failures and slower time-to-market. The transition to the GitLab DevSecOps platform enabled centralized pipeline management, reduced the number of tools used, and automated key stages of testing and deployment.

The economic benefits became evident within the first year of transformation. According to internal reports, the company had saved $300,000 a year by reducing downtime and confining third-party support solution expenses. The number of critical failures in the delivery pipeline reduced from 20 to 2 annually, while average deployment time was reduced by 50%. These improvements directly contributed to lowering the TCO of the DevOps infrastructure and increasing ROI by supporting more stable and predictable releases, thus allowing for faster delivery of new features while making it very reliable.

As part of the «Total Economic Impact™ of GitLab Ultimate» research study that Forrester Consulting conducted on behalf of GitLab, the effect of implementing the DevSecOps platform was analyzed in four organizations from diverse industries-financial services, defense, scientific research, and technology [9]. Based on interviews with these organizations, Forrester modeled a composite organization with $5 billion in annual revenue, 5,000 employees (40% in software development), and 50% of revenue derived from IT products. The objective of this composite organization was to increase team productivity, reduce operational costs, and ensure compliance with industry regulations by consolidating fragmented toolchains into a single platform-GitLab Ultimate-integrating automated testing, security, and continuous delivery within the CI/CD process. A summary of the aggregated economic outcomes of GitLab Ultimate adoption in this composite organization is presented in fig.2.

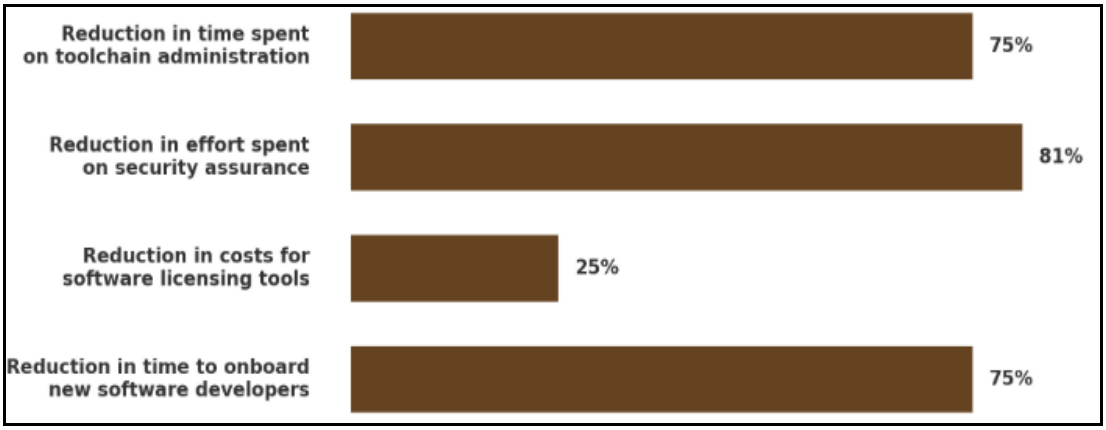

Fig 2: Key indicators of cost and labor reduction following the implementation of GitLab Ultimate (based on Forrester TEI data)

In addition, according to Forrester's findings, developers in the composite organization save an average of up to 305 hours per year thanks to test automation, enabling more frequent validations, faster issue detection and resolution, and reduced tool-switching overhead. The platform also contributes to accelerating new employee onboarding: time to full productivity is reduced fourfold, from 1,5 months to 1,5 weeks. There was also significant benefit in vulnerability management: due to scanning and security embedded, time spent on security tasks was decreased five-fold, and both the mean time to respond to events and the likelihood of vulnerabilities reaching production were significantly reduced. Furthermore, GitLab enables 15-times quicker initial product delivery through synchronized project launches, more frequent releases, and the introduction of security practices into all stages of the software development life cycle.

Therefore, test automation in the GitLab CI/CD pipeline demonstrates consistent cost reduction across all the critical phases of the mobile application life cycle. By optimizing the use of labor, reducing release cycles to a minimum, reducing incidents and failures, and minimizing maintenance expenses, organizations can significantly enhance operational efficiency without compromising product reliability and quality. Real-world implementations show that the impact of automation extends beyond technical improvement, revolutionizing the economics of software delivery through scalability, reusability, and the integration of security into every layer of the production environment.

Conclusion

The integration of automated testing into GitLab CI/CD pipelines has a substantial impact on the economics of mobile application development and maintenance. A GitLab-based infrastructure ensures a high degree of integration between testing tools, quality control mechanisms, containerization technologies, and scalable execution environments. This enables a stable and reproducible software delivery process, contributing to a reduction in TCO, an increase in ROI, and the reallocation

of human resources in terms of FTE, particularly in environments with frequent releases and distributed teams.

Automation delivers measurable benefits at all stages of the application lifecycle-from development to post-release support-including faster release cycles, fewer production defects, and reduced manual testing workloads. The empirical case studies presented demonstrate not only accelerated delivery times but also significant reductions in operational costs and downtime. Therefore, the systematic integration of QA automation into GitLab-based DevOps workflows can be regarded as a strategic enabler for improving the efficiency of mobile development and optimizing the economic performance of digital projects.

References

1. Ruiz ZC, Juan G. The role of DevOps in mobile application development: CI/CD pipelines for faster releases. Information Horizons: American Journal of Library and Information Science Innovation. 2023;1(2):4-12.
2. Koneru NMK. The role of GitLab Runners in CI/CD pipelines: configuring EC2, Docker, and Kubernetes build environments. The Eastasouth Journal of Information System and Computer Science. 2025;2(3):246-271. doi:10.58812/esiscs.v2i03.529
3. Srinivas N, Mandaloju N, Nadimpalli SV. Leveraging automation in software quality assurance: enhancing efficiency and reducing defects. The Metascience. 2024;2(4):84-95.
4. Garifullin R. Optimization of frontend application performance: modern techniques and tools. Professional Bulletin. Information Technology and Security. 2025;3:10-14.
5. Baitha S, Soorya V, Kothari O, Rajagopal SM, Panda N. Streamlining software development: a comprehensive study on CI/CD automation. In: 2024 4th International Conference on Sustainable Expert Systems (ICSES). IEEE; 2024. p. 1299-1305. doi:10.1109/ICSES63445.2024.10763207
6. Nuzhdin D. Performance and fault tolerance of iOS applications: optimization strategies at the architectural and UI thread levels. Cold Science. 2025;19:29-40. EDN: HFYXYR.
7. Airbus takes flight with GitLab, releasing features 144× faster. GitLab [Internet]. [cited 2025 Sep 24]. Available from: https://about.gitlab.com/customers/airbus/
8. Ally Financial cuts pipeline outages and eases security scanning with GitLab. GitLab [Internet]. [cited 2025 Sep 25]. Available from: https://about.gitlab.com/customers/ally/
9. GitLab Ultimate's total economic impact: 483% ROI over 3 years. GitLab [Internet]. [cited 2025 Sep 27]. Available from: https://about.gitlab.com/blog/gitlab-ultimates-total-economic-impact-483-roi-over-3-years/