International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

iOS Senior Software Engineer of FinTech industry

Sosin Vitalii *

Independent researcher, Russia.

International Journal of Science and Research Archive, 2025, 15(02), 1941-1944

Publication history: Received on 19 April 2025; revised on 27 May 2025; accepted on 30 May 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.15.2.1631

Abstract

This article explores the evolution of quality assurance methods for mobile applications in the financial sector. It considers different testing levels, from unit checks of business logic to user interface and API contract scenarios. Special attention is given to post-release quality control tools such as Crashlytics, MetricKit, Sentry, and other systems for monitoring stability and performance. The article emphasizes that ensuring the quality of fintech applications is a continuous process, where preventive testing complements systematic observability in the production environment.

Keywords: iOS; Quality; XCTest; Swift Testing; XCUITest; Crashlytics; MetricKit; Sentry; Performance; DevOps; Mobile banking

1. Introduction

The rise of mobile banking has made application quality control one of the core responsibilities of product development teams. A single error that might go unnoticed in another sector can lead to immediate loss of trust and reputational risk in the banking industry. Market research indicates that a single failure in a banking app can cause customer churn and reduce loyalty. Therefore, quality assurance cannot be a one-time activity; it must be an error-free, continuous process covering all stages of the product lifecycle.

2. Testing as a Cornerstone

iOS development generally relies on a multi-layered testing approach:

2.1. Unit Tests

The primary level of verification is unit testing. Unit tests are used to validate business logic, such as calculations, data formatting, and error handling. XCTest remains the de facto standard, while Swift Testing has recently emerged, enabling even shorter and more efficient tests. This level helps reduce the likelihood of logical errors before the code is integrated into the product.

2.2. API Contract Tests

Fintech applications are closely integrated with backend systems, so ensuring correct API interactions is critical. Verifying response patterns and handling edge cases (timeouts, 5xx errors, invalid data) reduces regressions and operational failures. This approach also enforces discipline across both client-side and server-side teams.

* Corresponding author: Sosin Vitalii

2.3. User Interface Tests

UI-level testing is necessary for critical user scenarios, such as authentication, fund transfers, and transaction confirmations. XCUITest, integrated with Xcode for the iOS platform, is typically used for this purpose. The use of UI tests should be strategic: focus on critical flows where errors would have the highest impact on the user.

2.4. CI/CD Automation

Quality must be integrated into the delivery pipeline. Automated testing, static analysis, linters, and code coverage goals help teams release products confidently without excessive manual verification. According to ThoughtWorks, integrating testing into the release pipeline is one of the key factors that accelerates time-to-market.

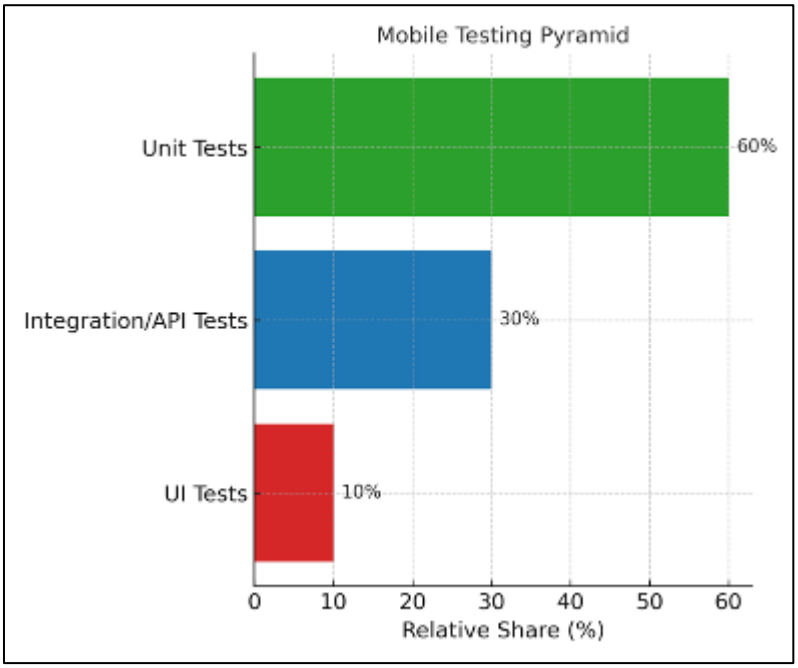

Figure 1 Mobile Testing Pyramid

3. Production Environment Monitoring

Even the most sophisticated testing environment cannot guarantee flawless operation with absolute certainty. Therefore, observability of the application after release is of paramount importance.

3.1. Crashlytics

This tool remains the standard for collecting crash data. Measuring the percentage of users who experience no crashes provides insight into the actual user experience: what proportion of clients can use the app without interruptions. In the fintech sector, it is important to track not only the number of crashes but also their trends across different app versions and user segments.

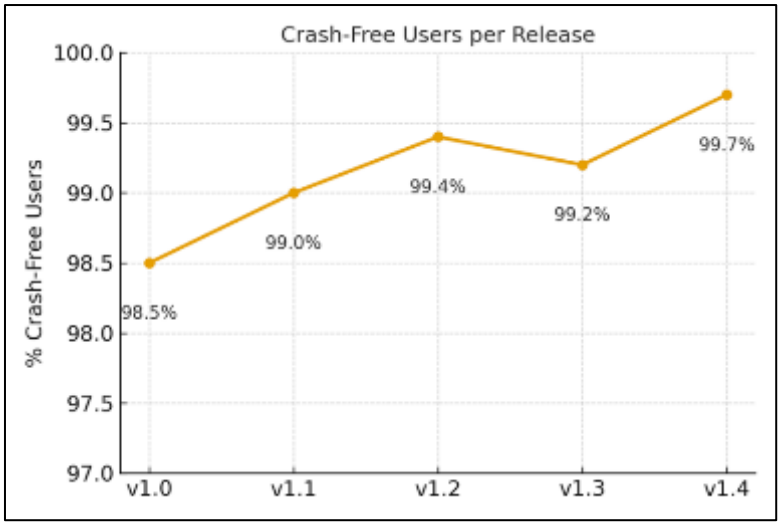

Figure 2 Crash-Free Users per Release

3.2. Apple Architecture Reports

Apple's architecture provides reports on crashes, freezes, energy consumption, and other diagnostics. Its value lies in deep integration at the iOS system level, allowing it to capture scenarios that may be inaccessible to third-party SDKs. Alongside Crashlytics, it offers a more comprehensive view of app stability.

3.3. Sentry and Distributed Tracing

Performance equals stability. Sentry enables tracking of screen load times, API request latency, and interface freezes. This monitoring is useful not only for engineers but also for product management, as it clearly identifies where the bottleneck occurs—whether in the client code or on the server.

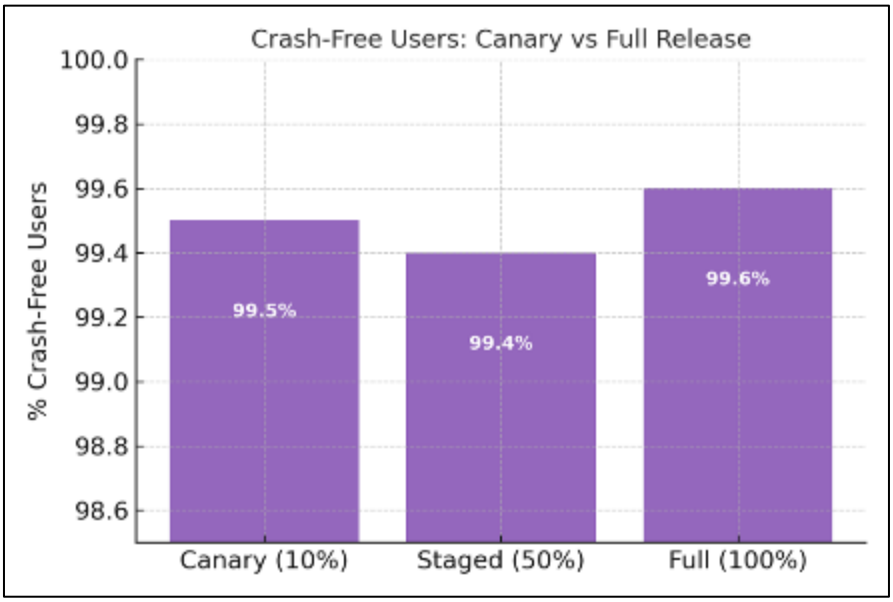

Figure 3 Crash-Free Users: Canary vs Full Release

3.4. Metrics and Thresholds

There are no fixed numerical standards. For some user groups, 99.5% of users experiencing no crashes may be acceptable, while for others, over 99.9% is required. More important are trends and responsiveness to deviations. This is why the "canary release" principle is widely used: a new version is released to a subset of users and tested for stability and performance before it becomes available to all users.

3.5. Quality Culture

Transforming a development culture is impossible without quality control. Engineers should not view testing and monitoring as bureaucracy, but as tools that accelerate work. Sufficient automation eliminates much of the manual testing burden, production alerts allow early detection of issues, and the development team gains confidence in the reliability of releases.

Thus, responsibility for quality falls on everyone, not just a dedicated QA team.

4. Conclusion

The quality of mobile applications in the fintech sector is defined by a combination of two strategies. The first is preventive testing: unit tests, API contracts, catastrophic UI failure scenarios, and CI/CD integration. The second is production monitoring: Crashlytics, MetricKit, Sentry, and tracking key performance metrics. Together, they form an integrated strategy that enables teams to release updates faster while ensuring users can use the application without interruptions.

References

[1] Apple. XCTest — Official Unit Testing Framework. June 2024 (WWDC 2024) https://developer.apple.com/documentation/xctest

[2] Apple. Swift Testing — Modern Approach to Writing Tests. June 2024 (WWDC 2024) https://developer.apple.com/documentation/testing

[3] Apple. XCUITest — UI Testing Framework in the iOS Ecosystem. September 2024 (iOS 18 Release) https://developer.apple.com/documentation/xcuiautomation

[4] Google. Firebase Crashlytics — Crash-Free Users and Crash-Free Sessions Metrics. January 2025 https://firebase.google.com/docs/crashlytics/crash-free-metrics

[5] Apple. MetricKit — Diagnostic Reports on Crashes and Performance. March 2025 (iOS 17.4 / iOS 18 SDK) https://developer.apple.com/documentation/metrickit

[6] Sentry. Tracing & Performance for iOS — Distributed Tracing and Performance Analysis. February 2025 https://docs.sentry.io/platforms/apple/guides/ios/tracing/