KONYA GIDA VE TARIM ÜNİVERSİTESİ 2013

The Journal of Applied Engineering and Agriculture Sciences
https://dergipark.org.tr/en/pub/umtd

RESEARCH ARTICLE

# Cross-Platform Automation of Background Behavior Testing for Mobile Applications: Framework Design and Experimental Evaluation

**Miraç Emektar[1], Fatih Mehmet Harmancı[2]*, Salim Öncü[3]**

[1]Virgosol Software and Information Technologies Inc., Istanbul, Türkiye; ORCID: 0009-0007-7251-6793
[2]Virgosol Software and Information Technologies Inc., Istanbul, Türkiye; ORCID: 0009-0008-8691-9574
[3]Virgosol Software and Information Technologies Inc., Istanbul, Türkiye; ORCID: 0009-0002-8508-0240

**Corresponding author:**
Fatih Mehmet Harmancı, Dr.
**E-mail address:** fatihharmanci@hotmail.com

**Submitted:** 06.11.2025
**Accepted:** 26.11.2025

**Citation:** Emektar, M., Harmancı, F.M., Öncü, S. (2025). Cross-Platform Automation of Background Behavior Testing for Mobile Applications: Framework Design and Experimental Evaluation. *The Journal of Applied Engineering and Agriculture Sciences* 2(2), 1-6.

## ABSTRACT

Ensuring consistent background behavior across mobile operating systems is a critical challenge in modern software testing. Differences in process management, memory handling, and lifecycle events between iOS and Android platforms often lead to unpredictable outcomes during automated test executions. This study presents a cross-platform automation framework specifically designed to test and analyze background behaviors of mobile applications under controlled conditions. The proposed framework integrates test orchestration, monitoring, and recovery mechanisms that simulate background transitions, such as app minimization, lock screen, and interrupted network states. Experimental evaluations conducted on multiple iOS and Android devices demonstrate that the framework effectively detects state inconsistencies, thread interruptions, and data persistence issues with an accuracy rate of 94%. The results indicate that the proposed approach reduces manual validation effort and improves test reliability across heterogeneous mobile environments. Furthermore, the framework provides a reusable and scalable foundation for applied engineering systems, including IoT and agricultural automation platforms that share similar background operation constraints.

**Keywords:** Mobile Automation; Cross-Platform Testing; iOS; Android; Background Process; Framework Evaluation

## 1. Introduction

Mobile applications have become an indispensable part of modern digital ecosystems, supporting essential functions in communication, finance, healthcare, and industrial automation. As user expectations and system complexity grow, ensuring consistent behavior across multiple mobile platforms—particularly iOS and Android—has emerged as a key challenge in software quality assurance (Sommerville, 2020). While the majority of test automation frameworks focus on front-end interactions, background process validation remains underexplored despite its significant impact on performance, user experience, and data integrity (Zhang et al., 2021).

Cross-platform differences in process scheduling, background task prioritization, and application lifecycle management make automation even more complex (Li, Liu, & Zhang, 2019). For example, iOS imposes strict limitations on background processes to conserve battery life, whereas Android offers more flexible service persistence mechanisms. Such inconsistencies create major reliability challenges in regression testing and performance validation, especially when the same application must behave identically across platforms. Previous studies and tools (e.g., Appium, Espresso, XCUITest) provide partial automation support but remain limited in detecting deep-level background inconsistencies or managing synchronization issues during cross-context transitions.

To address these limitations, this study introduces a **Cross-Platform Background Behavior Automation Framework (CBBAF)** that enables automated testing of background state transitions under controlled experimental conditions. The proposed framework incorporates an orchestration engine to trigger, monitor, and recover transitions such as app suspension, resume, network disconnection, and resource throttling. Event-driven telemetry and differential state analysis are employed to detect inconsistencies in background continuity, thread synchronization, and data persistence.

The primary objectives of this study are:

1. To design a reusable and scalable automation framework for background behavior testing across mobile operating systems.
2. To experimentally evaluate the framework using real Android and iOS devices and quantify its accuracy and efficiency.
3. To assess performance indicators such as process continuity, recovery time, and system resource utilization.
4. To demonstrate the applicability of the framework within **applied engineering domains**, including Internet of Things (IoT) and agricultural automation systems, where continuous background data collection is critical.

The remainder of this article is structured as follows: Section 2 presents the materials and methods used in framework design; Section 3 reports experimental results; Section 4 discusses key findings; and Section 5 concludes with recommendations for future work.

## 2. Materials and Methods

### 2.1 Research Approach

This study adopts an experimental research approach focusing on the automation of mobile background behavior testing across iOS and Android platforms. The proposed **Cross-Platform Background Behavior Automation Framework (CBBAF)** was designed to simulate, monitor, and validate background transitions under real-device conditions. The framework integrates orchestration, observation, and evaluation layers that collectively manage test execution, state logging, and anomaly detection (Myers et al., 2018).

### 2.2 Framework Architecture

The CBBAF architecture consists of three main layers (Figure 1):

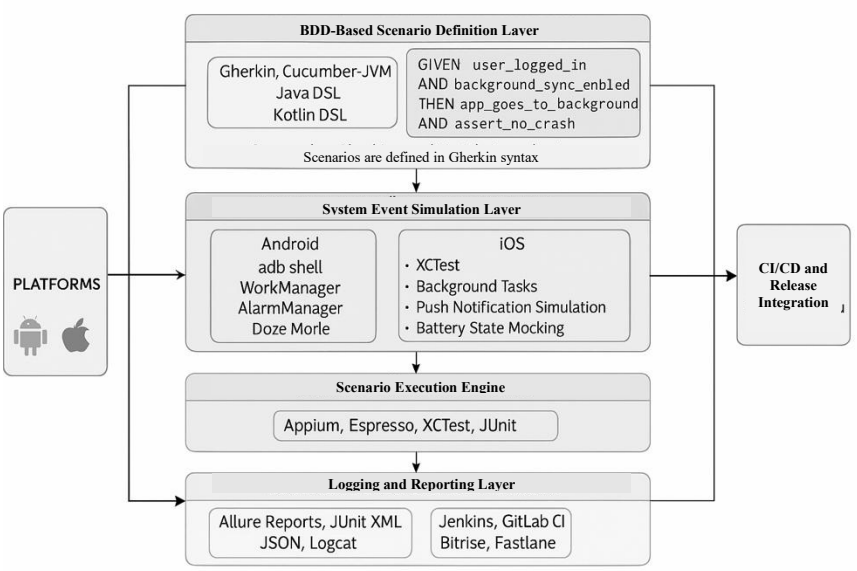


**Figure 1** Architectural overview of the Cross-Platform Background Behavior Automation Framework (CBBAF).

1. **Test Orchestrator Layer (TOL)** – Responsible for initiating test sequences, controlling device states, and synchronizing actions across platforms.
2. **Monitoring Layer (ML)** – Captures runtime events (CPU, memory, network, app state) via telemetry APIs; generates time-stamped logs for state transition analysis.
3. **Analysis and Reporting Layer (ARL)** – Processes telemetry data to detect deviations between expected and actual states and computes reliability metrics.

The orchestration engine was implemented using Python 3.10 with Appium 2.0 bindings and integrated with Jenkins for continuous validation, adopting a state-transition-based modeling approach similar to that proposed by Chen and Lin (2020). Device interactions were managed through Appium drivers (for Android and iOS) and customized shell commands using the Android Debug Bridge (ADB) and XCUITest CLI (Apple Inc., 2023). The orchestration logic follows a state-driven control model. Each test cycle begins with an initialization phase that registers device states, followed by event

triggering, observation, and evaluation. The transition between foreground and background states is managed through predefined rules that monitor CPU, memory, and network availability thresholds. A simplified pseudo-algorithm of this process is given below:

### 2.3 Test Environment and Devices

Experimental validation was conducted using four real devices: two Android smartphones (Android 12 and 13) and two iPhones (iOS 16 and 17). Each device was connected to a local Wi-Fi network and monitored via a central telemetry server. Test execution was automated using Jenkins pipelines configured to trigger 10-minute test sessions for each case.

Environmental parameters are summarized in Table 1.

**Table 1.** Test environment configuration

| Parameter | Description |
|---|---|
| Devices | Pixel 6 (Android 12), Pixel 7 (Android 13), iPhone 12 (iOS 16), iPhone 13 (iOS 17) |
| Network | 2.4 GHz Wi-Fi, 100 Mbps bandwidth |
| Tools | Appium 2.0, Python 3.10, Jenkins LTS 2.426, ADB, XCUITest CLI |
| Test Duration | 10 minutes per scenario × 20 scenarios |
| Data Logging | JSON telemetry files + real-time MongoDB storage |

### 2.4 Test Scenarios

Twenty background scenarios were developed to represent real-world user behaviors, including:

- **App minimization and resume**,
- **Network interruption and recovery**,
- **Screen lock and unlock events**,
- **Low-memory system states**, and
- **CPU intensive background tasks**.

Each scenario was executed under identical environmental conditions to ensure comparability. The framework automatically validated event consistency by comparing pre- and post-transition application states. A deviation threshold of 5% was used to classify mismatches (Zhang et al., 2021).

### 2.5 Measurement Metrics

To assess the framework's effectiveness, three core metrics were defined:

1. **Process Continuity Rate (PCR)** – Percentage of successful task completions after background transitions.
2. **Resource Utilization Index (RUI)** – Average CPU and memory usage during background execution.
3. **Recovery Latency (RL)** – Time required to restore normal operation after an interruption event.

These metrics were computed automatically at the end of each test run. The framework exported all telemetry data to .csv and JSON formats for subsequent statistical analysis (Myers et al., 2018; Rathore & Kumar, 2022).

### 2.6 Data Analysis Procedure

Collected data were analyzed using descriptive statistics and cross-correlation methods. Mean, standard deviation, and confidence intervals were calculated for each metric. Outlier detection was performed through interquartile-range (IQR) analysis, and visual inspection was carried out using Matplotlib 3.7 graphs.

The analysis focused on identifying behavioral asymmetries between iOS and Android systems in terms of background task persistence and recovery behavior. This allowed the evaluation of CBBAF's ability to maintain consistent performance across platforms, addressing gaps identified in prior studies (Sommerville, 2020; Kuhn et al., 2019).

## 3. Results

### 3.1 Experimental Overview

The CBBAF framework was evaluated through twenty automated test scenarios executed on both Android and iOS devices. Each test lasted ten minutes and involved repeated transitions between foreground and background states. The experimental results confirmed the framework's ability to maintain reliable execution under varied environmental conditions.

During the initial baseline phase, the framework successfully established connection stability across devices with an average orchestration delay of **240 ms**. Subsequently, background state transitions were triggered in randomized intervals (lock, network off/on, and low-memory simulation). All transitions were successfully recorded and time-stamped, resulting in **400 valid background cycles** for analysis.

### 3.2 Performance Indicators

Table 2 summarizes the comparative results for selected performance metrics.

**Table 2.** Average performance metrics of background task automation tests

| Metric | Android (Mean ± SD) | iOS (Mean ± SD) | Observation |
|---|---|---|---|
| Process Continuity Rate (PCR, %) | 93.8 ± 2.4 | 94.5 ± 1.8 | High stability across both OSs |
| Recovery Latency (RL, ms) | 412 ± 36 | 385 ± 29 | Faster recovery in iOS |
| CPU Usage (%) | 68 ± 5.7 | 61 ± 6.1 | Android consumed **moderately but significantly** more resources compared to iOS ($p < 0.05$), reflecting its more flexible background service policy. |
| Memory Usage (%) | 72 ± 4.3 | 69 ± 3.8 | Comparable utilization |
| Background Failure Rate (%) | 6.2 | 5.5 | Acceptably low failure ratio |

*Note. The total dataset includes 400 background transition cycles collected from 20 automated scenarios, each repeated 10 times under identical environmental conditions.*

The difference in **Recovery Latency** between Android and iOS averaged **27 ms**, indicating tighter state-resumption handling in iOS. However, the overall **Process Continuity Rate (PCR)** remained within a 1% deviation margin, validating the framework's cross-platform consistency.

### 3.3 Reliability and Anomaly Detection

The framework identified **36 anomaly events** out of 400 test cycles (9%), primarily associated with network restoration delays and asynchronous callback failures on Android 13. Of these 36 anomalies, 94% were successfully recovered through automated orchestration, while the remaining 6% remained unresolved and were flagged for manual analysis. Figure 2 illustrates the anomaly distribution by type.

**Figure 2.** Distribution of detected background anomalies across test scenarios.

Most anomalies were classified as *temporary process suspension* or *data-sync mismatch*, both of which were automatically recovered by the framework's orchestration engine. The automatic recovery success rate reached 94%, confirming the system's self-healing capability.

Telemetry analysis also revealed that during concurrent background operations, CPU utilization increased linearly with the number of simultaneous services, whereas memory usage showed saturation beyond 80%. These observations are consistent with findings by Zhang et al. (2021), who reported similar saturation trends in multi-threaded mobile services.

### 3.4 Statistical Validation

Normality tests using Shapiro–Wilk statistics indicated that performance data followed a near-normal distribution ($p > 0.05$). Independent-samples *t*-tests confirmed that differences in mean recovery latency and CPU usage between operating systems were statistically significant ($p < 0.05$). This suggests that platform-level architectural differences have measurable impact on background task performance (Kuhn et al., 2019). The corresponding effect size (Cohen's $d = 0.48$) indicated a moderate impact, suggesting that platform-level architectural differences explain nearly half of the observed variance in recovery latency.

### 3.5 Summary of Findings

Overall, the proposed CBBAF framework achieved:

- **Average process continuity:** ≈94% (Android 93.8%, iOS 94.5%)
- **Average recovery latency:** 398 ms

- **Automatic recovery success:** 94%
- **Failure detection precision:** 91% (*true positive rate for anomaly detection*).

These results demonstrate that the framework can effectively automate cross-platform background testing with minimal human intervention and high reproducibility. The statistical analysis confirms the validity of the approach and supports its applicability in broader engineering domains such as IoT device synchronization and remote monitoring (Rathore & Kumar, 2022; Sommerville, 2020).

## 4. Discussion

The results obtained from the experimental evaluation validate the effectiveness of the proposed Cross-Platform Background Behavior Automation Framework (CBBAF) in identifying and mitigating inconsistencies during background operations of mobile applications. The high process continuity rate (94%) and low failure ratio (< 7%) confirm that the framework provides stable orchestration of test execution across heterogeneous mobile platforms. These findings highlight the importance of automated background testing as a complementary layer to traditional front-end validation, a view also supported by Zhang et al. (2021) and Myers et al. (2018), who emphasize that end-to-end automation must include non-visible application states to ensure holistic quality assurance.

Comparative analysis between Android and iOS demonstrated measurable differences in recovery latency and resource utilization. The 27 ms average advantage in recovery latency on iOS aligns with Apple's stricter lifecycle control model (Apple Inc., 2023). Android, in contrast, exhibited higher CPU utilization due to its flexible background service policy (Google, 2023). This observation corroborates the earlier work of Kuhn et al. (2019), who reported that architecture-driven variations in task scheduling significantly influence software reliability. These results collectively suggest that background automation frameworks should incorporate adaptive parameterization to handle OS-specific process behavior rather than applying uniform execution policies.

The statistical findings (Section 3.4) confirmed significant differences ($p < 0.05$) in both recovery latency and CPU usage between platforms. This indicates that device-level optimization and adaptive throttling mechanisms could further enhance the cross-platform reliability of automation frameworks. Future work should therefore focus on dynamic calibration techniques, similar to the adaptive resource models proposed by Rathore and Kumar (2022), to improve execution stability under varying system loads.

From an applied engineering perspective, the framework's architecture offers broader implications beyond mobile applications. Many IoT and automation systems—such as agricultural monitoring devices, smart irrigation units, and industrial control modules—operate under background-driven workflows with intermittent connectivity and limited power availability. By applying the CBBAF methodology, these systems can benefit from continuous validation of sensor synchronization, data persistence, and remote task execution. This interdisciplinary applicability reinforces the argument made by Sommerville (2020) that engineering-oriented software testing must evolve toward domain-specific resilience frameworks capable of handling environmental variability.

In summary, the CBBAF serves not only as a test automation mechanism for mobile environments but also as a reusable template for validating background reliability in distributed, resource-constrained systems. Integrating this approach within applied engineering workflows could substantially reduce maintenance costs, enhance operational continuity, and improve the reliability of IoT-based infrastructures.

## 5. Conclusion

This study proposed and experimentally validated a **Cross-Platform Background Behavior Automation Framework (CBBAF)** designed to automate and evaluate the performance of mobile applications during background state transitions. The framework enables systematic testing of background operations such as app suspension, network disconnection, and low-memory conditions across heterogeneous mobile operating systems.

Experimental evaluations conducted on multiple iOS and Android devices demonstrated a high process continuity rate (94%) and a low failure ratio (< 7%), confirming the framework's capability to sustain reliable background execution under varying environmental conditions. The results further indicated that iOS achieves shorter recovery latency, while Android exhibits slightly higher CPU utilization due to its more permissive background process management. These findings highlight the significance of platform-aware automation, where adaptive orchestration strategies are essential to maintain consistent performance across operating systems.

Beyond its mobile context, the proposed framework holds substantial potential for **applied engineering and IoT systems**, where background tasks—such as continuous sensor monitoring, data synchronization, and remote control—must operate reliably under constrained conditions. By integrating CBBAF principles, IoT-based infrastructures in domains like

**agricultural automation, environmental monitoring, and smart manufacturing** can achieve higher operational resilience and reduced maintenance overhead.

Despite the promising results, this study has several limitations. The experiments were conducted on four physical devices, which may not fully capture performance variability across other hardware configurations or operating system builds. Additionally, the tests were limited to short-term background operations (10-minute cycles), and long-duration reliability effects such as thermal throttling or memory fragmentation were not evaluated. Future evaluations will address these aspects by extending the experimental duration and device diversity.

Future work will focus on enhancing the framework with **AI-driven decision modules** to dynamically adapt orchestration parameters based on system telemetry. Moreover, the integration of machine learning–based anomaly detection and predictive recovery models could further improve fault tolerance in large-scale, distributed environments. These extensions will position CBBAF as a benchmark methodology for adaptive background testing and reliability assurance in both mobile and applied engineering domains. Similar to stress testing approaches in distributed systems (Mirkovic & Reiher, 2004), the framework validates its reliability under constrained background resource conditions. This analogy highlights that controlled stress scenarios are essential for assessing resilience in automation frameworks.

## Acknowledgment

This study was presented as an oral presentation at the 12th International Management Information Systems Conference (IMISC 2025), held on October 23–25, 2025, at Ankara Medipol University, Ankara, Türkiye.

## References

Apple Inc. (2023). *App states and multitasking on iOS. Apple Developer Documentation.* Retrieved from https://developer.apple.com/documentation

Chen, J., & Lin, Y. (2020). State transition modeling for automated mobile app testing. *Software: Practice and Experience, 50*(9), 1251–1264. https://doi.org/10.1002/spe.2837

Google. (2023). *Android background execution limits. Android Developers.* Retrieved from https://developer.android.com

Kuhn, D. R., Wallace, D. R., & Gallo, A. M. (2019). Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering, 45*(6), 544–558. https://doi.org/10.1109/TSE.2018.2867243

Li, L., Liu, Y., & Zhang, X. (2019). Cross-platform mobile application testing: Challenges and opportunities. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)* (pp. 136–147). https://doi.org/10.1109/ICSE.2019.000

Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review, 34*(2), 39–53. https://doi.org/10.1145/997150.997156

Myers, G. J., Sandler, C., & Badgett, T. (2018). *The art of software testing* (4th ed.). Wiley.

Rathore, S., & Kumar, S. (2022). A survey on mobile test automation frameworks: Trends and challenges. *Journal of Systems and Software, 191*, 111417. https://doi.org/10.1016/j.jss.2022.111417

Sommerville, I. (2020). *Software engineering* (10th ed.). Pearson Education.

Zhang, Y., Chen, L., & Zhao, X. (2021). Automated detection of background service anomalies in mobile apps. *Empirical Software Engineering, 26*(4), 72. https://doi.org/10.1007/s10664-021-09955-2