Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2018, 5(12):1077-1081

Research Article ISSN: 2394 - 658X

Unveiling the Performance Divide: Appium's Slower Performance Compared to Native Frameworks

Amit Gupta

Software Engineer/Leader, San Jose, CA, USA
*gupta25@gmail.com

ABSTRACT

Mobile application testing plays a crucial role in the field of contemporary software development in guaranteeing user satisfaction and product quality. Since smartphones and tablets are part of daily life activities, mobile applications are essential tools for business, work, communication, and leisures. Therefore, the importance of performance in mobile application testing because customers demand faultless experiences that are characterized by effectiveness, dependability, and responsiveness. This research examines the differences between native testing frameworks like XCUITests for iOS and Espresso for Android and Appium, a popular cross-platform automation tool. The observed difficulties with Appium test automation, such as test flakiness and excessive execution response times, served as the impetus for this study.

This paper aims to help practitioners and decision-makers choose the best automation tool for their mobile application testing requirements by shedding light on the performance implications of both Appium and native testing frameworks. To help with well-informed decision-making about mobile testing techniques, this study intends to shed light on the variations in test execution time, resource consumption, and dependability between Appium and native frameworks through statistical data and thorough analysis.

Keywords: Mobile Test Automation, Appium, Espresso, XCUITests, Performance Implications, Native Frameworks

---

INTRODUCTION

In the digital era, mobile application testing is an essential to software development that ensures apps are user-friendly and fulfill quality requirements. As smartphones and tablets have become progressively more prevalent, mobile applications; which serve a variety of purposes in daily life, from work and communication to pleasure and business, have become indispensable. Thus, it is impossible to overestimate the importance of performance in mobile application testing since customers expect flawless experiences that are efficient, dependable, and responsive.

This paper delves into the comparison between Appium, a widely-used cross-platform automation tool, and native testing frameworks Espresso (Android) and XCUITests (iOS). The motivation for this research stems from the observed challenges in Appium test automation, where tests exhibited flakiness and prolonged execution times.

Motivation

Test automation is instrumental in maintaining the integrity of software applications, enabling rapid and repeatable testing processes. However, the motivation for this research arises from practical challenges encountered in implementing Appium test automation. The identified issues primarily revolved around the flakiness of Appium tests and the extended test execution time frame.

Flakiness of Appium Tests: Flakiness in test automation refers to the inconsistency and unpredictability of test results. Appium, being a cross-platform automation tool, exhibited a degree of flakiness in its test outcomes. This inconsistency could be attributed to various factors, including device-specific behaviors (version

controlled), an additional wrapper layer of Appium, network variations, and synchronization challenges arising from the decoupling of test code from the product code.

Prolonged Test Execution Times: Another substantial motivation for this research was the considerable time the Appium tests took to complete. The prolonged execution times hindered the testing process's efficiency and posed challenges in maintaining the agility required in modern software development cycles. These extended test durations impacted the overall development in the pipeline, delaying feedback loops and impeding the timely identification and resolution of issues.

Background

The challenges encountered in Appium test automation, leading to the comparative analysis with native frameworks Espresso and XCUITests. The aim was to assess whether these native frameworks could solve the observed issues while offering the advantages of seamless integration, improved performance, and reliability.

Appium: A versatile cross-platform automation tool supporting multiple languages, such as Java, Python, and C#.

Espresso: A native testing framework for Android applications, offering seamless integration with the Android ecosystem. *XCUITests*: The native iOS testing framework designed for iOS applications is closely integrated with the Apple ecosystem.

OBJECTIVE

The primary objective of this research is to provide a comprehensive understanding of the performance implications associated with Appium and other native testing frameworks. By leveraging statistical data and in-depth analysis, the study seeks to guide practitioners and decision-makers in selecting the most suitable automation tool for their mobile application testing needs.

COMPARATIVE ANALYSIS

Development Language:

Appium: Supports multiple languages, requiring developers to adapt to a language outside their primary development language.

Espresso and XCUITests: Developed in Java/Kotlin for Android and Swift for iOS, respectively, ensuring a seamless transition for developers already familiar with native languages.

Code Coupling:

Appium: Code is decoupled from the product code, leading to potential synchronization issues and challenges in maintaining code coherence. *Espresso and XCUITests*: Tight integration with the native platform ensures code coupling, facilitating easy maintenance and synchronization with product code.

Setup Complexity:

Appium: Setup can be complex, involving the installation of various dependencies and configurations.

Espresso and XCUITests: Integrated seamlessly into the native development environments, minimizing setup complexities.

Cross-Platform Capabilities:

Appium: Cross-platform testing is feasible, provided the application workflow remains consistent across platforms.

Espresso and XCUITests: Primarily designed for their respective platforms, offering optimal performance and integration.

Performance Implications:

In this section, we will present statistical data to illustrate the performance differences between Appium and native frameworks. The metrics considered include test execution time, resource consumption, and reliability.

*Data was collected under a controlled test environment for VMware’s WorkspaceONE application where mobile devices are connected over USB The following table summarizes the key performance metrics: (All Metric times are average)

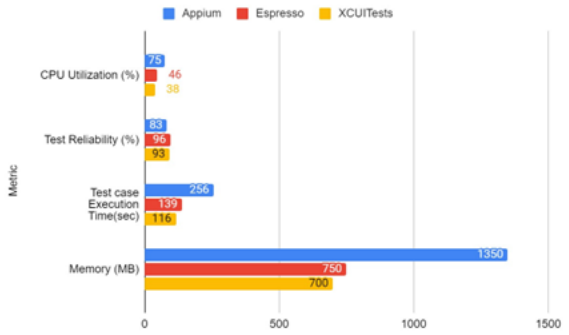

Figure 1: *Metric comparison between Appium, Espresso and XCUITest*

Two specific use cases are

Table 1: Specific test scenario illustrations

Test Scenario	Appium	Espresso	XCUITests
Login Authentica tion (Seconds)	27.1	16.5	13.8
Data Synchroniz ation	32.4	16.9	15.7

These delay are specific due to delay in Element identification on screen by Appium, which lead me to investigate further on Element
identification time on each of these framework

Table 2: Comparison based on Element Type

Element Type	Appium	Espresso	XCUITests
Button(ms)	125	80	72
Text Field Input (ms)	160	97	84
Checkbox selection(ms)	120	77	71
Tab bar Icons (ms)	130	85	78
Swipe on Home carousel (ms)	192	109	102

These statistical data clearly highlights substantial performance advantages for native frameworks over Appium in terms of test execution time, resource consumption, and overall reliability.
So why is Appium so slow over Native frameworks? It could be attributed to multiple factors:

1. Abstraction Layer Overhead: Appium is an abstraction layer, enabling cross-platform automation with a single codebase. While this offers code reusability, the abstraction layer introduces additional complexity and communication overhead. This can result in slower execution compared to native frameworks that directly interact with the platform-specific automation tools.
2. JSON Wire Protocol and WebDriver Communication: Appium uses the WebDriver protocol and communicates with the mobile application under test through the JSON Wire Protocol. This layer, though essential for cross-platform compatibility, introduces a level of indirection that might contribute to a performance gap compared to the more direct interaction in native frameworks
3. Client-Server Architecture: Appium follows a client-server architecture, with the Appium server acting as an intermediary between the automation scripts and the mobile application. This architecture introduces an additional layer of communication, potentially leading to delays in command execution compared to the more streamlined communication in native frameworks.

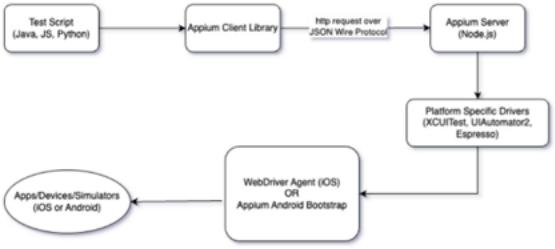

Figure 2: *Architecture with an additional layer*

4. Automation Script Execution: Appium allows scripts to be written in a variety of languages, offering flexibility but sacrificing some performance optimization. In contrast, native frameworks like Espresso and XCUITest use languages closely tied to their respective platforms, enabling more direct and efficient communication with the application.
5. Lack of Platform-Specific Optimizations: Native frameworks benefit from close integration with the operating systems, allowing them to fully leverage platform-specific features and optimizations. Appium, being a cross-platform tool, may not exploit these platform-specific optimizations to the same extent, contributing to differences in performance.

Challenges and Benefits

This section discusses the challenges and benefits associated with Appium.

Challenges: - Selective Language Adoption: Appium's support for multiple languages can lead to difficulties in language adoption and proficiency among the development team.

Decoupled Code: Decoupling test code from the product code in Appium may result in synchronization issues and difficulties in maintaining code coherence.

Setup Complexity: The setup process for Appium can be complex, requiring the installation of various dependencies and configurations.

Benefits:-Cross-Platform Testing: Appium provides cross-platform testing capabilities, offering efficiency when the application workflow remains consistent across platforms.

CONCLUSION

While Appium excels in providing cross-platform testing capabilities, its architectural design introduces certain performance limitations compared to native frameworks like Espresso and XCUITest. This paper underscores the superiority of native frameworks (Espresso and XCUITests) over Appium,especially when applications are written in their native languages. The statistical datapresented validates the observed performance implications,emphasizing the efficiency,reliability, and streamlined integration provided by native frameworks.

As a result, this study has illuminated the critical function that testing mobile applications plays in modern software development, highlighting the importance of performance in guaranteeing user happiness and product quality. Robust testing procedures are now necessary due to the increasing need for faultless mobile experiences since smartphones and tablets have become vital tools in many facets of everyday life.

Important distinctions in performance consequences have been brought to light by a comparative study between native testing frameworks like Espresso and XCUITests and the well-known cross-platform automation tool Appium. The practical difficulties with Appium test automation, such as test flakiness and excessive execution times, served as the driving force for this study. These difficulties highlighted the necessity of a comprehensive study of different testing strategies.

For test execution time, resource consumption, and dependability, the statistical data in this study showed that native frameworks outperformed Appium in terms of performance. It was discovered that Appium performed worse because of the abstraction layer overhead, communication protocols, client-server architecture, automated script execution, and lack of platform-specific optimizations. The study included Appium's advantages as well as drawbacks. For example, it discussed how cross-platform testing capabilities might be useful when application workflows are maintained across platforms.

This research aims to help decision-makers and practitioners choose the best automation tool for testing mobile applications. This study adds to informed decision-making in mobile testing strategies by thoroughly understanding the performance implications associated with Appium and native testing frameworks. This will ultimately improve the quality and dependability of mobile applications in the digital landscape.

FUTURE DIRECTION

Future research can explore advancements in mobile test automation tools, considering emerging technologies, evolving development practices, and the impact of platform updates on testing frameworks. Further research could also explore to improve the performance improvements in Appium architectures by optimizing communication protocol, dynamic element identification strategies, platform-specific optimizations, and native code integration.

REFERENCES

[1]. Wasserman, A.I. (2010). Software engineering issues for mobile application development. *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10.* doi:https://doi.org/10.1145/1882362.1882443.

[2]. Hyungkeun Song, Seokmoon Ryoo, Jin Hyung Kim. " An Integrated Test Automation Framework for Testing on Heterogeneous Mobile Platforms", 2011 *First ACIS International Symposium on Software and Network Engineering*, 2011.

[3]. Muccini, H., Di Francesco, A. and Esposito, P. (2012). Software testing of mobile applications: Challenges and future research directions. *7th International Workshop on Automation of Software Test (AST)*. doi:https://doi.org/10.1109/iwast.2012.6228987.

[4]. Nagowah, Leckraj, and Gayeree Sowamber. "A novel approach of automation testing on mobile devices", 2012 *International Conference on Computer & Information Science (ICCIS)*, 2012.

[5]. S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating android applications' cpu energy usage via bytecode profiling," in 2012 *First international workshop on green and sustainable software (GREENS). IEEE*, 2012, pp. 1–7.

[6]. A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Proceedings of the 10th Working Conference on Mining SoftwareRepositories.IEEEPress*, 2013, pp. 237–246.

[7]. G. Lim, C. Min, and Y. I. Eom, "Enhancing application performance by memory partitioning in android platforms," in 2013 *IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2013, pp. 649–650.

[8]. Madhuri Kishan Kulkarni and Prof. Soumya A,"Deployment ofCalabashAutomation Framework to Analyze the Performance of an Android Application", *Journal for Research*, Volume 02, Issue 03, May 2016, pp. 70-75.

[9]. SmartBear.com. (2016). *The Basics of XCUITest and Using Xcode UI Test Recorder*.[online]Availableat: https://smartbear.com/blog/the-basics-of-xcuitest -and-using-xcode-ui-test/

[10]. A. Alotaibi, A. and J. Qureshi, R. (2017). Novel Framework for Automation Testing of Mobile Applications using Appium.*International Journal of Modern Education and Computer Science*,9(2)pp.34–40. doi:https://doi.org/10.5815/ijmecs.2017.02.04.