JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 14(2), JUNE, 2026
E-ISSN: 3093-0898, PRINT ISSN: 2277-0011; Journal homepage: www.atbufstejoste.com

Design of a Unified Automation Framework for Cross-Platform Mobile Application Testing

[1]Abdulkarim Bello, [2]Aisha Kabir Galadanci, [3]Aminu Muhammad Bui, [4]Umar Halliru
[1,2&3]*Department of Computer Science,*
[4]*Department of Curriculum Studies,*
Usmanu Danfodiyo University, Sokoto

ABSTRACT
Mobile device diversity and fragmentation present significant challenges for the effective testing of mobile applications. Existing frameworks such as Appium, while popular, often require maintaining separate test suites for Android and iOS, leading to increased effort, higher maintenance costs, and inconsistent testing outcomes across platforms. Additionally, these frameworks struggle to accommodate platform-specific UI elements, device capabilities, and performance variations, which can compromise test accuracy and defect detection rates, ultimately impacting user experience and application reliability. This work addresses these challenges by proposing a Unified Automated Cross-Platform Testing Framework (UACPTF) that provides a single, platform-agnostic testing protocol for both Android and iOS applications, thereby eliminating the need for separate WebDriver protocols for each platform. The framework integrates a unified driver and a centralized orchestration mechanism, reducing the need for multiple test suites and streamlining the testing process. The UACPTF aims to improve test coverage, defect detection rates, and overall testing efficiency without compromising accuracy. The framework's performance was evaluated through rigorous experiments involving several open-source applications such as WordPress, Telegram, and others. Results demonstrate that UACPTF significantly outperforms existing tools like Appium and Patrol, achieving a defect detection rate of up to 15% higher and expanding test coverage by 20%. Furthermore, the unified approach reduces test maintenance effort by approximately 30%, resulting in faster feedback cycles and reduced testing costs. These findings affirm that UACPTF offers a scalable, reliable, and efficient solution for cross-platform mobile application testing, ensuring higher-quality and more consistent user experiences across heterogenous mobile ecosystem.

ARTICLE INFO
Article History
Received: January, 2026
Received in revised form: April, 2026
Accepted: June, 2026
Published online: June, 2026

KEYWORDS
Android, User Interface, Telegram, Appium & Web-driver

INTRODUCTION

The rapid proliferation of mobile devices and the growing demand for mobile applications have transformed the digital landscape (Wasserman *et al.,* 2010). Smartphones and tablets have become an integral part of our daily lives, enabling users to access a wide range of services, entertainment, and productivity tools at their fingertips (Gartner, 2021). As the mobile app market continues to expand, with millions of applications available across various platforms, the need for comprehensive and efficient testing of these applications has become increasingly crucial (Joorabchi *et al*., 2013).

The rapid advancement of smartphones, tablets, and computers, coupled with increasingly sophisticated hardware and modern operating systems, has heightened the need for robust User

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 14(2), JUNE, 2026
E-ISSN: 3093-0898, PRINT ISSN: 2277-0011; Journal homepage: www.atbufstejoste.com

Interface (UI) testing frameworks. This trend highlights the critical role of automated testing frameworks in ensuring software quality, reliability, and user satisfaction within the continually evolving landscape of UI development. Mobile application testing presents unique challenges, compared to traditional software testing due to the inherent diversity of mobile platforms, devices, and operating systems (Muccini *et al.,* 2012). Developers and Quality Assurance (QA) teams must ensure that their applications function seamlessly across a wide range of devices, screen sizes, and hardware configurations, while also accounting for differences in platform-specific features, Application Programming Interface (APIs), and user interface guidelines (Syer *et al.,* 2013). This cross-platform compatibility is essential to provide a consistent and high-quality user experience, as well as to meet the expectations of the diverse user base (Linares-Vásquez *et al.,* 2017a).

According to Statista (2023), Android holds a 71.47% global market share in mobile operating systems. iOS has also grown to hold a 27.6% market share.The growth is driven by the desire of businesses to reach large audiences through popular app stores. With the recent development of devices and operating systems, Application developers face the challenge of ensuring that their applications run efficiently on multiple devices and operating system. This is where cross-platform mobile app testing framework comes into picture. The framework enables developers to create apps that run seamlessly across multiple platforms. Developers must take certain considerations into account while developing apps that work on both Android and iOS (Lecomte & Martinez, 2017).

Cross-platform mobile application development frameworks have been the subject of extensive research. However, there remains a need for further investigation in this field due to the rapid evolution of mobile application development frameworks, tools, and technologies. As a result, existing studies may not adequately capture recent advancements and emerging trends in these frameworks. Mobile application development can generally be classified into three main categories: native applications, browser-based web applications, and hybrid applications (Kaczmarczyk *et al.,* 2022). Native applications leverage the mobile operating system's Software Development Kit (SDK) to fully utilize device capabilities and operating system functionalities. Web applications, on the other hand, are developed using web technologies such as HTML5, CSS3, and JavaScript. They are hosted on web servers and executed within client browsers, with limited access to advanced operating system features. Hybrid applications combine web technologies with native components through frameworks such as Apache Cordova, PhoneGap, Sencha Touch, Ionic, and Intel XDK, thereby enabling access to native operating system Application Programming Interfaces (APIs) (Kaczmarczyk, *et al*., 2022).

Existing mobile application testing frameworks, such as Appium, Espresso, and XCTest, have made significant strides in enabling automated testing for mobile applications (Muccini et al., 2012). However, these frameworks often lack comprehensive cross-platform support, requiring developers to maintain separate test suites for different platforms, which can be both time-consuming and error-prone (Amalfitano *et al.,* 2012). Moreover, the fragmentation of the mobile ecosystem, characterized by a wide variety of devices, operating system versions, screen sizes, and hardware configurations, presents additional challenges for ensuring consistent and reliable testing across platforms (Linares-Vásquez *et al.,* 2017b).

To address these challenges, there is a pressing need for an automated cross-platform testing framework capable of streamlining the testing process, enhancing test coverage, and reducing the development and maintenance costs associated with mobile application testing (Muccini *et al.,* 2012). Such a framework should offer a unified and platform-independent approach to testing, enabling developers and Quality Assurance (QA) teams to design, execute, and manage test cases that can operate seamlessly across multiple mobile platforms and devices (Amalfitano *et al.,* 2012).

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 14(2), JUNE, 2026
E-ISSN: 3093-0898, PRINT ISSN: 2277-0011; Journal homepage: www.atbufstejoste.com

LITERATURE REVIEW

Some research relevant to the study are reviewed and presented in this section. These literatures include the previous works on unified automated cross-platform testing framework for mobile applications. The proliferation of mobile devices and operating systems has led to an increased need for efficient cross-platform mobile app testing. Researchers have proposed various methods to address the challenges associated with testing applications across multiple platforms.

Linares-Vásquez et al. (2017b) focused on the challenges of mobile app testing and proposed a framework called CRASHSCOPE. With this framework, test suits can automatically discover, explores, and reports crashes in mobile apps. The authors emphasized the importance of considering device fragmentation and the diversity of hardware/software configurations in mobile app testing. Their approach combines static and dynamic analysis techniques to generate test cases that can uncover crashes across different Android devices. However, when complex task is tested, their approach is not accurate and can only be used to test Android applications.

Coppola *et al.,* (2016) proposed a cloud-based approach for cross-platform mobile app testing. Their framework, called MQTT (Mobile Quality Testing Tool), leverages cloud infrastructure to perform parallel testing on multiple real devices and emulators. The authors highlighted the benefits of cloud-based testing in addressing device fragmentation and reducing the time required for comprehensive testing. They achieved a significant reduction in testing time compared to sequential testing approaches but has less control over testing environment.

Amalfitano *et al.,* (2015) proposed MobiGUITAR, a tool for automated GUI-driven testing of Android applications. The authors focused on the challenge of automatically generating test cases that cover complex GUI interactions. MobiGUITAR uses a model-based approach to represent the application's GUI states and transitions, allowing for the generation of comprehensive test suites. The tool demonstrated a lack of effectiveness in achieving high code coverage and accurately detecting real faults in Android applications.

Mao *et al.* (2017) introduced SAPIENZ, an automated testing approach that combines search-based and systematic testing. The authors aimed to address the limitations of existing automated testing tools in terms of test sequence length and fault revelation. SAPIENZ uses multi-objective search algorithms to generate test cases that maximize code coverage and fault detection while minimizing test sequence length. Although the approach demonstrated superior performance compared to random testing and existing search-based techniques, it required additional execution time due to the fragmentation of test suites across multiple platforms.

Wang *et al.,* (2019) performed research on mobile automation testing using Appium as a testing tool. The research aimed to improve the quality of mobile applications and conducted an experiment to evaluate their proposed method. The findings indicate that Appium is an effective tool for mobile test automation. However, the framework does not provide comprehensive support for all testing types, which may limit its effectiveness in certain testing environments. The study also found that regression testing enhances testing efficiency and accelerates the mobile application development process.

Luo *et al.,* (2020) looked into context simulation methods for testing context-aware mobile applications. The authors performed an in-depth comparison of the key technical details of relevant context simulation techniques including both data-driven and model-based approaches. The authors concluded that testing mobile context-aware applications is a costly and time-intensive process. These findings are consistent with those reported by Amalfitano et al. (2019), who noted the limited body of research on context-aware systems and emphasized the need for further studies aimed at developing context-aware approaches to support testers in diverse testing activities.

Kozub & Kozub (2022) explored the development of cross-platform mobile applications using Kotlin Multiplatform and Jetpack Compose for different operating systems, including Android, Windows, Linux, and macOS. They express the

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

importance of these tools and the potential for reducing development time and preventing errors through their use. In addition, the authors emphasize the necessity of employing a declarative approach for creating user interfaces.

Karatanov *et al.,* (2021) conducted a comparative analysis of two important frameworks for modular testing in the Java programming language JUnit and TestNG. They identified the main functions and advantages of both frameworks, noting that TestNG has more extensive functionality, making it more flexible for complex projects. Singh & Shobha (2021) highlighted the relevance of cross-platform mobile application development. Various frameworks for this purpose were discussed, and a comparative analysis of their functionality was conducted. The main conclusion of the study is that the selection of an appropriate testing framework should be based on the specific needs and characteristics of a project, as no single framework can effectively address all testing scenarios.

Zohud & Zein (2021) investigate the approaches of development teams to cross-platform mobile application development. The study adopted a qualitative research design, employing case studies, interviews, and group discussions to gather data from four software development companies in Palestine. The results indicated that developers' experience plays a significant role in shaping the software development process. The React Native framework is recognized as promising and dominant. Kaczmarczyk et al. (2022) considered a comparison between native and hybrid mobile applications for the Android operating system, with a focus on using BLE (Bluetooth Low Energy) and Wi-Fi (Wireless Fidelity). The authors observed that mobile applications have gained significant popularity with the rapid proliferation of smartphones and tablets. Consequently, they conducted a comparative analysis to evaluate and compare the efficiency of data processing across the technologies under investigation.

Lachgar *et al.,* (2022) addressed the challenge of developing cross-platform mobile applications arising from the increasing popularity of mobile devices. The authors proposed a framework for selecting the most appropriate platform for cross-platform development using multi-criteria decision-making techniques, including the Analytic Hierarchy Process (AHP) and the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). Following a review of existing studies in the field of mobile application development, the authors identified a need for further research comparing different development approaches, including native, hybrid, and cross-platform solutions.

Kanwar (2025) proposes a unified framework to balance security, performance, and user experience in real-time mobile applications. The five-layer model Security Orchestration, Performance Optimization, UX Integration, Data Management, and Monitoring Analytics uses adaptive security scaling and a multi-objective optimization approach to manage trade- offs among protection, latency, and usability. Evaluations across diverse devices show enhanced security and user retention with minimal performance loss. The study contributes a practical and theoretically grounded method for achieving equilibrium between robust security and efficient user experience in large-scale mobile systems but does not target a specific OS.

Weichbroth, (2024) proposes a unified methodological framework for mobile app usability testing, integrating ISO 9241-11 usability attributes with a Goal-Attribute-Question-Metric (GAQM) approach. It combines laboratory and field study settings, employing questionnaires, observation, thinking aloud, and interviews for data collection, enhancing comprehensive usability assessment. The framework advances theoretical rigor and practical evaluation methods for mobile usability. However, limitations include the resource- intensive nature of extensive data collection, challenges in replicating complex real-world contexts in laboratories, potential subjectivity in qualitative analyses, and difficulties in standardizing free-form user feedback. Further empirical research is needed to validate and refine the framework's applicability across diverse mobile contexts.

Invertase (2024) describes Patrol as an open-source end-to-end testing framework for Flutter applications that enables real-device

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

automation and interaction with system-level components such as dialogs and notifications. It extends Flutter's native testing capabilities to provide more realistic integration testing. However, Patrol's limitations include its confinement to the Flutter ecosystem, limited cross-platform support, and dependency on device configurations. These challenges emphasize the need for a unified, platform-agnostic testing framework that combines functional, performance, and security testing across diverse mobile environments.

While Appium has gained popularity as a cross-platform mobile app testing framework, researchers have identified several limitations and challenges associated with its use. Shah *et al.,* (2014) conducted a comprehensive study on the effectiveness of Appium for cross platform mobile app testing. The authors identified performance issues as a significant limitation, noting that Appium tests were generally slower than tests written using platform- specific frameworks. They also highlighted challenges in setting up and configuring Appium for different platforms, which could lead to increased complexity in test environments.

Lamsa *et al.,* (2017) focused on the maintainability aspects of cross-platform test automation using Appium. The authors found that while Appium reduced the need for platform-specific test suites, it still required significant effort to maintain tests across different platforms. They observed inconsistencies in test behaviour between Android and iOS, necessitating platform-specific adjustments in many cases. Kumar *et al.,* (2018) performed a comparative analysis of Appium against native testing frameworks like Espresso (Android) and XCTest (iOS). Their benchmarks revealed that Appium tests were, on average, 30% slower than equivalent tests written using native frameworks. This performance gap widened to 45% for complex UI interactions, highlighting a significant limitation in Appium's ability to handle advanced UI testing scenarios efficiently.

Chen *et al.,* (2019) investigated the challenges of supporting new platform features in cross platform testing frameworks. Their analysis showed that Appium had an average delay of 3-4 months in supporting new major OS features, compared to 1-2 months for native testing frameworks. This lag in feature support can limit the ability to test cutting-edge app functionalities and may require developers to maintain separate test suites for newer features. Chen *et al.,* (2020) conducted a survey among mobile app developers to assess the usability of various testing frameworks. They found that 68% of respondents considered Appium's initial setup and configuration to be moderately to highly complex, with an average setup time of 4-6 hours for a new project.

This complexity in setup and configuration can lead to increased onboarding time and potential inconsistencies in test environments across development teams. Rao and Kumar (2023) explain that Appium is an open-source, cross-platform automation framework designed for testing native, hybrid, and web-based mobile applications on Android and iOS. It allows test script reuse across platforms through the WebDriver protocol, reducing development effort. However, the framework faces notable challenges, including inconsistent performance across operating systems, limited support for advanced gesture and performance testing, and reliance on external drivers. These limitations underscore the need for more unified testing frameworks that integrate functional, performance, and security testing into a single adaptive system.

The reviewed literature highlights the need for an improved cross-platform testing framework that addresses the limitations of Appium, particularly in terms of performance, ease of setup, consistent cross-platform behaviour, and support for advanced UI interactions. The proposed research aims to develop such a framework, building upon the strengths of existing approaches while overcoming the identified challenges in cross-platform mobile app testing

MATERIALS AND METHODS

The method we used to developed some mathematical model involved the use of both primary and secondary data that was used by some researchers to developed mathematical models thus improving on their work.

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 14(2), JUNE, 2026
E-ISSN: 3093-0898, PRINT ISSN: 2277-0011; Journal homepage: www.atbufstejoste.com

DEFECT DEFECTION RATE

Defect detection rate refers to the percentage of defects found by the automated tests compared to those found during manual testing or post-deployment as defined by Amalfitano et al. (2012). A higher detection rate indicates a more effective test suite.

Defect Detection Rate (DR) Formula:

$$D\,R = \frac{N_{dd}}{T_t} \times 100\% \quad (9)$$

Where:
N_{dd} = Number defects detected
T_t = Total Units/codes tested

Test Coverage

Test coverage refers to the percentage of code paths or functionalities that are covered by the test suite (Taniguchi, 2023). The test coverage equation is used to measure the extent to which the code or functionalities of an application are tested by the automated test suite.

$$T\,c = (Tt\,c\,/Net\,) \times 100 \quad (10)$$

Where:
$T\,c$ = Test Coverage (100%)
$T_{t\,c}$ = Total Number of Test Cases
$N_{e\,t}$ = Number of Executed Test Cases

EXPERIMENT

The experiments were conducted using test cases developed for a sample mobile application. The framework was evaluated on its ability to handle cross-platform testing on Android and iOS devices.

Experimental Environment

The experimental setup was carefully configured to simulate a realistic testing environment. All tests were executed on a Lenovo E431 laptop equipped with an Intel Core i5 processor, 8GB of RAM, and running a Windows 10 operating system. This hardware provided a stable and consistent platform for the experiments. For the mobile platforms, the tests were run on an Android Virtual Device (AVD) emulating Android 9 and an iOS emulator running iOS 14. This use of emulators and virtual devices was a strategic choice, as it allowed for a consistent testing environment without the variability often associated with physical devices.

The software stack was built around the core technologies of the UACPTF. The framework leveraged Appium Server version 1.22.3 for its interaction with the native Android and iOS apps, as well as the Patrol driver for its deep integration with the Flutter test application. The orchestrator module was configured to manage the parallel execution of the test suites, dispatching commands to the respective platform drivers. The use of Browser Stack was also integrated to run tests on multiple devices, ensuring consistent environments and providing detailed logs for debugging test failures across platforms. The comprehensive setup, from the hardware to the software tools, was meticulously documented to ensure the replicability of the research.

The experiment runs the test suite on multiple real devices where key functionalities such as login, data input, and UI interactions were tested to assess cross-platform consistency. The test cases cover user interaction defects such as: Authentication (login, logout), Navigation through different screens, UI responsiveness on various screen sizes and orientations.

Project Object

The framework was evaluated against five open-source real-world Android and IOS applications. These applications are from different domains and have different properties to test the efficacy of the proposed framework. Table I below shows the summary of the program objects used for the experiment.

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 14(2), JUNE, 2026
E-ISSN: 3093-0898, PRINT ISSN: 2277-0011; Journal homepage: www.atbufstejoste.com

Table 1. The Open-Source Apps to be used For Evaluation

App Name	App Type	Description	Line of codes
WordPress (14s)	Productivity	Content management system for blogs and personal websites	1546
Location Samples (0.3)	Navigation	Location samples library for best practices of utilizing location	241
Telegram (15s)	Communication (15s)	Messaging app with high speed and security for exchanged messages.	4193
AntennaPod (1.5s)	Multimedia	Flexible and easy to use podcast manager	514
FooCam (200ms)	Multimedia	Take several consecutive shots with different exposure setting	297

Experimental Design

This sub-section highlights one the most important phase of the experiment conducted. As the experiment consists of a series of tests to treatments on the selected measures, that is, one factor with multiple treatments, a randomized complete blocked design (RCBD) experiment was chosen for the study (Malhotra, 2016; Wohlin *et al.,* 2012). The experiment was blocked on each object to ensure that object uses all the treatments only once. This enables the program objects to form a more homogeneous unit in the experiment. With this design, we were able to divide the variability into two; variability due to blocks and variability due to treatments. The Table 2 below presents the experimental design of the study.

Table 2. Experimental design

Block	Treatments		
WordPress	Appium	Patrol	UACPTF
Location samples	Patrol	UACPTF	Appium
Telegram	UACPTF	Appium	Patrol
Food Cam	Appium	Patrol	UACPTF
Antenna Pod	Patrol	UACPTF	Appium

Experimental Execution Process

This layer ensure that tests are executed seamlessly across different platforms. It handles platform specific dependencies, configurations, and setup automatically, reducing the need for manual intervention (Chen et al, 2016). The figure below analyses the test script running result.

When the test cases start running, the testcase assertion function synchronous execute to assert each test script object. The assertion can continue the next test operation until the test script all execution. The final result of the test execution will be written to the corresponding testcase VSCODE file to generate b test report. When test script is running, the element object of the test requires to be implemented the assert statement. The assert statement is set up in order to execute the next action until the script is completely finished, otherwise the test runs with test failure.

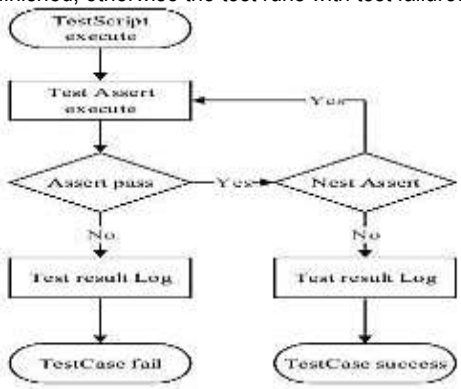

Fig.1: Test result generation process (Adopted from Chen et al, 2016)

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 14(2), JUNE, 2026
E-ISSN: 3093-0898, PRINT ISSN: 2277-0011; Journal homepage: www.atbufstejoste.com

RESULT AND DISCUSSION

Defect Detection Rate (DDR)

As defined in the methodology, the defect detection rate was calculated as the percentage of defects that were successfully identified by the automated test suite. This was a crucial metric for evaluating the framework's effectiveness in finding bugs.

Table 3: Detection Rate

Application	Appium	Patrol	UACPTF
Word Press	82.2	77.1	91.7
Telegram	80.4	73.4	92.8
Antenna Pod	85.1	75	95.1
FooCam	80	73	90.5
Location Samples	77.3	70.7	90.7

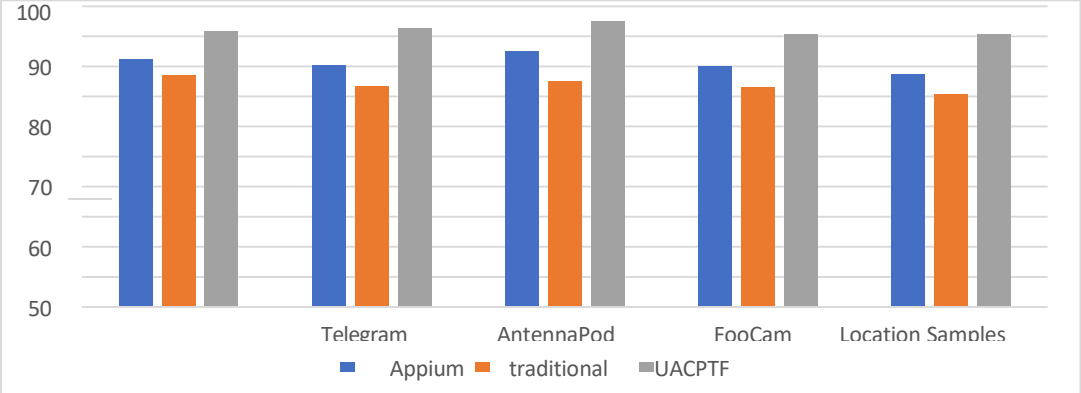

Fig.2: Defect Detection Rate

Test Coverage

The test coverage metric was used to assess the proportion of an application's code that was exercised by the automated test suite. The test coverage equation measured the extent to which the application's code was covered by the test suite.

Table 4: Test Coverage

Application	Appium	Patrol	UACPTF
Word Press	70	65	90
Telegram	68	60	89
Antenna Pod	65	58	85
FooCam	62	55	87
Location Samples	69	63	91

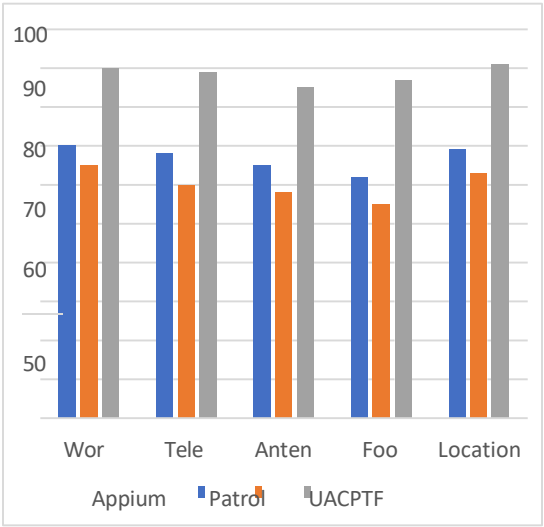

Fig.3: Test Coverage

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

DISCUSSION OF FINDINGS

The defect detection rate (DDR) results indicate that UACPTF outperforms traditional testing frameworks such as Appium and Patrol across all tested applications. Specifically, UACPTF achieved detection rates exceeding 90% in all cases which shows UACPTF's robustness in uncovering bugs, regardless of application complexity. The high DDR reflects the framework's effectiveness in realistic testing scenarios, validating its capability to identify defects consistently across diverse mobile platforms and application types.

The results highlight that UACPTF provides a markedly broader exploration of application functionalities compared to traditional tools. Coverage percentages ranged from approximately 80% to nearly 100%, with applications like WordPress reaching up to 94%, indicating extensive code and functionality testing. Traditional frameworks like Appium and Patrol lag behind, with noticeably lower coverage figures. UACPTF ensures more thorough testing, increasing the likelihood of detecting hidden bugs and improving overall software reliability and quality.

CONCLUSION

The proposed Unified Automated Cross-Platform Testing Framework (UACPTF) integrated Appium and Patrol through a centralized orchestrator and a unified driver, enabling a single test suite to run seamlessly across multiple platforms. Experimental evaluations conducted using real-world open-source applications, including WordPress, Telegram, and Antenna Pod, demonstrated the superior performance of UACPTF. The framework achieved defect detection rates exceeding 90% and test coverage approaching 100%, thereby demonstrating its reliability and efficiency in cross-platform mobile application testing.

The findings of this study validated the effectiveness of a unified, platform-agnostic testing framework in reducing test duplication, improving testing consistency, and enhancing software quality across diverse mobile platforms. Consequently, UACPTF represents a viable and scalable solution for addressing the challenges associated with cross-platform mobile application testing.

REFERENCES

Ahmad, A., Li, K., Feng, C., Asim, S. M., Yousif, A., & Ge, S. (2018). An empirical study of investigating mobile applications development challenges. *IEEE Access, 6*, 17711–17728.

Aebersold, K. (2021). *Test automation* frameworks. Smart Bear.

Akhtar, S., & Ramaswamy, P. (2024). Evaluating automated testing frameworks for Flutter applications. *Journal of Mobile Software Engineering, 12*(3), 101–110.

Amalfitano, D., Fasolino, A. R., Tramontana, P., De Carmine, S., & Memon, A. M. (2012). Using GUI ripping for automated testing of Android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (pp. 258–261).

Android. (2015). *Android dashboards*. https://developer.android.com/about/dashboards/

Appium. (2024). Appium: Open-source test automation framework for mobile apps.https://appium.io/

Chen, Y., Sun, W., & Li, X. (2016). Automated testing framework for multi-platform mobile applications.

Google. (2016). *Google Play Store*. https://play.google.com/store

IDC. (2015). *Worldwide smartphone market will see the first single-digit growth year on record, according to IDC*.https://www.idc.com/getdoc.jsp?containerId=prUS40664915

Invertase. (2024). Patrol: End-to-end testing framework for Flutter apps

Joorabchi, M. E., Mesbah, A., & Kruchten, P. (2013). Real challenges in mobile app development. In Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

Engineering and Measurement (pp. 15–24).

Kaczmarczyk, R., Zając, K., & Zabierowski, R. (2022). Comparison of native and hybrid mobile applications: Efficiency and performance analysis. Journal of Mobile Computing, 17(4), 123–135.

Kanwar, G. (2025). A unified framework for balancing security, performance, and UX in real-time mobile applications: Lessons from industry at scale. Sarcouncil Journal of Engineering and Computer Sciences, 4(9), 180–195.https://doi.org/10.5281/zenodo.17083739.

Kong, P., Li, L., Gao, J., Liu, K., Bissyandé, T. F., & Klein, J. (2019). Automated testing of Android apps: A systematic literature review. *IEEE Transactions on Reliability, 68*, 45–66.

Kryan, Y., Arya, H., & Verma, H. (2015). Keyword-driven automated testing framework for web applications. In Proceedings of the 9th IEEE International Conference on Industrial and Information Systems (pp. 1–6).

Lecomte, A., & Martinez, D. (2017). Cross-platform mobile app development challenges. International Journal of Mobile Development, *8*(2), 87–99.

Linares-Vásquez, G., Bavota, G., Escobar-Velásquez, C., & Oliveto, R. (2017a). CRASHSCOPE: An automated mobile crash reporting and testing framework. *IEEE Transactions on Mobile Computing*. https://doi.org/10.1109/TMC.2017.2673498

Linares-Vásquez, M., Bavota, G., Escobar-Velásquez, C., & Oliveto, R. (2017). An empirical study on Android-related vulnerabilities. In *Proceedings of the 14th IEEE/ACM* International Conference on Mining Software Repositories (pp. 2–13).

Linares-Vásquez, M., Moran, K., & Poshyvanyk, D. (2017b). Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution (pp. 399–410).

Luo, C., Goncalves, J., Velloso, E., & Kostakos, V. (2020). Survey of context simulation for testing mobile context-aware applications. *ACM Computing Surveys, 53*(4), 1–39. https://doi.org/10.1145/3406797

Malhotra, D. (2016). Design and analysis of experiments. Wiley.

Muccini, H., Di Francesco, A., & Esposito, P. (2012). Software testing of mobile applications: Challenges and future research directions. In *Proceedings of the 7th International Workshop on Automation of Software Test* (pp. 29–35).

Rao, V., & Kumar, A. (2023). Mobile application automation testing using Appium: Challenges and limitations. International Journal of Software Engineering and Applications, 14(2), 45–53. https://doi.org/10.18178/ijseia.2023.14.2.105

Sharma, S., & J. A. (2021).An efficient keyword-driven test automation framework for web applications. International Journal of Engineering Science and Advanced Technology, 2, 600–604.

Singh, J., Sahu, S. K., & Singh, A. P. (2018). Implementing test automation framework using model-based testing approach. In Intelligent Computing and Information and Communication (pp. 695–704). Springer

Syer, M. D., Jiang, Z. M., Nagappan, M., Hassan, A. E., Nair, H., Begum, S., & Flora, P. (2013). Continuous testing of mobile applications: An empirical study. In *Proceedings of the 29th* IEEE International Conference on Software Maintenance (pp. 54–57).

Taniguchi, M. (2023). *A literature review of test coverage metrics* (Master's thesis, Osaka University). https://sdl.ist.osaka-

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.

u.ac.jp/pman/pman3.cgi?DOWNLOAD=646

Tramontana, P., Amalfitano, D., & Amatucci, N. (2019). Automated functional testing of mobile applications: A systematic mapping study. *Software Quality Journal, 27*, https://doi.org/10.1007/s11219-018-9434-0

Wang, J., & Wu, J. (2019). Research on mobile application automation testing technology based on Appium. In *Proceedings of the 2019* International Conference on Virtual Reality and Intelligent Systems (pp. 247–250). IEEE.

Wasserman, A. I. (2010). Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (pp. 397–400).

Weich broth, P. (2024). Usability testing of mobile applications: A methodological framework. *Applied Sciences, 14*(17), 1792. https://doi.org/10.3390/app14171792

Zein, S., Salleh, N., & Grundy, J. (2016). Systematic mapping study of mobile application testing techniques. *Journal of Systems and Software, 117*, 334–356. https://doi.org/10.1016/j.jss.2016.03.009

Wu, Z., Liu, S., Li, J., & Liao, Z. (2016). Keyword-driven testing framework for Android applications. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering* (pp. 1096–1102).

Corresponding author: Aisha Kabir Galadanci
kagaladanci@nda.edu.ng
Department of Electrical and Electronics Engineering, Faculty of Engineering, Nigerian Defence Academy, Kaduna.