A Glance of Mobile Testing Automation: Overview, Case Study, Challenges and the Future

Bao Cheng
Wenzhou-Kean University
1335784
baoc@kean.edu

Qiu Xinxin
Wenzhou-Kean University
1335785
qiuxin@kean.edu

Zhang Dinggen
Wenzhou-Kean University
1336574
1336574@wku.ed.cn

Chen Hao
Wenzhou-Kean University
1336564
1336564@wku.ed.cn

***Abstract*—Automated mobile application testing is a crucial aspect of software development lifecycle (SDLC), ensuring the quality and reliability of mobile applications. This paper provides an overview of the current state of automated mobile application testing, a study case between two tools that widely used in modern industry, as long as the challenges faced in this domain, including device fragmentation, varying operating systems, and the need for efficient test automation frameworks. Furthermore, we explore the future trends and advancements in automated mobile application testing, such as AI-driven testing solutions and cloud-based testing platforms. By addressing these challenges and embracing emerging technologies, we can enhance the efficiency and effectiveness of automated mobile application testing.**

***Index Terms*—Automated Testing, Mobile Application, Case Study, Automated Testing Tools**

I. Introduction

Over the past decades, the development of Information & Communication Technologies (ICT)—including 5G, fiber to the home (FTTH), cloud computing and others—has driven rapid growth in both mobile devices and the mobile applications market. According to International Data Corporation (IDC), more than 1.2 billion mobile devices were in circulation in 2014[1], and 18.22 billion by 2025[2]. The Grand View Research's mobile market report indicated that the mobile application market reached USD 252.89 billion in 2023 and is projected to grow at a compound annual growth rate (CAGR) of 14.3% from 2024 to 2030, reaching USD 626.39 billion[3]. As Fig. 1 shows, mobile device operating systems have consolidated from multiple options like Symbian OS[4] and Windows Phone[5] to two main systems: Google Android[6] and Apple iOS[7]. Yet this reduction in operating system choices has not simplified mobile application development.

With the rapid growth of mobile devices and applications, the demand for high-quality mobile applications has increased significantly[8]. This surge has created a pressing need for effective mobile software development across all lifecycle phases to ensure application timeliness, reliability, and performance. However, in the context of mobile application automation, there remain a number of unique challenges that require attention, such as: Device heterogeneity, Operating system fragmentation, Unstable and variable network conditions, Cross-platform compatibility issues[9]. Traditional testing strategies often fall short when addressing these issues effectively and comprehensively[10], [11]. Automated testing, as an integral part of the software development lifecycle, has emerged as a key solution to these challenges, offering faster and more efficient testing processes than manual approaches[12]. This paper examines the comprehensive landscape of automated mobile application testing, addressing both its significance and challenges. By listing all representative automated testing tools and comparing two main tools, which is Appium[13] and Airtest[14], we try to seek the path of how to achieve the efficient way of testing mobile application in real-world practice. Drawing from previous research, we explore future trends in automated mobile testing, particularly the integration of cutting-edge technologies such as AI and cloud computing, to outline the field's trajectory. At the same time, we also try to identify the efficiency and effectiveness of automated mobile application testing.

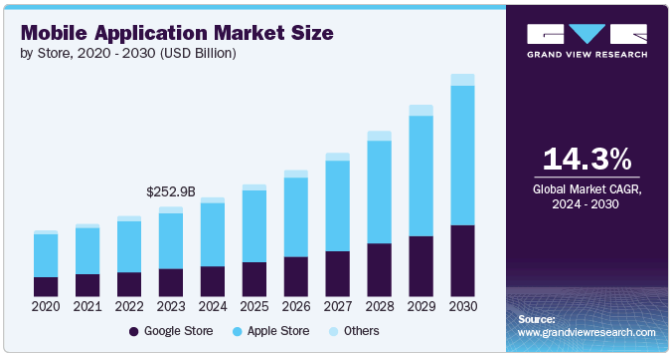

Fig. 1: The Mobile Application Market Size Insight (by store, 2020 - 2030), Grand View Research report, 2025

II. Background Knowledge

A. Software Testing Fundamentals

According the definition[15], software testing is a process aimed at evaluating the correctness, completeness, and quality of software by executing it under controlled conditions[16].

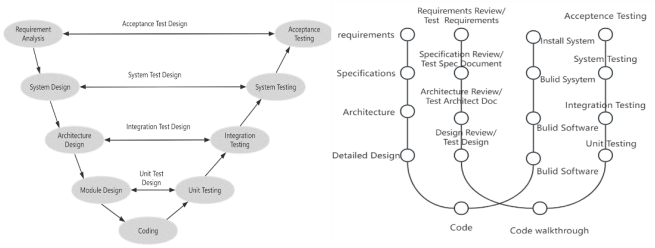

Fig. 2: The V model (left) and W model (right) of software development life cycle

Its primary goal is to identify defects and ensure that the software system meets its specified requirements[17].

B. The Core Principles of Software Testing

The ability of testing to show the presence of defects, the impossibility of exhaustive testing, the cost savings of early testing, the phenomenon of defect clustering, the pesticide paradox, and the dependence of testing on the specific environment[18]. The software testing life cycle consists of the phases of requirements analysis, test planning, test case[19] design, test environment setup, test execution, defect logging and retesting, and test closure[20].

In the software develop life cycle (SDLC), as Fig. 2 shows, the V model plans the development phases (user requirements analysis, system design, module design, coding implementation) in parallel with the corresponding testing phases (unit testing, integration testing, system testing, acceptance testing) [21]; while the W model is an enhanced version of the V model, emphasizing that testing activities should not be postponed until the completion of the development but should be carried out synchronously in each development phase, such as requirements validation during the requirements analysis phase and testability analysis in the requirements analysis phase, and creation of system test plans in the system design phase.

C. Mobile Testing Automation

To clarify what we discuss, the definition of mobile application testing must be specified. According to Wikipedia, mobile application testing is a process by which application software developed for handheld mobile devices is tested for its functionality, usability and consistency[22]. The types of mobile application testing divide into two kinds: manual testing and automated testing. Automated mobile application testing, also called mobile testing automation in some fields, involves using specialized tools and scripts to execute tests, validate an application's behavior, and ensure it meets the desired quality standards without human intervention. Because of the exclusion of the human factor, it accelerates the testing process and allows for achieve the continuous integration and continuous delivery (CI/CD) workflow in the development life cycle, as long as the many automated testing tools or platforms, shown as Fig. 3 [23].

D. Types of Mobile Application Testing

As Table I shows, considering the way of testing and the function-related aspect, the types of mobile application testing also can be divided into two major categories: functional testing and non-functional testing. Functional testing focuses on verifying the application's functionality, ensuring that it behaves as expected and meets the specified requirements. This includes unit testing, integration testing, system testing, and acceptance testing. Non-functional testing, on the other hand, evaluates the application's performance, security, usability, and compatibility across different devices and platforms. This includes performance testing, security testing, usability testing, and compatibility testing[23].

When in the context of mobile application testing, functional testing is often performed using automated testing tools and frameworks, while non-functional testing may require a combination of automated and manual testing approaches. The choice of testing methods depends on the specific requirements of the application and the goals of the testing process, also considered by the scale of the application and the development teams. Following are the details of each mobile application testing type:

a) *Functional Testing:*

Functional testing ensures that an app's features and functionalities work as intended. It involves testing user interactions, APIs, databases, security, and client/server applications. In addition, automated functional testing verifies the end-to-end workflow of an app to confirm it meets the specified requirements[24].

b) *Performance Testing:*

Performance testing evaluates how well a mobile application performs under various conditions, including different network speeds, high user loads, and varied device configurations. It identifies performance bottlenecks and ensures the app can handle peak usage times without degrading user experience.

Performance testing is critical for mobile applications due to the vast array of devices and network conditions they must support, emphasizes Michael Facemire, Vice President and Principal Analyst at Forrester. His extensive research underlines the complexities of mobile performance testing[25], [26], [27], [28].

c) *Usability Testing:*

Usability testing identifies vulnerabilities within the mobile application to protect it from potential threats. Automated

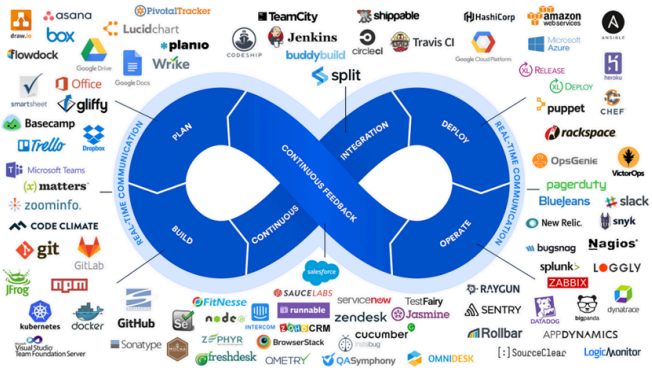

Fig. 3: Continuous Integration and Continuous Delivery (CI/CD) workflow & related tools in Mobile Application Testing

TABLE I: TYPES AND WAYS OF MOBILE APPLICATION TESTING

Categories	Sub-categories	Way of testing
Functional Testing	Unit Testing	Automated tools & frameworks
	Intergration Testing	
	System Testing	
	Acceptance Testing	
Non-Functional Testing	Performance Testing	Combined automated & manual testing
	Security Testing	
	Usability Testing	
	Compatibility Testing	
	Regression Testing	

security tests check for weaknesses such as unauthorized access, data leaks, and other security flaws[29].

d) *Security Testing:*

Security testing identifies vulnerabilities within the mobile application to protect it from potential threats. Automated security tests check for weaknesses such as unauthorized access, data leaks, and other security flaws[29].

e) *Compatibility Testing:*

Compatibility testing ensures the app works across different devices, operating systems, and screen sizes. Automated compatibility tests verify that the application delivers a consistent user experience regardless of the device or platform, become more and more important in modern mobile landscape[30].

f) *Regression Testing:*

Regression testing ensures that new code changes do not adversely affect the existing functionality of the application. Automated regression tests are crucial for maintaining the integrity of the app after updates or enhancements[31].

III. AUTOMATED TESTING FRAMEWORKS & TOOLS

With the development of mobile testing automation going on, there are several frameworks and tools used widely in this field. The "framework" here refers to a combination of tools or software that provide the foundation for establishing automation testing by the users' own desire. Using the framework can accelerate the process of performing automation testing and prevent "re-inventing the wheel", which is considered a modern way of mobile testing automation in real-world practice. The following shows some of the most popular frameworks or tools, along with their features, pros and cons:

A. Selenium

Selenium is a widely adopted open-source framework primarily designed for web application testing but extended for mobile web testing through integration with tools like Appium or Selendroid[32]. Selenium is not just one tool or API, it comprises many tools like Selenium WebDriver, Selenium IDE, and Selenium Grid. At its minimum, WebDriver talks to a browser through a driver, such as ChromeDriver for Google's Chrome/Chromium, GeckoDriver for Mozilla's Firefox, etc. Communication is two-way: WebDriver passes commands to the browser through the driver, and receives information back via the same route. Remote communication can also take place using Selenium Server or Selenium Grid, both of which in turn talk to the driver on the host system[33].

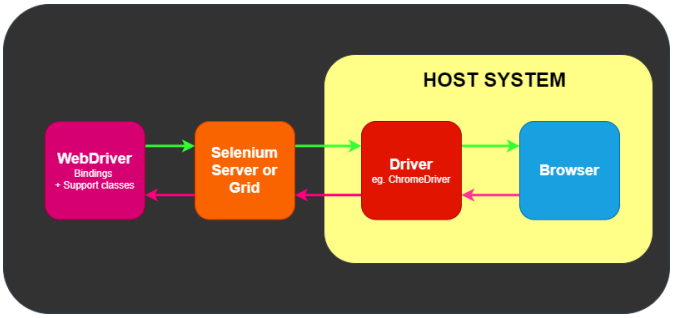

Fig. 4: Remote Communication Flow of Selenium

a) *Features:*

A key strength of Selenium for mobile web testing is its support for cross-browser testing, allowing testers to verify application behavior across various mobile browsers (such as Chrome, Firefox, and Safari) on both Android and iOS devices. It leverages the WebDriver protocol—W3C-compliant in Selenium 4—to interact with mobile browsers, enabling simulation of user actions like clicks, swipes, and text input. Additionally, Selenium offers language flexibility, supporting multiple programming languages (Java, Python, C#, Ruby, JavaScript), so testers can write scripts in their preferred language.

To enhance mobile testing, Selenium integrates with Appium (for Android and iOS) or Selendroid (Android-only), utilizing its WebDriver API for mobile browser automation. It also provides the flexibility to run tests on real devices, emulators, or simulators, accommodating different testing environments. For scalability, Selenium Grid enables parallel test execution across multiple devices and browsers, reducing execution time. Test scripts can be made reusable and maintainable using frameworks like TestNG or JUnit, often following the Page Object Model (POM) for better organization. Furthermore, Selenium supports cloud-based testing through platforms like BrowserStack and Sauce Labs, allowing testers to evaluate mobile web applications across a wide range of devices and browser versions without requiring a local setup. These features make Selenium a robust choice for mobile web automation.

b) *Pros & Cons:*

To breakdown the pros and cons of using Selenium, the following are the pros of using Selenium:

- Open-Source & Free: No licensing costs, with strong community support.
- Broad Browser Support: Works on Chrome, Firefox, Safari, and more.
- Language & Framework Flexibility: Supports Java, Python, C#, etc., and integrates with TestNG/JUnit.
- Scalability with Selenium Grid: Enables parallel testing across devices for faster execution.
- Strong Community & Ecosystem: Extensive documentation and third-party tool integrations (e.g., Allure, Extent Reports).

- Easy Transition for Web Testers: Familiar Selenium WebDriver API works with Appium/Selendroid for mobile.
- Cloud Integration: Works with BrowserStack, Sauce Labs for testing on multiple devices without local setup.

And the cons of using Selenium are:

- No Native Mobile App Support: Requires Appium/ Selendroid for mobile apps, adding complexity.
- Setup Complexity: Configuring Appium, Android SDK, or Node.js can be difficult for beginners.
- Dynamic Content Challenges: Struggles with AJAX-heavy pages, needing manual waits.
- Device Fragmentation Issues: Testing across different devices/OS versions can lead to inconsistencies.
- No Built-in Reporting: Requires third-party tools (e.g., Allure) for detailed reports.
- Limited Gesture Support: Mobile gestures (swipes, pinches) depend on Appium's TouchAction API.
- Maintenance Overhead: Scripts may need frequent updates due to browser/UI changes.

B. *Appium*

Appium is a popular open-source framework used for automated mobile app testing. It allows developers to automate the testing of native or hybrid iOS and Android applications. Appium doesn't work alone. It runs the test cases using the WebDriver interface[13].

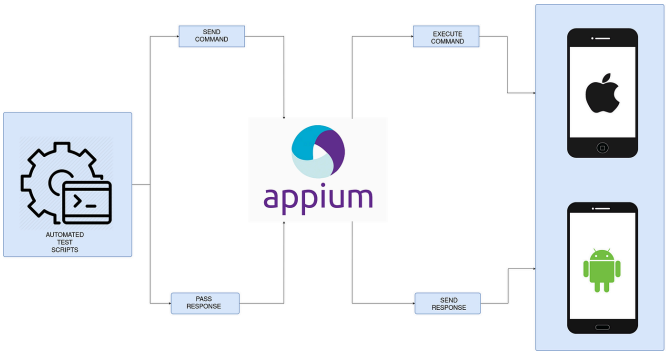

Fig. 5: Appium Software Architecture

a) *Features:*

Similar to Selenium, Appium is a versatile testing framework that supports multiple programming languages including Java, JavaScript, PHP, Ruby, Python, and C#. As a flexible cross-platform solution, it enables testers to create scripts that work across Windows, iOS, and Android platforms using a unified API. One of its key advantages is the ability to reuse source code between Android and iOS, significantly reducing development time and effort. Notably, Appium doesn't require modifications to the app's source code for automation purposes. The framework supports testing of native, hybrid, and mobile web applications, while also integrating seamlessly with popular cross-platform development frameworks like React Native, Xamarin, and Flutter. To enhance the testing experience, Appium provides useful built-in tools such as Appium Desktop and Appium Inspector, which help in inspecting app elements and generating test scripts.

b) *Pros & Cons:*

The following are the pros of using Appium:

- Supports multiple programming languages (Java, JavaScript, Python, C#, etc.).
- Enables cross-platform testing for iOS and Android with a single API.
- Doesn't require modification of the app's source code.
- Supports native, hybrid, and mobile web applications.
- Works with frameworks like React Native, Xamarin, and Flutter.
- Offers built-in tools (Appium Desktop, Appium Inspector).

Along with the pros, there are also some cons of using Appium:

- Slower test execution compared to native frameworks.
- Requires a learning curve for beginners.
- Limited support for advanced gestures and animations.

C. *Airtest*

Airtest is a cross-platform UI automation testing framework developed by NetEase, designed especially for games and apps with graphical interfaces. It utilizes image recognition technology to locate UI elements and supports scripting in Python, making it highly suitable for tasks involving visual feedback[14].

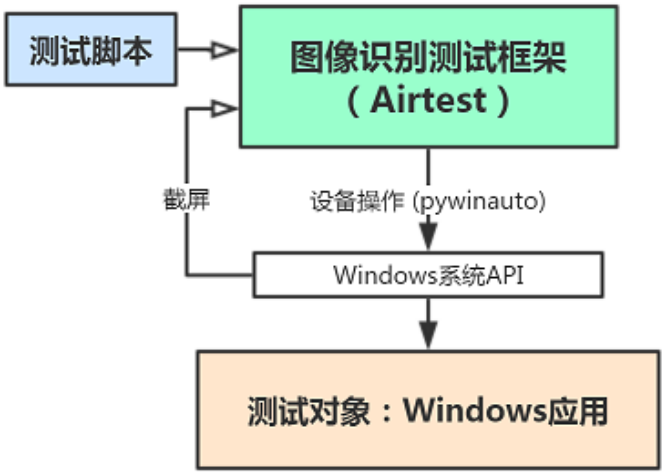

Fig. 6: Airtest Software Architecture

a) *Features:*

Airtest is an open-source, cross-platform automation testing framework specializing in game testing and mobile app UI automation, leveraging image recognition and simulated touch operations to interact with apps without relying on traditional element locators. It supports Android, iOS, Windows, and web applications, enabling concurrent multi-device testing, and uses Python for scripting with a built-in AirtestIDE for drag-and-drop test recording and playback. The framework integrates Poco for element-based testing when UI hierarchies are accessible, supports OCR (text recognition), and simulates taps, swipes, and keypresses. While it excels in dynamic environments like games, its reliance on image recognition can lead to slower execution and higher maintenance due to

sensitivity to screen changes, making it ideal for scenarios where DOM-based tools like Appium fall short.

b) *Pros & Cons:*

The following are the pros of using Airtest:

- No deep coding knowledge required; scripts can be generated via GUI actions.
- Scripts written for Android can often be adapted for iOS with minimal changes.
- Like Appium, it doesn't require access to source code.
- Superior for Unity3D, Cocos2dx, and other game engines.

With the pros, there are also some cons of using Airtest:

- Accuracy affected by screen resolution, lighting, or UI changes (requires frequent template updates).
- Image processing is more resource-intensive than DOM-based methods like Appium.
- Complex animations or gestures may not be fully supported.
- Requires ADB, JDK, and device-specific configurations.

D. NightwatchJS

NightwatchJS is a Node.js based framework that is developed and maintained by BrowserStack. Nightwatch uses Appium under the hood to achieve mobile application automation on virtual simulators & real devices. Nightwatch also takes care of the entire installation with just a single command[34].

a) *Features:*

NightwatchJS offers several key features that make it a powerful tool for mobile testing automation. It provides a streamlined setup process with one-command installation of all necessary SDKs and libraries. The framework comes with native Nightwatch commands and integrates seamlessly with Appium Inspector. One of its standout features is support for parallel test execution, enabling tests to run simultaneously across multiple browsers and devices to significantly reduce testing time. Additionally, Nightwatch.js is versatile in its testing capabilities, supporting both real devices and emulators/simulators for web-based mobile application testing.

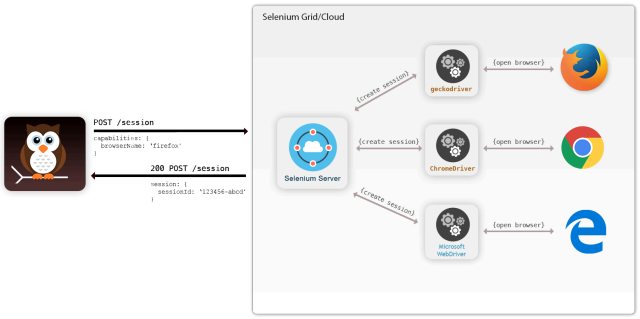

Fig. 7: NightwatchJS Operation Cloud Architecture

b) *Pros & Cons:*

The following are the pros of using NightwatchJS:

- Easy setup with one-command installation.
- Supports parallel test execution for faster test runs.
- Works with real devices and emulators/simulators.
- Integrates seamlessly with Appium and Appium Inspector.
- Uses JavaScript, making it developer-friendly.

With the pros, there are also some cons of using NightwatchJS:

- Limited community support compared to Appium and Espresso.
- Requires Node.js, which may not suit all development teams.
- Best suited for web-based mobile apps rather than native apps.

E. Calabash

Calabash is a mobile test automation framework that works with multiple languages. It supports Ruby, Java, Flex, and NET. Testers can use APIs to enable native applications that run on touchscreen devices. This framework has libraries that allow test scripts to interact programmatically with native and hybrid apps[35].

a) *Features:*

Calabash enables writing and executing automated tests for mobile applications across both iOS and Android platforms. It allows test scripts to be reused across different platforms and devices, which significantly reduces maintenance effort. The framework supports both native and hybrid mobile applications, and offers compatibility with multiple programming languages including Ruby and .NET languages like C#. Additionally, its parallel testing capability helps decrease overall test execution time substantially.

b) *Pros & Cons:*

The following are the pros of using Calabash:

- Supports multiple languages (Ruby, Java, .NET).
- Works for both iOS and Android platforms.
- Allows interaction with native and hybrid apps.
- Enables parallel test execution to speed up testing.

Along with the pros, there are also some cons of using Calabash:

- Requires additional setup for integration with CI/CD pipelines.
- Official support and development have slowed down.
- Not as widely adopted as Appium or Espresso.

F. XCUItest

XCUITest is Apple's native automation framework for testing iOS applications. Among mobile testing tools, this one is best known for testing iOS apps. Launched by Apple in 2015, the XCUITest framework is meant to create and run UI tests on iOS apps using Swift / Objective-C[36].

a) *Features:*

XCUITest, built and maintained by Apple, ensures seamless compatibility with the latest iOS versions and features. Known for its fast execution, intuitive operation, and low flakiness, it allows developers to write test scripts in Swift or Objective-C while leveraging their existing language skills. The framework supports comprehensive testing on both real iOS devices and simulators, while providing APIs to interact with various device features like camera, GPS, and accelerometer. XCUITest also includes accessibility testing features to ensure app usability for individuals with disabilities. With its robust

automation features and tight integration with XCode and XCTest, it stands as a reliable choice for iOS app automation testing.

```
import XCTest

class SampleXCUITest: XCTestCase {

override func setUp() {
super.setUp()
continueAfterFailure = false
XCUIApplication().launch()
}


func checkText() {
let appObj = XCUIApplication()
appObj.otherElements.containing(.image, identifier:"XCUITest Tutorial").element.tap()
appObj.buttons["Learn More"].tap()
XCTAssert(appObj.staticTexts["Automation Test iOS"].exists)
}
```

Fig. 8: A Sample code of XCUITest Implementation

b) *Pros & Cons:*

The following are the pros of using XCUITest:

- Native framework for iOS, ensuring high performance.
- Faster execution compared to cross-platform tools.
- Deep integration with Xcode and Swift.
- Supports accessibility testing.

Along with the pros, there are also some cons of using XCUITest:

- Limited to iOS and cannot be used for Android testing.
- Requires knowledge of Swift/Objective-C.
- No built-in cross-platform testing support.

IV. Case Study: Appium vs Airtest

In this study, we compare the performance and usability of two automated testing frameworks, Airtest and Appium, in a controlled testing environment[37], [38].

To ensure a fair comparison, both frameworks were configured to interact with an Android virtual device running on a Windows platform. The connection between the frameworks and the virtual device was established via Android Debug Bridge (ADB), allowing both Airtest and Appium to issue commands and capture responses in a consistent manner. The testing task was designed as follows:

- Launch the virtual Android environment.
- Use each framework to open the Settings application and navigate to the Battery options page.
- Once the Battery option is detected, trigger a screenshot of the page and save it to local storage.
- Measure the total execution time from the initiation of the test script to the successful capture of the screenshot.

A. Comparison of Base Features

As the summary of previous section, the basic differences between Appium and Airtest are shown in Table II. In conclusion, both tools support Android, iOS, and H5 platforms and have similar installation complexity. Appium holds a dominant market position, with extensive community support and abundant related resources, whereas Airtest is gaining popularity gradually but offers comparatively fewer resources. In terms of programming language flexibility, Appium supports almost all major languages, while Airtest is primarily Python-based. A notable distinction lies in development tools: Airtest provides a dedicated IDE to simplify test creation, while Appium lacks an official IDE, requiring integration with external editors. Regarding usability, Airtest demonstrates lower learning difficulty and provides essential features such as test reports, CI/CD integration, and web support—comparable to Appium. However, Appium offers additional advantages like available APIs and broader language adaptability, making it more suitable for complex, scalable automation tasks.

TABLE II: Comparison of base features between Appium and Airtest

Features	Appium	Airtest
Installation	Medium	Medium
Supported Platforms	Android, iOS, H5	Android, iOS, H5
Market Share	Top 1	Gaining Increasing
API Availability	Available	Limited
Related Resources	Many	Few
Language Support	Almost All	Primarily Python
Development IDE	None	Yes
Learning Difficulty	Medium	Low
Test Report	Available	Available
CI/CD Integration	Available	Available
Web Support	Available	Available

B. Runtime Testing

For Appium, we need to write python scripts and we need to start the Appium server first, connect to the VM with ADB and then use python code for testing. The code needs to index the specified target tag, so it requires the tester to be familiar with the operation principle of the app under test, and requires a high threshold for use. The result of the runtime testing is shown in Fig. 9.

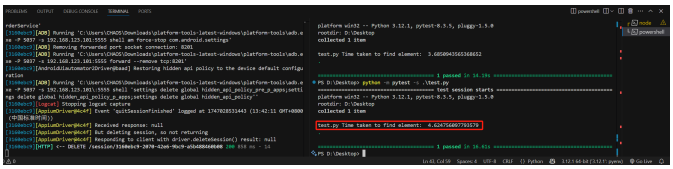

Fig. 9: Appium Runtime Testing Result

And Airtest also need to write code, with its code is an extension of the python code, and then after installing a specific IDE, the code can be used in the picture (screenshot in the system) to identify the app components you want to access to be tested, which greatly reduces the threshold of use, and facilitate the use of more developers. The result of the runtime testing is shown in Fig. 10.

However, in terms of testing efficiency, we can find that because of the different architectural design of the two test frames, the time consumed for the same task is not the same. Appium uses element indexing to find the corresponding element in the front-end DOM tree, which is very efficient,

whereas Airtest, because of its graphical programming, needs to use computer vision techniques[39] to find the corresponding tags. In summary, we believe that developers who have some familiarity with the system and need high performance, we recommend Appium, on the contrary, if you don't have a lot of basic knowledge, we recommend using the Airtest framework, which will be easier to get started in your testing process.

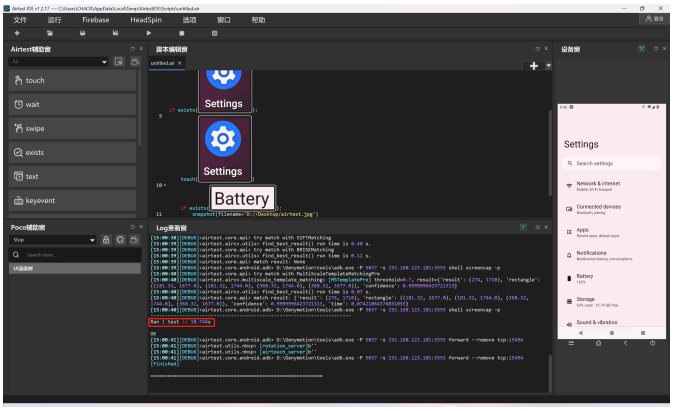

Fig. 10: Airtest Runtime Testing Result

C. Cross-platform Compatibility Assessment

As the Table III summarizes, both frameworks provide full native support for Android, ensuring robust automation capabilities. For iOS, Appium demonstrates stronger support through integration with XC Test, while Airtest offers only limited functionality. In terms of web (H5) testing, Appium supports WebDriver and WebView automation, offering more reliable handling of web contexts compared to Airtest's image-based approach. Neither framework supports native automation for Unix/Linux GUI applications, which fall outside the primary scope of both tools.

D. Handing of Unique Mobile Testing Challenges

In addressing complex mobile testing challenges, Appium shows several limitations that require manual intervention or third-party tools. Tasks such as network switching and battery/CPU monitoring rely on external ADB triggers or profiler tools, while context-aware testing—such as GPS simulation or handling input explosion—requires custom scripts and lacks native support. Although Appium can manage app crashes and recovery scenarios, it demands additional logic to ensure reliability. These limitations highlight the need for enhanced support mechanisms or integration with complementary tools[40] to streamline testing workflows and improve efficiency in handling dynamic mobile environments.

TABLE III: TESTING ON DIFFERENT PLATFORMS

Platform	Appium	Airtest
Android	Full Native Support	Full Native Support
iOS	XCUITest integration	Limited Support
Web (H5)	WebDriver/WebView Automation	Image-based Locators
Unix/Linux	No directly Support	No Support

E. Result & Discussion

The results indicate that while Appium's execution speed is slightly lower than Airtest, its strengths lie in cross-platform compatibility, community support, and integration flexibility. These make it especially suitable for enterprise-level testing and CI/CD workflows.

Appium's active open-source community ensures rapid adaptation to new Android/iOS versions and devices, enhancing test stability across fragmented environments. Its client-server architecture, though adding some latency, provides high reusability[41] and broad compatibility, making it ideal for teams working across multiple platforms.

However, this generality comes at the cost of limited support for device-specific features (e.g., sensor simulation, advanced gestures), where image-based frameworks like Airtest may perform better.

Based on the experimental findings, Appium—despite its performance overhead—proves to be a reliable and scalable solution for integrating automated testing into the mobile development lifecycle. To maximize its value, development and QA teams should:

a) *Plan automation early:* Introduce Appium-based test architecture at the requirement/design phase, aligning it with UI components and functional modules to support early regression coverage.

b) *Adopt a hybrid tool strategy:* Combine Appium with lightweight tools like Airtest for tasks requiring image recognition, sensor simulation, or high-frequency test execution, ensuring flexibility and coverage.

c) *Integrate with CI/CD:* Use Appium in continuous integration pipelines (e.g., Jenkins, GitHub Actions) to enable automated build validation and reduce manual testing overhead.

d) *Allocate test resources strategically:* Focus automation on high-risk areas such as login, payment, and navigation flows, while using manual or exploratory testing for edge cases.

e) *Promote collaboration:* Encourage shared ownership of test scripts between developers and testers using unified coding standards and reusable Page Object models.

These practices can help reduce testing cost and risk, shorten release cycles, and ensure higher product quality across diverse mobile platforms.

V. CHALLENGE OF MOBILE TESTING AUTOMATION

As the techniques are developing rapidly in the field of mobile testing automation, the situation of testing remains challenging. Several significant issues, due to the fragmented nature of devices, operating systems, and testing environments, needed to be solved desperately. One major issue is the diversity of devices and platforms, such as iOS, Android, and others, each with varying hardware specifications, screen sizes, and OS versions. This fragmentation necessitates testing across numerous combinations to ensure compatibility, which is both time-consuming and resource-intensive. Additionally, automation scripts must be maintained and updated frequently to accommodate app updates and new features, requiring

robust frameworks that can handle regression testing without introducing errors[8].

Another critical challenge is the reliance on both emulators and real devices. While emulators are cost-effective, they often fail to replicate real-world conditions, such as processor limitations, network variability, and user interactions, leading to gaps in test coverage. Furthermore, distributed testing across different geographical locations introduces complexities, including time zone differences, network latency, and local carrier-specific issues, which can affect app performance and user experience. The tools mentioned above help mitigate some of these issues, but challenges like script maintenance, cross-platform compatibility, and accurate crash reporting persist, requiring continuous innovation in testing methodologies[23].

Despite the challenges listed above, some key problems will also be faced when implementing automated mobile testing[22], they are:

a) *Application downloadable:* The application must be available for the specific platform, typically through an app store.

b) *Diversity in mobile platforms/OSes:* The market includes several mobile operating systems, primarily Android, iOS, and Windows Phone. Each operating system has distinct limitations.

c) *Device availability:* With an ever-growing variety of devices and operating system versions, accessing the right test devices is a persistent mobile application testing challenge. This becomes more complex when testing teams are geographically distributed.

d) *Mobile network operators:* Over 400 mobile network operators worldwide use different standards, including CDMA, GSM, FOMA, and TD-SCDMA.

e) *Scripting:* Device diversity complicates test script execution. Since devices vary in keystrokes, input methods, menu structures, and display properties, a single script rarely works across all devices.

f) *Test method:* Mobile applications can be tested on either real devices or emulators. While emulators are convenient, they may miss issues that only real devices can detect. However, testing on real devices is costly and time-consuming due to the wide variety of devices available.

g) *Compatibility:* Applications must maintain compatibility across various device properties, including different screen sizes and resolutions.

h) *Phone functionality:* Applications must handle incoming calls appropriately during execution.

i) *Device diversity:* Mobile devices feature various input methods (QWERTY, touch, standard) and different hardware capabilities.

j) *Load testing limitations:* Engineers often lack sufficient hardware resources to create comprehensive user scenarios for performance testing of mobile applications.

VI. about the future

Starting from 2023, the advancements in AI and machine learning are enhancing almost every aspect of computer science. The shift towards AI-powered automated testing is not just a trend, but a fact. With the help of AI, the pioneers in mobile testing automation are trying a new efficient way to change the very cornerstone of automated testing, which is element identification. By introducing hybrid strategies that include context-aware locators, heuristic-based identification, and early implementations of adaptive and self-healing locators, to reduce the usage of traditional element identification methods, such as XPath, CSS selectors, and static attributes, to solve the key challenges above[42].

Another breakthrough of the mobile testing automation will be the change of the testing device or environment. The definition of Test-as-a-Service (TaaS) was introduced back in 2015[43], but because of the limitations of cloud resources and other technical barriers, there have been no significant breakthroughs recently. Nowadays, with the concept of cloud-native becoming more and more acceptable for developers, many applications, as well as mobile applications, are "born" with cloud-native. With cloud-based platforms like AWS Device Farm and Firebase Test Lab, tester can quickly and efficiently test their mobile application across multiple devices and operating systems without any local preparation. Using cloud-based platforms also makes it easier for testers to keep track of their results and analyze their performance.

Finally, in the foreseeable future, mobile app testing will continue to become more accessible and affordable. The cost of testing mobile applications has been steadily decreasing over the years, and this trend is likely to continue in the coming years. This will make it easier for developers and testers to access the tools and resources they need to ensure that their apps are of the highest quality.

VII. conclusion

In summary, automated mobile application testing has become an indispensable part of the software development lifecycle, with various tools and frameworks available to address different testing needs. While established tools like Appium, Calabash, and XCUITest provide robust solutions for current testing requirements, significant challenges persist in the form of device fragmentation, platform diversity, and resource constraints. However, the future of mobile testing automation appears promising with the emergence of AI-powered testing solutions, cloud-based testing platforms, and more accessible testing tools.

The case study finds that Appium has a broad application prospect in mobile application automation testing due to its excellent cross-platform features, rich API support and good community support. It can effectively improve testing efficiency, shorten the testing cycle, and help development teams maintain an advantage in the fierce competition in the market. However, Appium also has some limitations, for example, it may be slightly delayed when dealing with new devices and operating system version adaptation, and the stability of testing is greatly affected by external environmental factors.

The integration of artificial intelligence in element identification and the growing adoption of cloud-native testing approaches are set to revolutionize how mobile applications are tested. The shift towards AI-powered automated testing is

particularly significant, as it addresses one of the fundamental challenges in mobile testing - element identification. With advanced techniques like context-aware locators and self-healing mechanisms, testing frameworks are becoming more resilient and adaptable.

Furthermore, the evolution of cloud-based testing platforms has opened new possibilities for scalable and efficient testing environments. The concept of Test-as-a-Service (TaaS) is gaining momentum, allowing organizations to leverage cloud resources for comprehensive testing across multiple devices and operating systems without significant infrastructure investments. This democratization of testing resources, combined with decreasing costs, is making quality assurance more accessible to developers and organizations of all sizes.

As the mobile application landscape continues to evolve, the testing methodologies and tools will need to adapt accordingly. The challenges of today are driving innovation in testing approaches, and the solutions being developed are not just addressing current limitations but are also paving the way for more sophisticated, efficient, and accessible testing practices in the future. This ongoing evolution in mobile application testing will ultimately contribute to higher-quality applications and better user experiences across the mobile ecosystem.

References

[1] D. Bernardo Silva, A. T. Endo, M. M. Eler, and V. H. S. Durelli, "An analysis of automated tests for mobile Android applications," in *2016 XLII Latin American Computing Conference (CLEI)*, Valparaíso, Chile: IEEE, Oct. 2016, pp. 1–9. doi: 10.1109/CLEI.2016.7833334.

[2] "Number of Mobile Devices Worldwide 2020-2025." Accessed: May 15, 2025. [Online]. Available: https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/

[3] Grand View Research, "Mobile Application Market Size, Share & Growth Report 2030." [Online]. Available: https://www.grandviewresearch.com/industry-analysis/mobile-application-market/segmentation

[4] "Symbian OS," *Wikipedia*, May 2025, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Symbian&oldid=1288336212

[5] "Windows Phone," *Wikipedia*, Apr. 2025, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Windows_Phone&oldid=1286569954

[6] "Android (Operating System)," *Wikipedia*, May 2025, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Android_(operating_system)&oldid=1288461632

[7] "iOS," *Wikipedia*, Apr. 2025, [Online]. Available: https://en.wikipedia.org/w/index.php?title=iOS&oldid=1286015929

[8] M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real Challenges in Mobile App Development," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Baltimore, Maryland: IEEE, Oct. 2013, pp. 15–24. doi: 10.1109/ESEM.2013.9.

[9] A. N. Mamedova and S. A. Bortsov, "Automated Solution of Qualitative Image Analysis Problem Based on Laplace Variation for Mobile Application Development," *Automatic Documentation and Mathematical Linguistics*, vol. 58, no. 6, pp. 401–414, 2025.

[10] D. T. Thanh, D. T. Huy, H. L. Su, and e. al, "Mobile Robot: Automatic Speech Recognition Application for Automation and STEM Education," *Soft Computing*, vol. 27, no. 15, pp. 10789–10805, 2023.

[11] S. J. S, R. S. K, and S. N, "Assessment of Influencing Factors in Selecting Effective Mobile Applications—a Drive toward Automation in Construction Projects," *Modeling Earth Systems and Environment*, vol. 9, no. 3, pp. 3747–3754, 2023.

[12] M. Linares-Vasquez, K. Moran, and D. Poshyvanyk, "Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Shanghai: IEEE, Sep. 2017, pp. 399–410. doi: 10.1109/ICSME.2017.27.

[13] "Welcome - Appium Documentation." Accessed: May 15, 2025. [Online]. Available: http://appium.io/docs/en/latest/

[14] "Welcome to Use - Airtest Project Docs." Accessed: May 29, 2025. [Online]. Available: https://airtest.doc.io.netease.com/en/

[15] D. R. Kuhn and e. al, *Introduction to Software Testing*. Cambridge University Press, 2017.

[16] B. Ardic, C. Brandt, A. Khatami, and e. al, "The Qualitative Factor in Software Testing: A Systematic Mapping Study of Qualitative Methods," *The Journal of Systems & Software*, vol. 227, p. 112447, 2025.

[17] A. Polanski, A. Roman, and J. Zelek, "Optimal Solutions for Variants of Graph Coverage-Related Problems in Software Test Design," *Expert Systems With Applications*, vol. 277, p. 127216, 2025.

[18] A. Wairagade and M. K. Cuthrell, "A Systematic Review of AI-Powered Software Testing in Healthcare: Methodologies, Challenges, and Future Directions," *Journal of Engineering Research and Reports*, vol. 27, no. 4, pp. 264–277, 2025.

[19] S. Gerasimou and e. al, "Search-Based Test Generation for Mobile Apps: A Survey," *Information and Software Technology*, vol. 104, pp. 1–20, 2018.

[20] H. Wu, "Analysis of the Effectiveness of Artificial Intelligence Technology in Defect Prediction in Software Testing," *Innovative Applications of AI*, vol. 2, no. 1, pp. 13–16, 2025.

[21] C. Dongmo, "A Review of Non-Functional Requirements Analysis Throughout the SDLC," *Computers*, vol. 13, no. 12, p. 308, 2024.

[22] "Mobile Application Testing," *Wikipedia*, Jan. 2025, Accessed: May 11, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Mobile_application_testing&oldid=1272006010

[23] "Mobile Test Automation: Everything You Need to Know." Accessed: May 11, 2025. [Online]. Available: https://www.tricentis.com/learn/mobile-test-automation-a-practical-introduction/

[24] "Functional Testing." Accessed: May 11, 2025. [Online]. Available: https://www.tricentis.com/learn/functional-testing/

[25] M. Facemire, "Mobile's Next Era: Not Apps, Not Websites — Experiences!." Accessed: May 11, 2025. [Online]. Available: https://www.forrester.com/blogs/mobiles-next-era-not-apps-not-websites-experiences/

[26] M. Facemire, "Development In The Enterprise: The Mobile Path Is Clear And Getting Easier!." Accessed: May 11, 2025. [Online]. Available: https://www.forrester.com/blogs/13-04-22-development_in_the_enterprise_the_mobile_path_is_clear_and_getting_easier/

[27] M. Facemire, "The Breathtaking Future of Software Development – It's Already Here!." Accessed: May 11, 2025. [Online]. Available: https://www.forrester.com/blogs/the-breathtaking-future-of-software-development-its-already-here/

[28] "Michael Facemire." Accessed: May 11, 2025. [Online]. Available: https://www.forrester.com/blogs/author/michael_facemire/

[29] "Software Testing: What It Is and How to Do It Right - Tricentis." Accessed: May 11, 2025. [Online]. Available: https://www.tricentis.com/learn/software-testing

[30] "Learn with Tricentis: What Is Software Testing?." Accessed: May 11, 2025. [Online]. Available: https://www.tricentis.com/learn/software-testing/

[31] "Regression Testing." Accessed: May 11, 2025. [Online]. Available: https://www.tricentis.com/learn/regression-testing/

[32] "Selenium Overview." Accessed: May 18, 2025. [Online]. Available: https://www.selenium.dev/documentation/overview/

[33] "documentation." Accessed: May 18, 2025. [Online]. Available: https://www.selenium.dev/documentation/overview/components/

[34] "Nightwatch V3 | Node.Js Powered End-to-End Testing Framework." Accessed: May 15, 2025. [Online]. Available: https://nightwatchjs.org/

[35] "Calabash." Accessed: May 15, 2025. [Online]. Available: https://github.com/calabash

[36] "XCUITest Tutorial – A Detailed Guide On XCUITest Framework." Accessed: May 15, 2025. [Online]. Available: https://www.lambdatest.com/xcuitest

[37] A. A. Alotaibi and R. J. Qureshi, "Novel Framework for Automation Testing of Mobile Applications Using Appium," *International Journal of Modern Education and Computer Science*, vol. 9, no. 2, p. 34, 2017.

[38] S. Yu and e. al, "Layout and Image Recognition Driving Cross-Platform Automated Mobile Testing," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, 2021.

[39] M. Linares-Vásquez and e. al, "Enabling Visual GUI Testing for Mobile Apps with GATOR," in *Proceedings of ICSE*, 2017, pp. 540–550.

[40] L. Li and e. al, "DroidBot: A Lightweight UI-Guided Test Input Generator for Android," in *Proceedings of Mobiquitous*, 2015, pp. 2–11.

[41] A. Rauf and A. M. Memon, "Test Reuse in Automated Mobile App Testing," *Empirical Software Engineering*, vol. 20, no. 1, pp. 1–28, 2015.

[42] Vinaysimha Varma Yadavali, "Advancing Element Identification for Web and Mobile Automation," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 45–56, Aug. 2024, doi: 10.48175/IJARSCT-6232.

[43] C. Tao, J. Gao, and B. Li, "Cloud-Based Infrastructure for Mobile Testing as a Service," in *2015 Third International Conference on Advanced Cloud and Big Data*, Yangzhou, Jiangsu, China: IEEE, Oct. 2015, pp. 133–140. doi: 10.1109/CBD.2015.30.