An Automated and Web-Based Platform for Remote Certification Testing of Android Mobile Devices

by

Sundos MOJAHED

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS
M.A.Sc.

MONTREAL, "14/06/2025"

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

M. Lokman Sboui, Thesis supervisor
Professor, Systems Engineering Department, École de Technologie Supérieure

M. Mohamed Cheriet, Chair, Board of Examiners
Professor, Systems Engineering Department, École de Technologie Supérieure

Mme. Imen Benzarti, Member of the Jury
Professor, Software and IT Engineering Department, École de Technologie Supérieure

M. Rejean Drouin, External Examiner
Senior Engineer, Certification of Wireless Devices Department, Vidéotron

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON "21/05/2025"

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

FOREWORD

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Engineering at the École de Technologie Supérieure (ÉTS), University of Quebec. The research presented herein was conducted under the supervision of Professor Lokman Sboui.

My master's research was part of an automation project at Vidéotron from winter 2023 to winter 2025. I was an intern with the certification of wireless devices team, which is responsible for certifying mobile devices. During my internship, I worked to develop an automation platform, troubleshoot issues, assist fellow engineers and interns in using the tool, and continuously enhance the platform by adding new features based on their feedback and requirements.

The results presented in this thesis are based on real data collected at the Vidéotron certification lab in collaboration with their engineers and interns.

This thesis is structured as a compilation of two papers:

- S. Mojahed, R. Drouin, and L. Sboui, "ODACE: An Appium-Based Testing Automation Platform for Android Mobile Devices Certification," In 2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 301–308, 2024, **published**.
- S. Mojahed, R. Drouin, and L. Sboui, "ODACE-RMS: A Remote Web-Based Platform for Automated Multi-Device Android Testing and Certification", IEEE Access, **accepted, May 2025**.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Lokman Sboui, for his unwavering support throughout both my master's studies and my internship. His continuous guidance, mentorship, and encouragement have helped me through every stage of my journey. I'm truly thankful for everything I've learned from him.

I would also like to acknowledge the invaluable contributions of the certification of wireless devices laboratory team and interns at Vidéotron. Their collaboration and assistance were instrumental in the development of ODACE. I am especially grateful to my colleague Réjean Drouin, senior certification engineer at Vidéotron, whose generous support, guidance, and encouragement were key to helping me face and overcome many challenges. My sincere thanks also go to my manager, Yuan-Peng Ssuto, Senior Engineering Manager at Vidéotron, for his understanding and support over the past two years.

A special note of appreciation goes to my second family—my friends from the Canadian Institute of Islamic Civilization (CIIC). Their help and moral support lifted me through every moment. I am deeply grateful for their presence and the countless ways they helped me continue.

I extend my heartfelt thanks to my family. To my mother, father, and mother-in-law, your endless prayers, encouragement, and emotional support gave me the strength to continue. To my brothers and sisters, your endless love and uplifting words kept me going, even as you face hardship under occupation in Palestine. Your resilience and dignity inspire me daily.

To my brother and sister-in-law, thank you for being by my side, always ready with encouragement and help whenever I needed it.

My deepest and most heartfelt appreciation goes to my beloved husband, Saif, whose patience, unconditional support, and belief in me never wavered. His understanding, especially during the most challenging moments, was a source of strength I could always rely on. And to our precious children, Bassel and Saeda, thank you for being my light and motivation throughout this journey.

Lastly, I dedicate my deepest respect to my people in Gaza, who continue to endure unimaginable hardship and genocide under occupation. Your patience, sacrifice, and unshakable spirit serve as a profound source of strength and pride. Your resilience is not forgotten, and it always inspires me.

Une plateforme web automatisée pour la certification à distance des appareils mobiles Android

Sundos MOJAHED

RÉSUMÉ

L'évolution constante de l'industrie du logiciel a entraîné une augmentation de la complexité et du coût des tests logiciels. Ce rapport souligne le besoin critique d'automatisation dans les tests, en particulier pour la certification des appareils Android mobiles. Nous présentons ODACE-RMS (Outil D'Automatisation des tests de CErtification - Remote Multi Session), une plateforme conçue pour simplifier le processus de certification en permettant l'exécution automatisée de scénarios de test télécom complets. ODACE-RMS s'exécute comme une application sur l'ordinateur du testeur et propose une interface web basée sur Appium.

Le rapport décrit également l'architecture modulaire d'ODACE-RMS, qui combine Appium, ADB et la technologie USB-over-IP pour permettre des tests parallèles et à distance. Dotée d'un backend Spring Boot et inclure d'une interface web interactive, la plateforme offre une grande flexibilité pour la gestion de sessions de test multi-appareils, que les dispositifs soient connectés localement via USB ou à distance via un hub USB-over-IP. Ces fonctionnalités permettent de réduire considérablement le temps de certification et d'effectuer les tests sans manipulation physique des appareils. Notre étude compare ODACE-RMS aux systèmes traditionnels et montre une réduction de 89% du temps d'engagement des ingénieurs de certification dans les tests de certification, diminuant ainsi significativement les interventions humaines et améliorant l'efficacité globale du processus. De plus, les résultats obtenus avec l'architecture proposée d'ODACE-RMS démontrent que les tests à distance ne sont que légèrement plus lents que les tests en local via des ports physiques, avec un retard moyen d'environ 7%, même lors de tests simultanés sur plusieurs appareils.

Mots-clés: Appareils mobiles Android, Appium, Automatisation, Certification, Tests multi-appareils, Tests à distance, Tests logiciels, Cadre de test, USB-over-IP.

An Automated and Web-Based Platform for Remote Certification Testing of Android Mobile Devices

Sundos MOJAHED

ABSTRACT

The evolving nature of the software industry has increased the complexity and cost of software testing. This report highlights the critical need for automation in software testing, specifically for mobile Android device certification. We introduce (ODACE-RMS Outil D'Automatisation des tests de CErtification [Certification Test Automation Tool] Remote Multi-Session), a platform designed to streamline the certification process by enabling the automated execution of comprehensive telecommunication test scenarios. ODACE-RMS runs as an application on a tester's PC, featuring a browser-based interface powered by Appium.

The report also outlines ODACE-RMS's modular architecture that combines Appium, ADB, and USB-over-IP to support remote and parallel testing. With a Spring Boot backend and including a web-based frontend, the platform enables flexible multi-device test sessions, whether connected locally via USB or remotely through a USB-over-IP hub. These features significantly reduce certification time and allow engineers to execute tests without physically handling devices. Our study compares ODACE-RMS with traditional systems, which reduced engagement time in certification testing by 89%, significantly decreasing the need for human intervention and enhancing the overall efficiency of the certification process. Additionally, the proposed ODACE-RMS architecture results show that testing remotely is not much slower than testing locally through physical ports, even when multiple devices are tested in parallel, with an average 7% delay.

Keywords: Android Mobile Devices, Appium, Automation, Certification, Multi-Device Testing, Remote Testing, Software Testing, Testing Framework, USB-over-IP.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

ODACE	Outil D'Automatisation des tests de CErtification (Certification Test Automation Tool)
ODACE-RMS	ODACE Remote Multi Sessions
ADB	Android Debug Bridge
CE	Certification Engineers
MO	Mobile Originating
DUT	Device Under Test
SDLC	software development life cycle
VoLTE	Voiceover LTE
VoWi-Fi	Voiceover Wi-Fi
PSAP	Public Safety Answering Point
PC	Personal Computer
RF	Radio Frequency
UI	User Interface
JPA	Java Persistence API

INTRODUCTION

Telecommunication operators are required to certify each new mobile device and software update to ensure reliable service delivery to network users while meeting marketing, operational, and legal requirements. Traditionally, the certification process was entirely manual, which demanded significant time and effort for routine tasks. This manual procedure underlined the critical need for automation in the certification process. According to an earlier study by Bertolino (2007), testing costs can account for nearly 50% of the total development cost; this proves the importance of efficient and streamlined testing methods, Berihun, Dongmo & Van der Poll (2023). Especially considering that nearly 5 million mobile devices are sold every day, Godboley, Dalei, Sadam & Mohapatra (2023), the demand for mobile technology continues to grow. Additionally, the total number of mobile users has reached 5.75 billion in recent years, with expectations for significant growth in the near future, Kemp, Simon (2024), AbuSalim, Ibrahim & Wahab (2021).

Until 2022, all mobile device certification tests at Vidéotron were performed manually by the certification engineering team, which included senior engineers, junior staff, and interns. These routine and repetitive tasks significantly consumed the time and effort of the team. As the volume of devices and test scenarios increased, developing a new automation tool began. This led to a noticeable improvement in productivity, with approximately 60% of the certification tests becoming automated. However, challenges remained, particularly the inability to test multiple devices in parallel and the growing need to support remote testing, driven by the company's adoption of a hybrid work model. These limitations highlighted the need for a more scalable and flexible solution.

Motivation

Automation of certification is expected to increase productivity while simultaneously reducing costs by saving the certification engineers (CE) time and effort and reducing the risks of human errors, Chaves, Oliveira, Tiago & Castro (2024), Halani, Kavita & Saxena (2021). In addition, automated testing is a highly efficient alternative to manual testing that offers several other advantages, as follows: Providing greater flexibility for testing, handling repetitive tasks, Thant & Tin (2023), making debugging easier, providing more accurate results, offering reusable processes, automatically recording test results (execution logs, counter results, summaries, etc.) and providing standardization for testing executions.

These points motivate us to address new ideas that were not implemented previously, such as using the traditional automation tool to automate the testing of mobile applications to enhance the mobile phone certification process. This involves ADB (Android Debug Bridge) and Appium for automated control and testing of the device functions and features. Another idea is to integrate dedicated mobile phones acting as bots to fully automate telephony scenarios.

Research Problem

With the rapid global growth of mobile devices such as smartphones, tablets, and smartwatches, the need for device certification has become increasingly critical. Each new device model or software update must be validated to ensure compatibility with telecom networks and meet the regulatory standards. However, the current certification process remains manual, making it time-consuming and difficult. While several automation platforms exist, they mainly focus on application-level testing and do not support the device functions and network tests required for device certification. This gap highlights the need for an automated certification platform that can address all test scenarios.

General Objective

To adapt to ongoing developments, we aim to develop an automation platform to assist certification engineers. The primary goal of this platform is to bridge the gap between traditional Appium-based solutions, which are used for application testing, and the comprehensive requirements for mobile device certification. Minimize manual intervention, while ensuring accessibility and ease of use for all users, and deliver notable improvements in efficiency.

The main objective of the solution is to reduce overall certification time and resources, while enhancing flexibility in the testing process.

Sub-Objectives

To achieve the general objective, we define the following three sub-objectives:

- Objective 1: Automating Device Certification Testing.
- Objective 2: Enabling Parallel Automated Test Execution.
- Objective 3: Implementing Remote Automated Testing.

Research Questions

To address the research problem and achieve the objectives, we pose the following research questions:

RQ1: How feasible is it to automate mobile device certification tests to reduce manual effort and time?

RQ2: Can parallel automated testing be effectively applied to speed up the certification process?

RQ3: Can the proposed framework improve testing behavior and offer more flexibility for certification engineers?

Methodology

This report proposes a new automated solution for mobile device certification with remote and multi-device capabilities. To achieve the research objectives, the following methodology was adopted:

Literature Review A detailed analysis of existing mobile automation tools and solutions was performed to identify their limitations, particularly their lack of support for device-level certification tests and remote or parallel execution.

Solution Architecture and Implementation We are designing an automation platform to support telecommunication operators in the mobile device certification process. This solution is being developed using Java and uses ADB and Appium with a black-box testing approach, making the system more secure and accessible, even for users with no technical background. Moreover, employing bot devices to handle the main commands without user intervention. This aligns with the sub-objective of reducing certification time and minimizing user interaction.

To support multi-device and parallel testing, the platform is being extended with Spring Boot, enabling a web-based interface and integration with multiple Appium servers. This allows simultaneous execution of test sessions across several devices and significantly improving the performance.

For the remote testing sub-objective, we integrate USB-over-IP technology to allow certification engineers to connect to devices remotely with screen copy application, so users can interact with and control the DUT without physical access.

Additionally, to optimize system performance we need to minimize the number of ADB commands, which helps lower CPU and memory usage, contributing to the sub-objective of enhancing platform efficiency.

Data Collection The data used in this study should collect from real certification tests performed in Vidéotron's laboratories. Some datasets could be retrieved from previously executed test logs stored between 2022 and 2024, which were originally generated and archived for internal engineering use. Additional data need to collect specifically for research and evaluation purposes in collaboration with certification engineers and interns.

Comparative Evaluation The assessment focus on performance efficiency, resource utilization, and overall user experience. Quantitative evaluations perform across several key metrics, including engagement time, execution time, CPU and memory usage. Specific attention should be given to measuring the number of test executions since starting to run the tool in the team between 2022 and 2024, comparing automated versus manual testing processes. These metrics also cover various scenarios such as single versus multi-device sessions and local versus remote connectivity. In addition, a qualitative evaluation was conducted through user surveys to assess the perceived efficacy and usability of the remote feature among certification engineers. This is very important to ensure the solution achieve the general and sub objectives.

Expected Outcomes

The proposed platform is expected to deliver several key outcomes that align directly with the defined sub-objectives. First, in line with the sub-objective to ensure accessibility, we aim to develop a flexible and user-friendly framework that addresses the limitations of existing solutions, with a specific focus on mobile device certification testing. Second, to fulfill the sub-objective of automating and streamlining tests, the platform is designed to reduce test execution time compared to traditional manual testing, minimizing human error and avoiding the need to repeat the same tasks during each testing cycle. Third, supporting automation and usability, extract a report and summary of the testing flow after each execution for better debugging and review. Fourth, by introducing automation and enabling remote access, the platform will enhance the efficiency of certification engineers, saving both time and effort. Finally, in response to the

sub-objective of supporting multi-device, the solution is expected to support parallel testing of multiple devices up to the maximum typically required without significantly impacting the performance of the PC.

Outline of the thesis

This chapter provides a general introduction. Chapter 1 reviews the related work relevant to the research scope. Chapter 2 provides an overview of the certification process within the telecommunications industry. Chapter 3 presents the proposed automated certification solution, while Chapter 4 describes the remote testing and multi-session capabilities. Chapter 5 discusses the experimental results, the outcome, and the challenges of the platform. Finally, the last chapter concludes the work and highlights potential directions for future research.

CHAPTER 1

LITERATURE REVIEW

This chapter reviews existing literature related to automated testing frameworks for mobile applications, with a specific focus on their applicability to mobile device certification. It identifies current tools and frameworks, highlights their strengths and limitations, and defines the gap that this research aims to address.

1.1 Automation Testing Tool

Automated testing in the mobile devices industry has become more important and widespread. In fact, many tools have been built to automate parts of the testing process, such as Appium, an open-source tool that uses the WebDriver protocol to perform testing of iOS and Android applications, Li & Li (2022). For instance, Calabash is an automation framework that aims to perform automated UI acceptance tests on the Android and iOS platforms, Berihun *et al.* (2023). Also, Frank is an automated acceptance testing framework for testing iOS applications, Vajak *et al.* (2018). On the contrary, Robotium is specifically designed to automate Android application UI test cases, Sinaga, Pratama, Siburian *et al.* (2021).

As mentioned previously, many testing tools have been developed and have become widely used to test different software applications, Vajak *et al.* (2018). Four tools are compared and evaluated in Table 1.1 based on the most crucial features that determine their suitability as the foundation of the testing environment. As shown in Table 1.1, Appium outperformed the other tools, fulfilling more feature requirements and supporting a more comprehensive range of programming languages.

Appium is one of the most critical tools used nowadays. It is an open-source tool used for automating testing and utilizes web driver protocols to execute Android and iOS applications.

Table 1.1 Testing Tool Features, adapted from Vajak *et al.* (2018).

Features	**Testing Tool**			
	Calabash	***Appium***	***Frank***	***Robotium***
Open source	+	+	+	+
Android	+	+	-	+
iOS	+	+	+	-
Black box	-	+	-	+
Server-client	+	+	-	-
Languages	Ruby Cucumber	Ruby Python Java JavaScript Objective C PHP C#	Cucumber	Java

The primary structure of Appium consists of two components: the Appium server and client, which interact with bootstrap.js. Figure 1.1 presents the structure of Appium in iOS vs. Android.

1.2 Automated Testing Environment for Mobile Applications

The development of an automation framework to test software with less effort and cost is a crucial goal of multiple previous works. Numerous automation tools for application testing have recently emerged. However, we identified a gap in the availability of automation tools specifically designed for the certification process in the telecom sector. Furthermore, we found a lack of platforms or tools dedicated to testing mobile phone devices' functionalities beyond just

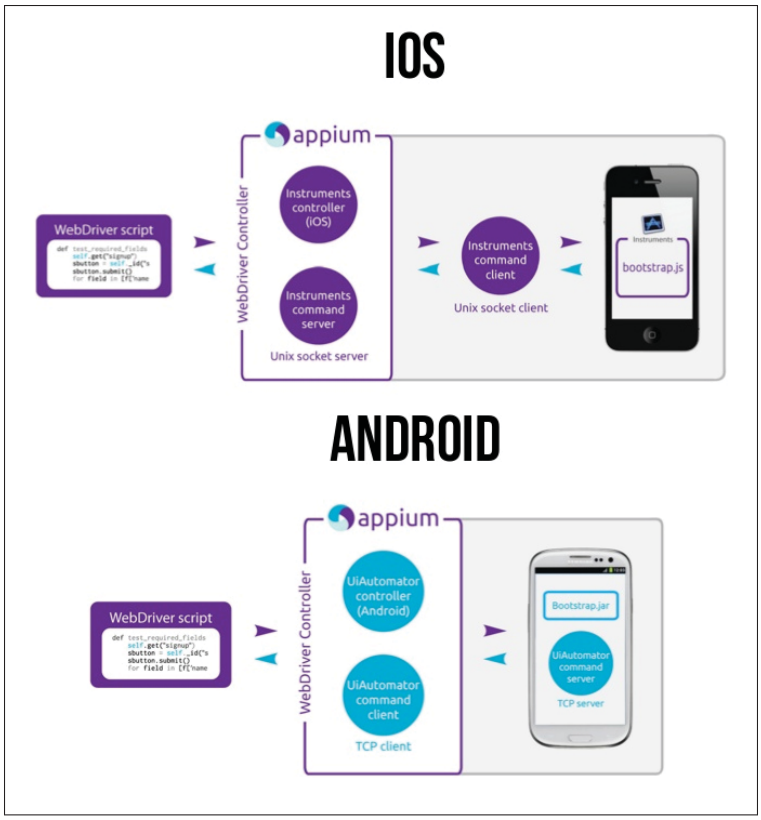

Figure 1.1 Appium supports iOS schematics vs. Appium supports Android schematics (This figure from Wang & Wu (2019)).

the applications. For instance, Vajak *et al.* (2018) presented a novel idea for automation tests, which is a simple environment that uses efficient libraries to reduce code lines and execution time. The architecture of their solution presented in Figure 1.2 is based on a server-client model, which contains the server computer (macOS) that hosts an Appium server that will be connected with IOS and Android devices to test the two platforms simultaneously. Additionally, the client computer with the Appium client facilitates the sending of instructions from the Python test script and receives the responses.

Their work can be a base upon which to build for future work. One potential future work is addressing this testing environment's limitation, which is its capacity to test native applications.

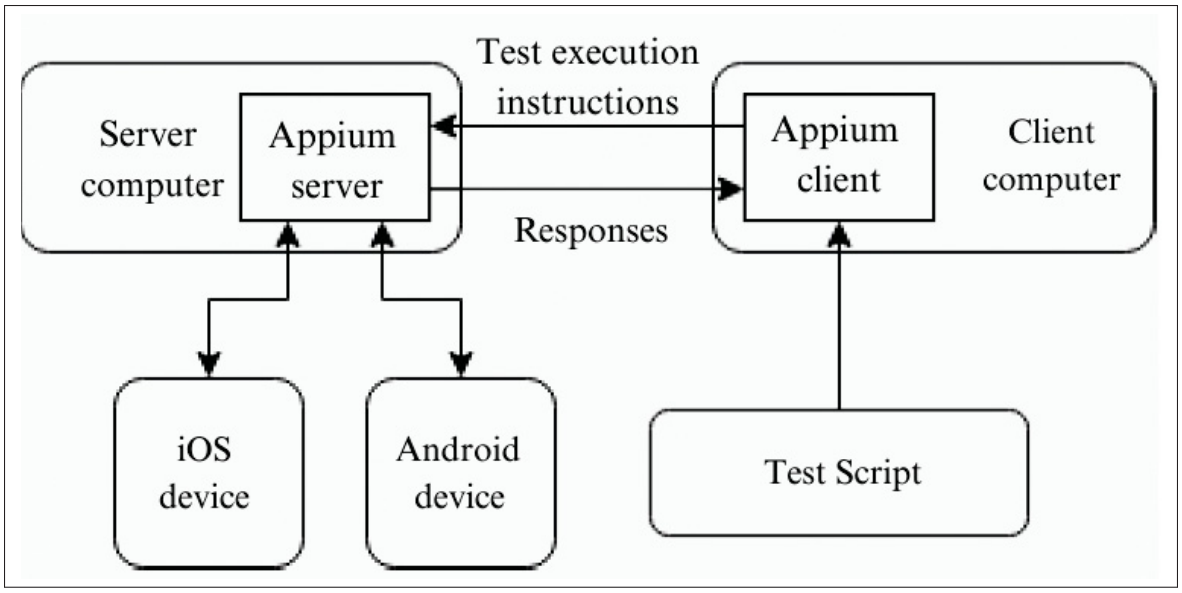

Figure 1.2 Architecture of The Proposed Testing Environment of Vajak *et al.* (2018)

A valuable contribution would be generalizing the solution to include testing of all applications (native and non-native). Also, to extend the work, the researchers should find a way to test iOS and Android under the same conditions and the same script file. In addition, reducing the execution time is a possible idea to develop in further works, especially for the time the environment spends loading the libraries. Moreover, the user interface is essential to allow users with low technical experience to use this environment. This proposal starts the tests from the terminal, limiting the experts' solution.

A further framework proposed by Salam, Taha & Hamed (2022) discusses an advanced automated functional testing framework for mobile applications designed to be easy to use with a testing process throughout the software development life cycle (SDLC). Also, this design proposed new ideas and solutions for issues in previous tools and frameworks. Figure 1.3 presents their advanced design framework using Appium

This solution used the MAC server to build the framework on both mobile platforms (Android and iOS). Prepared the device by installing all requirements, then created the five packages (baseEnv, tests, screens, resources, and utilities). Each package was responsible for a specific function. Firstly, the limitation of the particular server port must be addressed. The Appium will be started pragmatically. Also, the design suggested creating instances for some libraries to be reused during the testing process. All libraries were added to the utility package. The resource package contains the configuration file that should be updated before executing the test script, as well as the execution files for Android and iOS. The baseEnv package contains the base class BaseEnv, which is the base class that all other classes will inherit. So, all classes will use common instances and methods. Accordingly, when the test starts running the main class, it installs all requirements and imports the libraries. Then the tests will be executed. After each case, a log file will be created and stored in a Logs folder.

However, if the test fails, the video of this case will be recorded and stored in a captured videos folder. Then a ticket in Jira will be opened automatically. The log file, the video, and all details will be attached. By the end of all scenarios, the test process will be finished, and the framework prepares three types of reports: TestNG reports, Extent reports, which have many diagrams for the test result, metadata, and a summary of the test execution. Also, this report classifies the tests by category. Allure reports that explain the number of tests, status, time, and all test steps.

Nevertheless, this solution uses many dependency libraries, so that maintenance will be more complex in the future. To tackle this concern, forthcoming enhancements could include a class to copy the necessary parts from the library and load them during the application's initialization. This way, the duration of loading the library can be reduced compared to loading the libraries each time the test calls it. Moreover, they needed to explain more about the interface and how the user can operate the proposal. Also, as mentioned in their conclusion, AI will make the framework smart and reduce human effort.

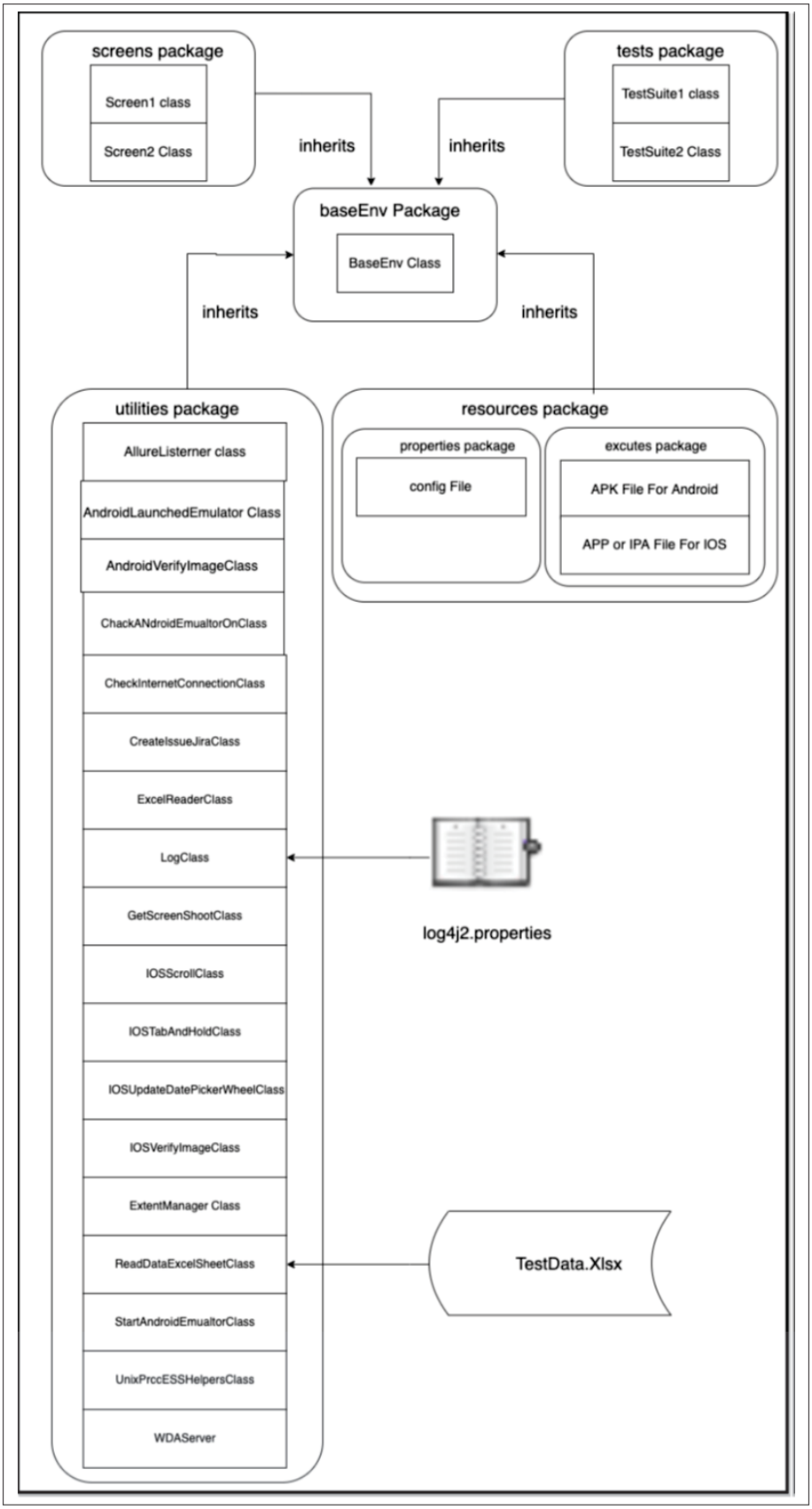

Figure 1.3 Architecture Diagram of Automation Testing Framework by Salam *et al.* (2022)

Another work by Alotaibi & Qureshi (2017) suggested a data-driven test framework based on Appium presented in Figure 1.4.

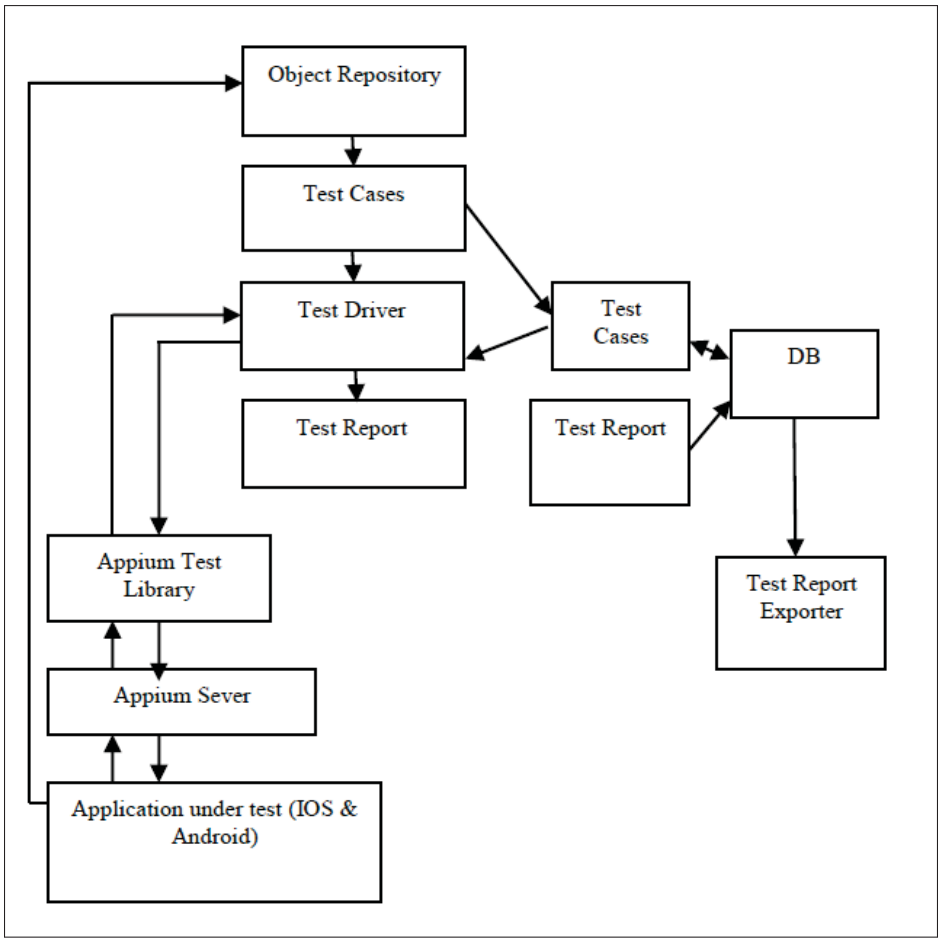

Figure 1.4 Architecture of Proposed Solution by Alotaibi & Qureshi (2017)

This solution includes Appium Library, Repository, Report, Test Driver, Appium Server, and Exporter. The novel framework applied an architecture based on a client/server model. So, the client establishes an automation session with the server and sends the desired capabilities. The server initiates the session and responds with the session ID. Additionally, the behaviour of the server is determined by the desired capabilities. The Appium server executes and connects the script with the simulator/emulator. The new framework is developed in Java, and each package is responsible for a specific task. For instance, the TestCases-Recorder package is

designed for building the test cases class, the TestReportGenerator package is responsible for generating report classes, and the database package is responsible for storing data. The SerME database was chosen to implement the database according to its functionality of processing multi manipulations of data. Firstly, the spy in the Repository collects the object's properties in the interface of the under-test application and stores them in the Repository. Then the Driver executes the test scripts stored in the database, compares the test data with the test scripts, and performs various operations on the application's UI using the Appium test library. Subsequently, the Report generator generates a report based on the run details. It saves it in the database, which The Exporter will send to an external file, which can be displayed on a mobile device.

However, the solution needs more technical details to be clear, and the article should have discussed the framework's user interface and provided examples or test cases. Furthermore, the framework may require a certain level of technical expertise and knowledge of programming languages, which could potentially limit its accessibility to non-technical users.

For testing iOS application, the AirMochi solution introduced by Lukić, Talebipour & Medvidović (2020), is a versatile and platform-independent tool designed for automated testing and vulnerability analysis of iOS apps. It enables remote control, video stream recording of app interfaces, and timestamps UI events. Additionally, its Model Extractor component extracts behavioural models of apps for further analysis. The tool produces accurate and valuable behavioural models. Figure 1.5 shows the solution design, which is built from six-level modules.

However, the solution is complex and uses many technologies and languages, which could be challenging to debug. It is limited to iOS and requires a certain level of technical expertise and knowledge to use it effectively.

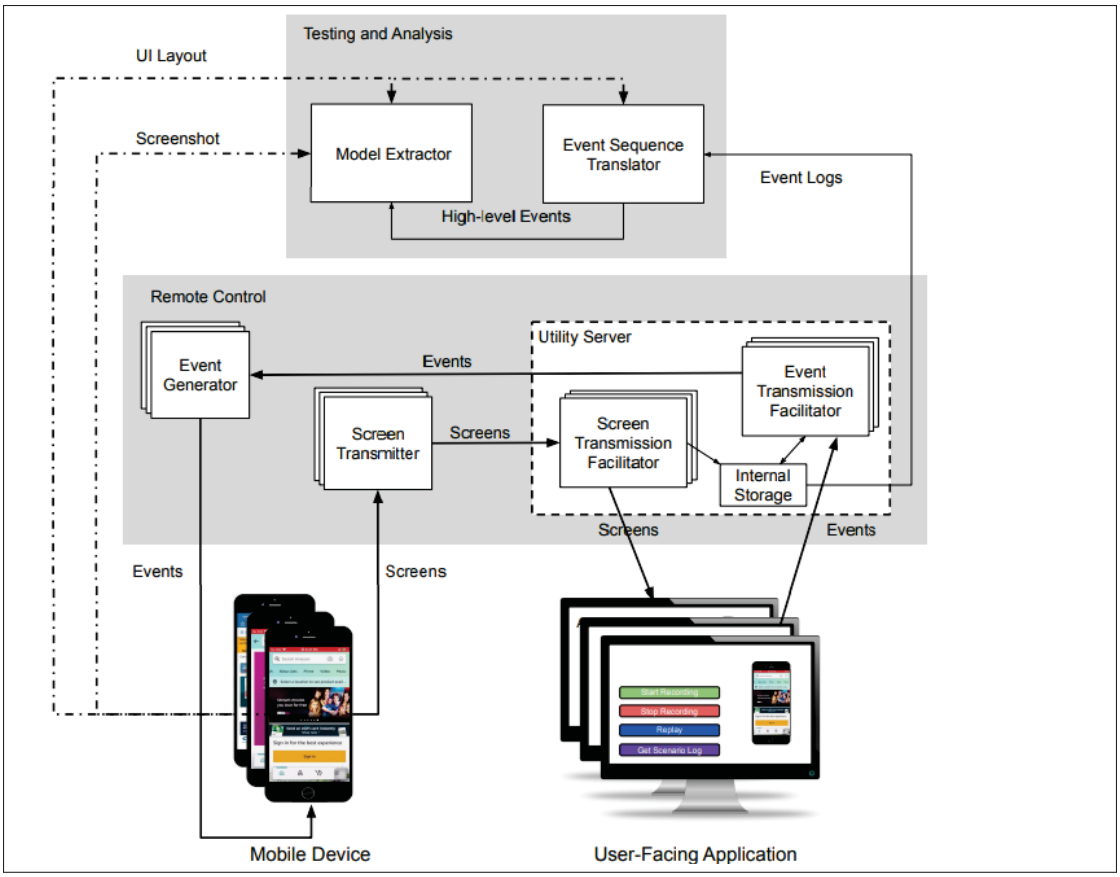

Figure 1.5 Architecture of the layer of the Solution by Lukić *et al.* (2020)

1.3 Remote Parallel Testing Solution for Mobile Applications

Kaasila, Ferreira, Kostakos & Ojala (2012) introduced a solution called Testdroid, an online platform that allows users to test their Android applications remotely across different smartphone models and versions in parallel, without the need to buy and maintain all device types. The platform hosts a pool of physical handsets connected to online servers, and users can chose one or multiple devices by the platform interface then execute their tests as shown in Figure 1.6.

Testdroid is based on recording UI scripts using the Robotium automation framework, then executing these scripts remotely on different devices. However, execution is not immediate, the tests enter a queue and are executed once the devices are available. While each device runs one

test at a time, users can send their tests to multiple devices simultaneously to achieve parallel testing. After execution, users receive results including the test status, screenshots, and execution logs.

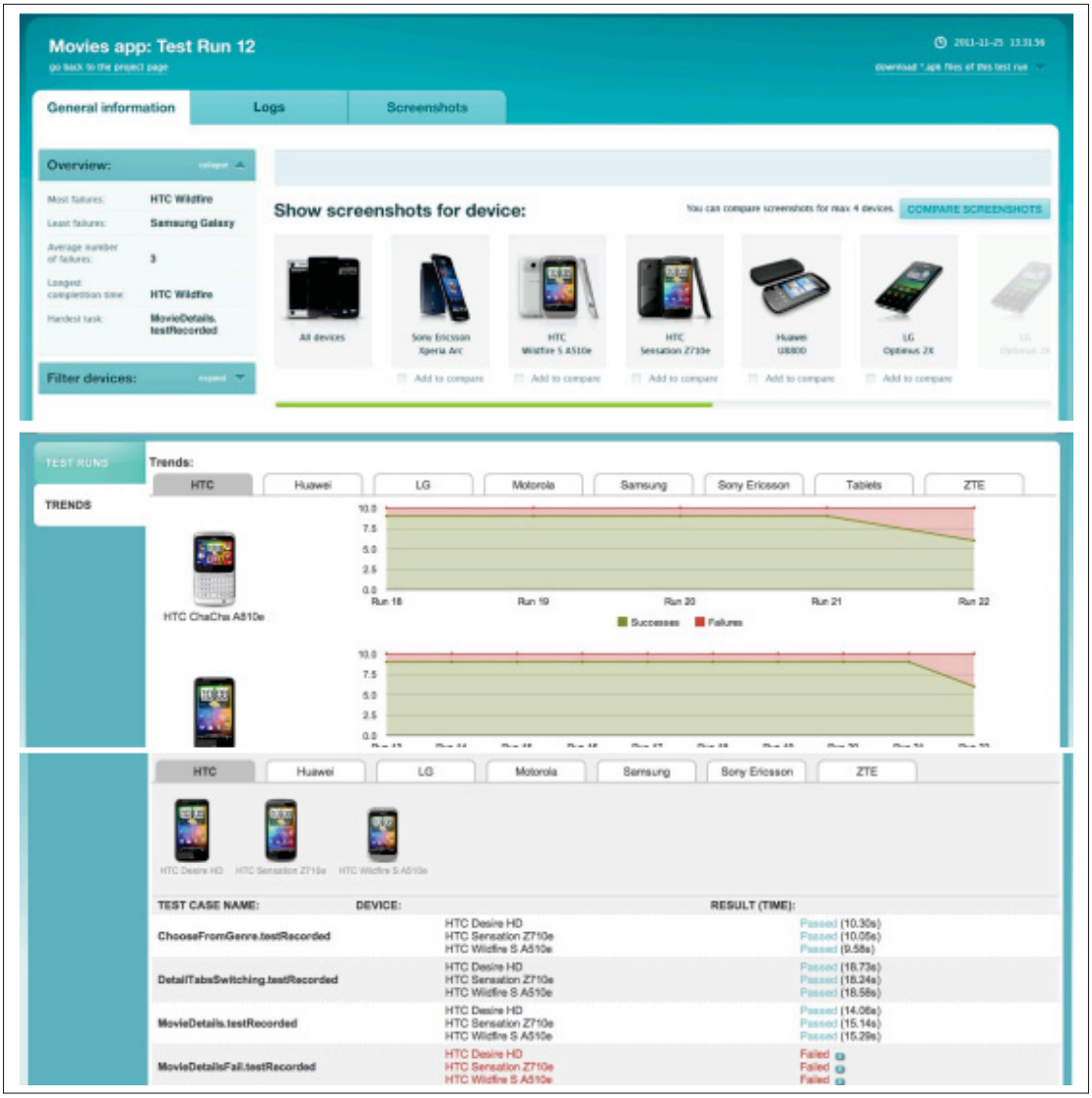

Figure 1.6 Testdroid Platform by Kaasila *et al.* (2012)

This solution addresses an important challenge in remote and parallel testing but it is limited to Android applications. It does not support all application for example could not execute test scenarios that require voice input. Additionally, the remote hardware architecture is complex

and requires significant server infrastructure. Also, parallel testing may experience delays due to the queuing system.

1.4 Different Tools and Experimental Approaches

Some earlier studies by Kim, Choi & Yoon (2009) investigated testing and automation and looked at the performance of these tools and the vulnerabilities of testing software tools.

In addition, Verma (2017) discussed using Appium to test mobile applications, but this work did not implement a touch action instance for each test case. Also, there are no log files for each test executed in this solution.

Moreover, Singh, Gadgil & Chudgor (2014) explained the importance of the Appium testing tool in efficient software and bug-free and quality-rich applications.

Motwani, Agrawal, Singh & Shrivastava (2015) developed a framework for browser compatibility testing using Selenium WebDriver. However, none of the previous articles addressed the scenarios where Appium might not be the optimal choice.

Another mobile automation framework was designed and implemented to be compatible with both iOS and Android platforms by Cui (2024). This framework was built based on Appium, and the user could use it to customize it and automate the testing of their own application. This solution does not cover the automation testing for the device and network functions.

Additionally, Rao, Prasad & Rao (2012) proposed a method to develop the testing process. Also, Jain & Sharma (2012) created a test-driven automation framework by creating keyword-based test cases.

Tran, Hieu Minh and Ninh, Tuan Duc and Tran, Thinh Duc and Van Ngo, Vuong and Nguyen, Linh Duc (2023) published an article about a framework based on Appuim to automate the

testing of IP multimedia subsystems, which focuses on automating service validation such as voice calls and messaging within telecommunications systems. This solution could include some other tasks, like taking videos of the process and rebooting the device. However, the solution does not cover testing other device functions or technologies beyond the IP Multimedia Subsystem.

Segron's Automated Testing Framework, SEGRON (2024) is a solution designed for telecommunications services and device testing. It supports different technologies, including 5G, VoLTE, and it offers automated test execution on various devices and networks. However, it is limited to testing the network functions without the other device's essential functions.

Limitations of Existing Solutions

While the previous works offer robust frameworks for UI testing, they fall short in automating the comprehensive telecommunication testing required for certification, Yoon, Feldt & Yoo (2023). Most existing platforms are limited to application-level testing, and dose not cover the essential device functionalities, including voice calls, SMS, network connectivity, and system setting modifications such as Bluetooth, airplane mode, and others, which is required for the certification process, Yu *et al.* (2024), Liu *et al.* (2024). In addition, the load times of libraries before each test run were a big challenge, which increased execution time and testing the user interface, Salam *et al.* (2022). Additionally, there are no log files to monitor and track the test execution process, Verma (2017).

To address these limitations, we aim to introduce a platform to test all mobile telephony functions and both native and non-native applications with enhanced execution efficiency.

Our Key Contributions

- Specialized Device Certification: While existing tools focus on application-level testing, we introduce a new framework which focuses explicitly on device certification requirements to cover not only application-level testing but also the device functionalities and features, ensuring comprehensive validation for every new device software update. This specialization fills a gap not covered by general mobile testing solutions and reduces the manual testing, and engagement time for the certification engineers.
- Optimized Test Execution Architecture: The solution introduces a session-based initialization process that pre-loads required libraries and Appium configurations on both mobile devices and testing PCs. This significantly reduces redundant setup time across multiple tests and reduces execution time.
- Parallel Testing: The platform supports simultaneous testing sessions across multiple devices, enabling users to execute parallel tests on different devices without waiting for prior sessions, which improves productivity.
- Remote Testing: Incorporates a remote access feature that enables certification engineers to perform certification tasks and do their tests without being physically present in the lab. This is especially relevant in hybrid work environments, which provide more flexibility.
- Accessible Interface: Offers a user interface specifically engineered to accommodate users with different technical skill levels, lowering the difficulty of certification workflows.

Table 1.2 presents a quick comparison between the existing tools and our proposed solution

Table 1.2 Feature comparison of mobile testing frameworks

Features	**Salam et al. (2022)**	**Air Mochi**	**Alotaibi et al. (2017)**	**Vajak et al. (2018)**	**Kaasila et al. (2012)**	**Segron 2024**	**ODACE**
App-level Testing	+	+	+	+	+	-	+
Device feature Testing	-	-	-	-	-	limited	+
Support iOS	+	+	+	-	-	+	-
Support Android	+	-	+	+	+	+	+
Parallel Execution	-	-	-	+	+	-	+
Remote Execution	+	-	+	-	+	-	+
Easy to Use	+	-	-	-	-	+	+
Export the Reports	+	+	+	-	-	+	+

1.5 Conclusion

This chapter presents an overview of previous research related to automation and platform testing. It presents several automation tools and compares them. It also reviews existing platforms and solutions, and discusses their outcomes and limitations. It also introduces our proposed solution contributions. The next chapter will provide an overview of the certification process, highlighting its importance, key steps, and categories, and offering examples of some main tests.

CHAPTER 2

OVERVIEW OF MOBILE DEVICES CERTIFICATION LIFE CYCLE

While the previous chapter reviewed existing automation solutions, most of them focused on mobile applications and do not support the core device functions required in mobile certification. In this chapter, we present an overview of the certification process.

2.1 Introduction to Mobile Devices Certification

In the dictionary Cambridge University, certification refers to the process of giving official or legal approval to a person, company, product, etc., that has reached a particular standard.

To ensure quality services and adhere to legal regulations, telecommunications operators mandate certification for each new mobile device and software version update. This certification encompasses tests to validate device-network compatibility and the device's ability to perform basic and advanced functions. The tests range from assessing signal strength, call quality, and data transmission rates to battery life across various network conditions. Basic tests, for example, involve making calls, sending texts, and accessing emergency services (e.g., 911), whereas advanced tests include performing a conference call with multiple devices and verifying if the mobile device is transmitting in the correct frequency band combinations.

These certification tests are grouped into specific test plans, each targeting different technologies such as LTE, Voice over LTE (VoLTE), Voice over Wi-Fi (VoWi-Fi), 5G. At Vidéotron, the four categories of test plans are:

1. **Basic Test Plan**: Encompasses essential telephony functions (voice calls, messaging, emergency calls) and mobile data (internet browsing, email) across various mobile technologies.
2. **High-level Test Plan**: Extends the Basic category to include advanced telephony functions like call forwarding and conference calls, along with non-telephony functions such as Wi-Fi connectivity and internet browsing.

3. **Regression Test Plan**: Builds upon High-level testing with thorough coverage of most functions. Primarily utilized following significant software updates (e.g., Operating System upgrades).
4. **Complete Test Plan**: This exhaustive testing phase evaluates all device features and functionalities, including the user interface, connectivity, battery performance, audio and video quality, roaming behaviour, Wi-Fi APN settings, and more. Conducting complete test plans ensures that both the device and its initial software iterations comply with company standards and governmental regulations.

Figure 2.1 shows the structure for the test categories and how each category contains the previous one plus other tests.

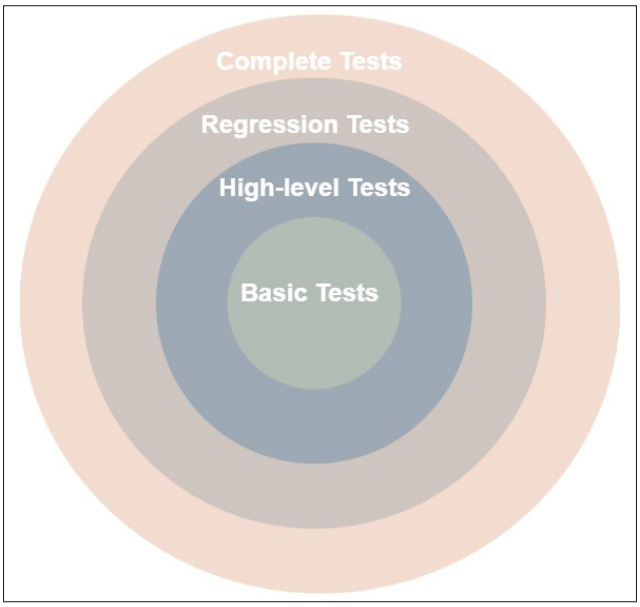

Figure 2.1 Tests Categories

The aim of having different test plans is to cover each certification scenario with different technologies like LTE, 5G, VoLTE, and VoWi-fi, as well as device testing scenarios. For instance, the launch of a new device requires the Complete test category for each technology, while software updates are certified with Regression, High-level or Basic.

2.2 Certification Process and Prerequisites

The certification process starts with the reception of a new device or software from the device manufacturers. The certification engineers prepare the devices before creating a ticket in a test management tool (e.g., Jira) with the relevant information about the task, adding the test list based on technology types and levels of certifications, and then start conducting manual certification tests. Some functions, such as network connectivity and telephony functions, must pass testing with success to obtain certification approval. The process involves the following steps:

1. Receive new units of device models, label them and document their details in the inventory.
2. Review the handset requirements form previously filled out by the manufacturer.
3. Evaluate the level of testing required based on the software Release Notes.
4. Update the software version of the devices.
5. Create the test campaign in the test management tool.
6. Retest and update previously reported anomalies.
7. Execute the tests and analyze the results. In case of new urgent anomalies or issues discovered, manufacturers are contacted immediately.
8. Prepare the certification report to determine whether the device has passed the certification test process. If accepted, the new device and software are documented in the certification database.

This process takes a few days since certification engineers perform all these steps manually, which costs time and effort. This explains the need for a tool or platform that can save time, cost and effort while maintaining the efficiency and accuracy of the certification process.

2.3 Certification Tests Examples

Before automating the certification tests, it is important to note that they follow the same sequence as manual executions. Verifying that each step is completed as expected before going to the next. For instance, to test the Call Function (Figure 2.2), the corresponding steps are:

- Preparation :
  - Update the DUT (Device Under Test) with the correct software version.
  - Insert a valid SIM card into DUT.
- Execution :
  - Initiate an Outgoing Voice call.
  - Verify that the call can be connected successfully.
  - Verify that the radio technology corresponds to SIM and DUT capabilities.
  - Verify that the call stays connected for 60 seconds.
  - Verify that the call termination completes correctly.
  - Repeat with an Incoming Voice Call.
- End of the Test

Testing the Email Application example, (Figure 2.2), the corresponding steps are given by:

- Preparation :
  - Update the DUT with the correct software version.
  - Insert a valid SIM card into DUT.
  - Activate Mobile data.
  - Turn Wi-Fi off.
- Execution :
  - Open the email application.
  - Create a new email.
  - Set destination address and subject.
  - Attach a file to the new email.
  - Send the email.
  - Wait for a reply email.
  - Open reply email.
  - Verify the attached file.
- End of the Test

If a test step fails, the test is declared Failed, and the event is documented as an anomaly reported in the test management tool and communicated to the mobile device manufacturer.

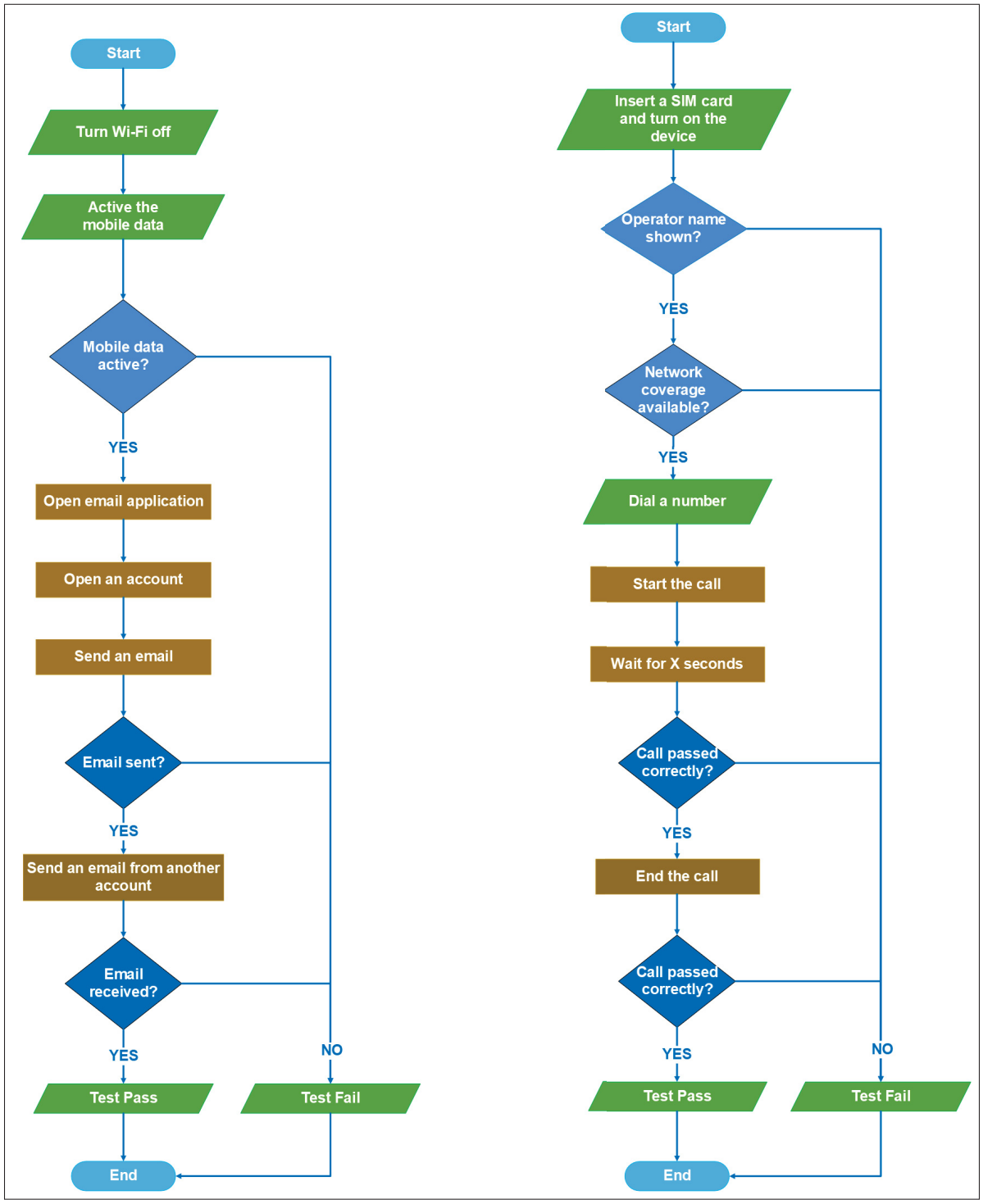

Figure 2.2 Flow Chart of Testing the "Call" Function and the "Email" Function

However, some tests could be more complex such as the 6-party conference call test (conference call with five parties plus one call on hold). This test aims to validate the ability of a device to handle a complex conference call involving multiple participants. The test must be done using other parties with different voice technology, different directions (incoming, outgoing), and the order of exit.

The test follows these steps:

1. Call Connecting: Each participant is sequentially added to the conference with Outcome and status are verified at each step.
2. Conference management: Test the different features by swapping participants, merging calls, and verifying the participant limit (up to 6 participants).
3. Interaction validation: Validate the main actions such as adding a call on hold and ending a call from the device tested or from the other party.
4. Results: The test evaluates the robustness and compliance of the device under test in the face of complex conference call scenarios.

Another example is Access Point Name (APN) management test. This test verifies the Access Point Names (APN) configuration handling of the Device under test. The user execute the test by following these steps:

1. Navigation to APN Settings: Navigate to the "Access Point Names" (APN) settings in the device's UI.
2. Verify the Default APN:
    - Verify that DUT activates the correct default APN.
    - Verify that default APN can not be modified.
3. Verify the Custom APN:
    - Create a new custom APN.
    - Save custom APN.
    - Verify custom APN integrity.
    - Activate custom APN.

- Verify call and data functionality with custom APN active.

4. Reset default APN:

   - Use the “Restore default values” function.
   - Verify that custom APN is removed, and default APN activated.
   - Verify call and data functionality with default APN active.

This method ensures that the device’s APN settings are correctly configured, and the user can define and use custom APN.

2.4 Conclusion

This chapter aims to clarify the certification process by providing a detailed description of its purpose and importance for telecommunications operators. It explains the different levels of certification and outlines the steps that certification engineers follow to test a new mobile device or an OS update, providing details with two examples of the basic texts for each certification. All of these procedures are performed manually. The following chapter presents the proposed solution to automate part of this process to reduce user interaction and make certification more efficient and accessible.

CHAPTER 3

PROPOSED AUTOMATION OF CERTIFICATION TESTS

3.1 Introduction

The previous chapter mentioned that network operators must certify mobile devices to meet various requirements and specifications. The certification process can include more than 400 tests, leading to long execution times for each certification. Therefore, we aim to design a solution that reduces the time and effort spent on mobile device certification.

To achieve Research Objective 1 (Automating Device Certification Testing), we design a solution to enable certification engineers (CEs) to configure and start automation processes for each DUT quickly. The solution streamlines test execution, allowing CEs to focus on other tasks during the process. The proposed solution also simplifies the repetition of tests for performance testing. Moreover, we design the tool to be user-friendly and accessible to all users, requiring no programming or technical skills.

3.2 Automation Design

The automated certification testing process aims to enhance the efficiency of mobile device testing. This process involves the following key components:

1. Automated Test Script Development and Software Environment: Scripts are developed to automate repetitive and standardizable tests. These scripts simulate user interactions with the device, such as navigating menus, initiating calls, or sending messages. The automation framework requires a stable software environment (like Appium).
2. Hardware Setup for Automated Testing: This includes setting up devices in a controlled environment where they can be remotely accessed and tested.
3. User Interface for Test Execution: A user-friendly interface developed for testers to easily initiate, monitor, and analyze test runs. This interface also allows for manual intervention when necessary.

4. Reporting and Analytics: Post-test execution, the system automatically generates detailed reports and analytics, providing insights into test coverage, defect trends, and other key metrics.

We developed the solution to be based on Android Debug Bridge (ADB) and Appium, an HTTP server that interacts with Android clients using frameworks like Google's UiAutomator or could be run across iOS devices, Li & Cao (2023). Additionally, we employ Scrcpy, a screen mirroring tool that enables test monitoring and recording on a PC.

3.3 Automatability of Tests

The testing process for mobile devices encompasses a variety of tests, some of which can be automated, while others require manual/human intervention. For instance, the Emergency Call test necessitates human interaction with Public Safety Answering Point (PSAP) agents, making automation infeasible. Likewise, tests involving Google Maps and Voice Messaging systems resist automation due to the need for dynamic responses to voice instructions.

Additionally, the Factory Reset test requires manual reactivation of the USB debugger mode. Furthermore, both video call and audio tone tests require human supervision to verify audio and image quality. Most tests are performed under live commercial cellular coverage, requiring control only over the tested device. However, a subset of tests must be executed under controlled radio environments, such as lab Radio Frequency (RF) boxes, preventing automation. Additionally, certain physical actions, like swapping SIM cards or connecting headphones, are manual by nature and thus resistant to automation. There are also tests that remain unautomated due to their infrequent use and the lack of substantial benefits that automation would bring. In addition, some tests can be partially automated to reduce manual interactions. Therefore, we propose an automatability percentage that indicates whether the test can be automated or not.

3.4 Hardware Architecture

The ODACE platform has developed significantly to improve efficiency and scalability. The original ODACE hardware architecture comprised three main components:

1. Personal Computer (PC): Serving as the system's server, the PC run the required software and supports any operating system to execute the certification processes. Android SDK and Appium must be installed. This component is responsible for detecting connected devices, hosting the automation solution, and establishing the connection between the PC and the device under test (DUT) to start the automation process.
2. Device Under Test (DUT): The DUT is the primary component. The certified engineer (CE) connects it to the PC via a USB cable to initiate the certification process. It requires an active internet connection through both a Wi-Fi access point and mobile data over a cellular network to perform the tests.
3. Bots: We propose the concept of Bots, which are Android mobile phones dedicated to executing the tests and interacting with the DUT autonomously, eliminating the need for manual intervention.

   Traditionally, certification engineers (CEs) need to use multiple devices with different technologies to certify a single device, which consumes time and resources. To address this challenge, we employed a group of Android-based devices from different brands, configured like dedicated "robot phones." These devices are connected to a charging hub to ensure they are always available. Each device is equipped with a SIM card from a different technology (e.g., 3G, LTE, VoLTE) to cover all required test scenarios. At Vidéotron, we currently use a pool of 15 such devices.

   In the example of call testing, an additional device other than the DUT is required. ODACE implements a novel solution to test telephony functions based on the 'bot' phones instead of relying on other DUTs.

   The bots are programmed to execute various tasks using the Automate Android application. Custom-made automated "flows" handle tasks like receiving and sending SMS and MMS, initiating and answering calls, etc. For example, when a bot receives a call, it searches for

the correct Automate flow and follows the designed steps automatically. As shown in Figure 3.1, the Bot will wait for 2s and then answer the call.

Bots are available in a pool and are listed with their MSISDN (phone number) and capabilities - radio technology, voice codec and services - in a shared document. ODACE selects bots randomly and updates their success/fail counters to automatically eliminate those that are not working correctly from the pool.

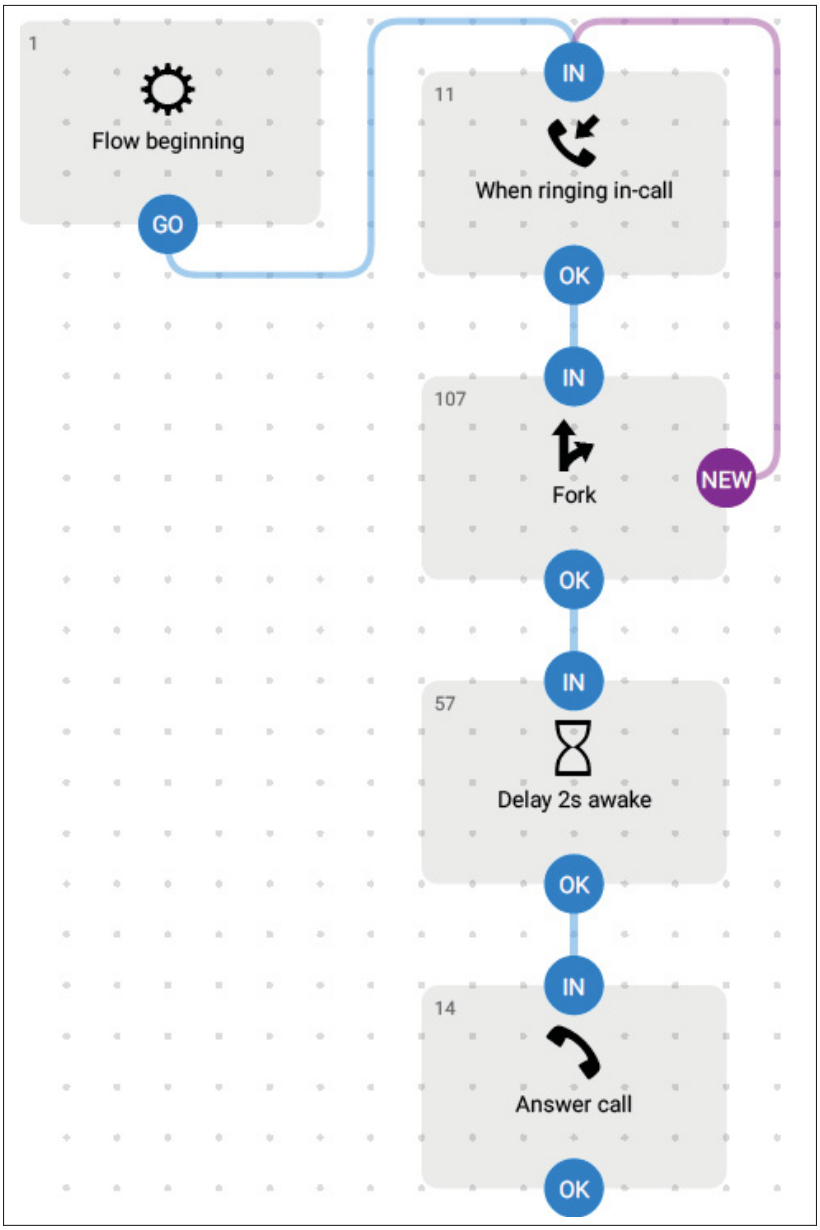

Figure 3.1 Incoming call flow chart - Bot

Figure 3.2 shows the hardware architecture components.

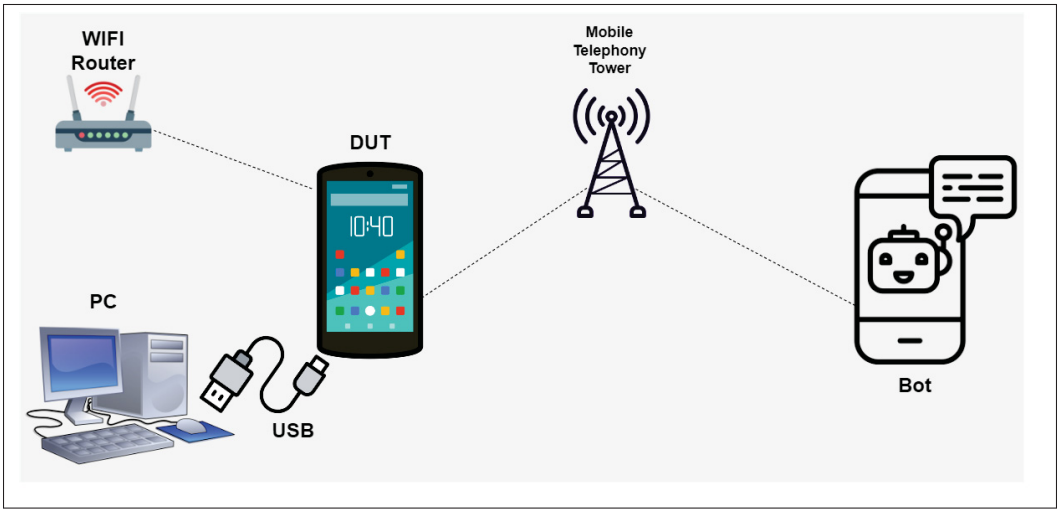

Figure 3.2 ODACE Hardware Architecture

3.5 Software Architecture

The ODACE platform is composed of several software components, each of which plays a critical role in automating the certification process for Android devices. Following, we outline the key components:

1. Device Under Test (DUT) Software: Device manufacturers regularly release new software or new updates for their software. Depending on Vidéotron engineers, each device model has almost a new software update every month, which could be a major update, a fault fixing, or a security edit. These updates sometimes necessitate adaptations within the ODACE software architecture to maintain compatibility with new versions. Consequently, DUT software is a crucial component of the platform.
2. Solution Frontend and Backend: It includes the following parts:
    a. Backend: We designed and developed the solution to automate mobile certification testing for Android devices using the Java programming language. ODACE utilizes black-box and object-oriented technology to execute test scripts that emulate user interactions with a mobile device to test its functionality and performance. We chose

Java as the primary programming language due to its strong integration capabilities with Android development tools and frameworks. Java support the Android Debug Bridge (ADB), which is a command-line tool that enables communication with Android devices[1]. In addition, Java integrates easily with Appium, providing rich libraries and frameworks that simplify the integration and creation of maintainable and reusable test scripts, which are required to build a strong automation platform. Also, Java is cross-platform compatible, so no need to have a specific PC requirement to run the solution.

b. Frontend: The frontend is developed using JavaFX to build a friendly user interface. JavaFX enables the development of desktop applications with a clean and dynamic layout, making the automation tool more accessible to users who are not familiar with command-line interfaces.

3. ODACE Scripts: ODACE execution is driven by script files that define the actions to be performed. Script files are provided with ODACE to cover defined test sets (Plans), others are used to execute each specific test (Tests), and a set of command files provides manipulations (Manipulations) like activating Wi-Fi, managing Plane mode, restarting the DUT, etc.

   Script files are in plain ASCII text format. Users can view, edit, and create them, providing flexibility and modularity. These script files are parsed by the dedicated LineScript class. Each line in these files is a combination of a command and predefined parameters in LineScript, which calls the required text execution methods as shown in Figure 3.3.

Our solution includes 56 script files, which are parsed and executed through more than 20,000 lines of Java code. These script files enhance the interaction between users and the platform, making it more user-friendly and easier to understand. They allow for flexible test execution and customization without modifying the core source code of ODACE. The use of scripts also provides a standardized default testing process, enabling users to easily modify, add, or remove test steps based on specific certification requirements or device conditions.

[1] For more information: https://developer.android.com/tools/adb

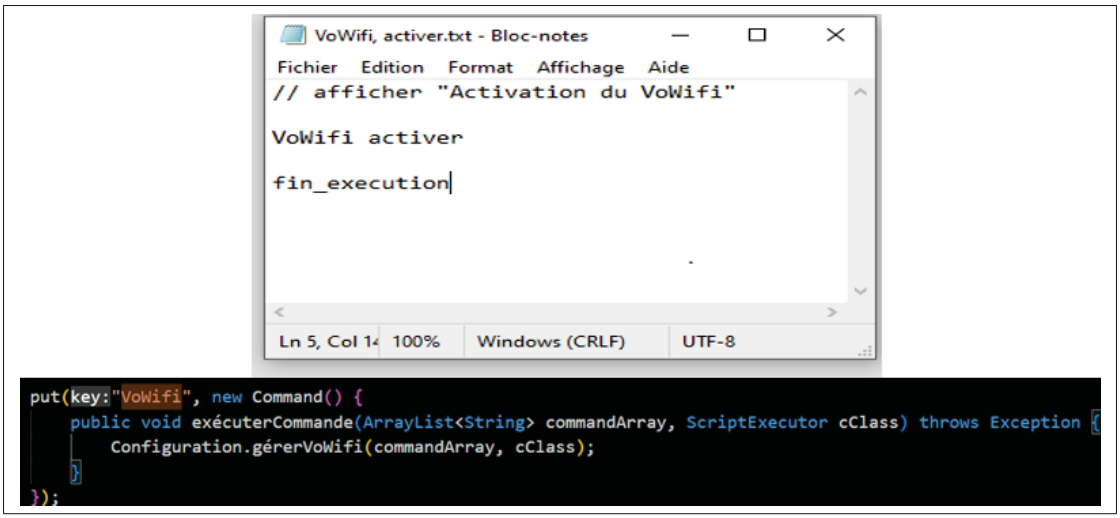

Figure 3.3 Example Implementations of ASCII Text Handling in ODACE code

4. Appium-based Testing Tool: Appium is the core testing tool within ODACE, chosen for its superior features compared to alternative tools, as we explained previously. Appium supports a wide range of programming languages, excels in black-box testing, and offers extensive compatibility with different devices, as highlighted by da Silva & de Souza Santos (2023).

   Appium has a client-server architecture and uses the WebDriver protocol to enable automated interaction with mobile applications Wang & Wu (2019). The Appium client can be implemented in different programming languages and includes test scripts and libraries. These clients translate the developer's test commands into JSON Wire Protocol or W3C WebDriver protocol messages, which are then sent to the Appium server.

   The Appium server receives these commands, establishes a session with the target mobile device, and executes the actions by simulating user interactions such as taps, scrolls, and input without requiring any manual intervention.

   To configure Appium, we specify a port number (default: 4723) to establish communication between the Appium client and server. Additionally, a set of Desired Capabilities must be provided. These capabilities define parameters such as the platform name, device name,

and application path. They are essential for initializing a session and ensuring the server connects to the correct mobile device. [2]

This tool allows ODACE to execute test scripts that simulate user interactions, validating both the functionality and performance of the DUT.

5. Bots Automation Application: The design employs bot devices to perform some tests that require interaction with the DUT. At each Bot, we develop a flow of the "Automate" Android application (developed by LlamaLab) that controls these devices, which uses a customized automation program (flowchart) for specific tasks and functions such as answering incoming calls or initiating calls to particular numbers without human interaction. In our solution, we use 19 flows to do different tasks, and these flows are used for both testing and setting up the DUTs.

Figure 3.4 explains the software architecture and the connection between them.

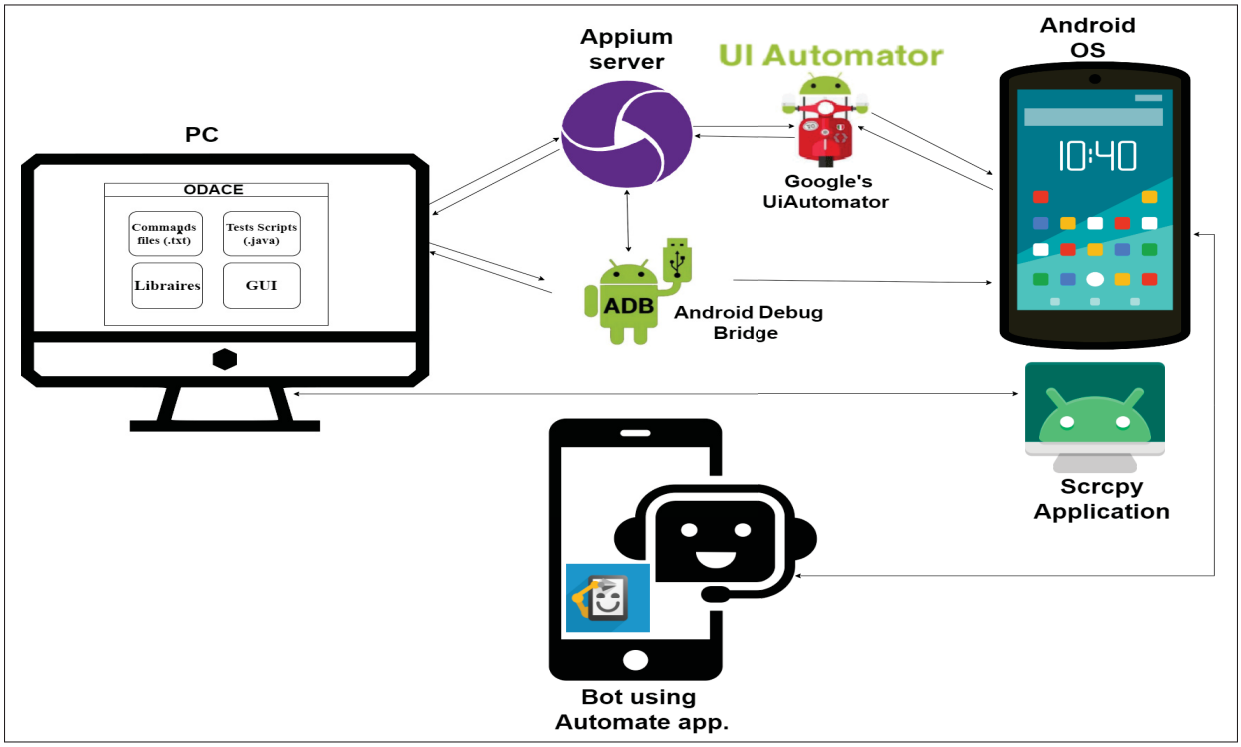

Figure 3.4 ODACE Software Architecture

[2] More details available at https://appium.io/docs/en/2.18/intro/appium
https://www.browserstack.com/guide/appium-tutorial-for-testing

3.6 Conclusion

This chapter presented the main methodology and architectural design of the proposed solution, ODACE. It described both the software and hardware components, including the script files, bot solution, and other hardware and software elements that enable test automation.

Also, we provided a detailed explanation of ODACE structure and automation logic. This contribution showed how the proposed framework reduces the need for manual intervention in the certification workflow.

Furthermore, this chapter answered the first research question: "How feasible is it to automate mobile device certification tests to reduce manual effort and time?" The implementation of the ODACE solution showed that automation is not only feasible but has been successfully achieved, significantly reducing manual effort and improving overall efficiency.

The following chapter will describe the extended version of ODACE-RMS (ODACE Remote Multi-Session), which introduces new features such as remote device controlling and multi-device parallel testing. It will also illustrate the system workflow, user interface, and provide a case study.

CHAPTER 4

MULTI-SESSION DESIGN AND IMPLEMENTATION

4.1 Introduction

While the initial version of ODACE is bringing automation to certification testing, it is limited to automation of single-device testing, which restricts engineer productivity. This limitation motivates us to a transformative approach to mobile Android device certification, leveraging automation to enhance efficiency and scalability in certification testing.

For these reasons, we aim to extend ODACE to address our second research objective: enabling parallel automated test execution. This new feature allows certification engineers to execute different test scenarios on multiple devices simultaneously.

Before extending the solution, we considered the concept of a parallel pipeline, which involves breaking tasks into smaller steps that run in parallel, as described in Lamy-Poirier (2023). We applied the same concept to automated testing by allowing multiple devices to run simultaneously. Each session executes different tests independently, which aim to increase efficiency and reduce overall certification time.

4.2 Multi-Session Hardware Architecture

In the proposed ODACE with multi-session feature (ODACE-MS), we added an additional hardware component to the original architecture.

- Server: The server in ODACE-MS is implemented as a virtual machine (VM) and is responsible for hosting the central database. This database stores necessary data such as user profiles, DUT information, test execution logs, and session history.

  Although the server plays a critical support role by ensuring data persistence and consistency across multiple test sessions. This enables ODACE-MS to efficiently handle multiple sessions, both locally when running on a single PC and remotely across multiple users.

This lightweight VM-based setup reduces infrastructure overhead while maintaining the centralized data storage required for collaborative testing and future analysis.

There is no special requirement for the PC to implement ODACE-MS. We designed the architecture to be as simple and flexible as possible. The solution works on any regular PC without needing a specific operating system, high memory, or powerful CPU.

The goal was to avoid centralized setups and instead allow each CE to run their own sessions locally. This makes it easier to use ODACE-MS in different environments, supports multi-session testing on standard machines.

Figure 4.1 shows the hardware architecture components.

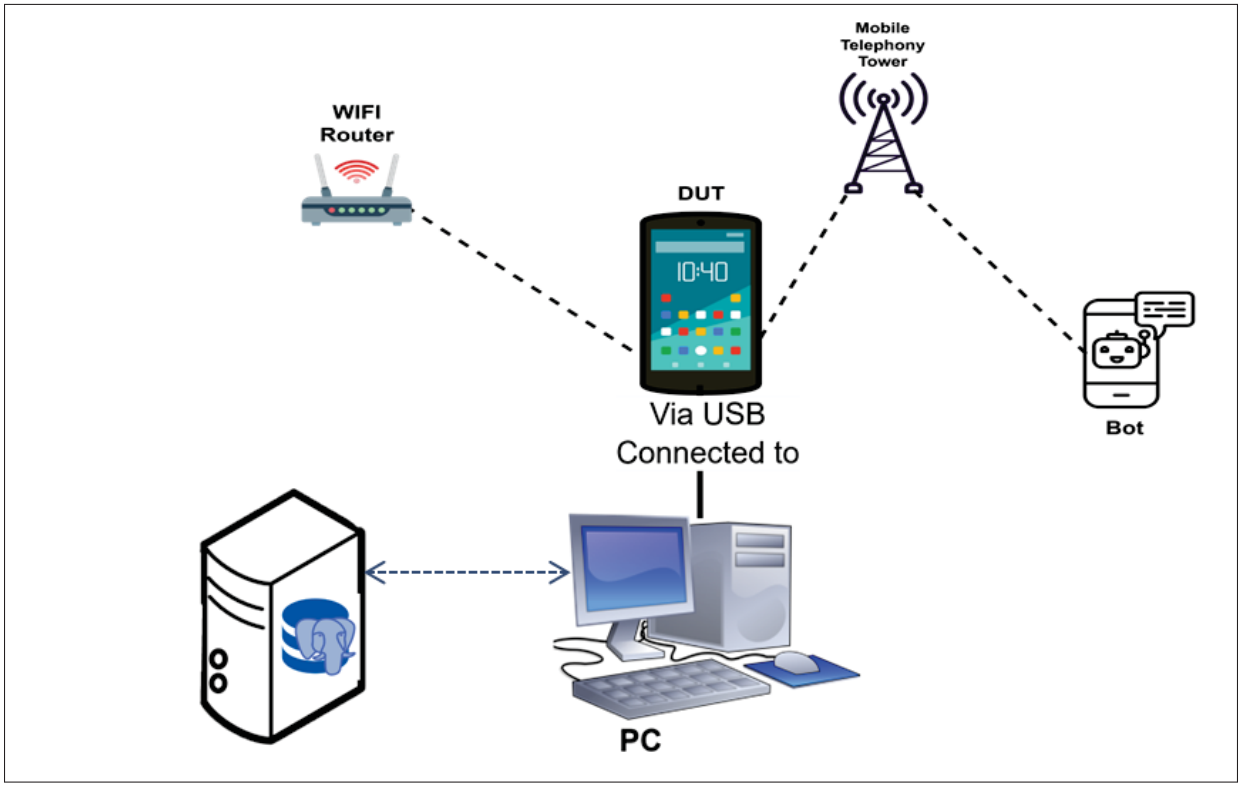

Figure 4.1 Multi-session Hardware Architecture

4.3 Multi-Session Software Architecture

1. ODACE-MS Frontend and Backend: It includes the following parts:
    a. Frontend: The frontend is developed using jQuery and HTMX, enabling dynamic and responsive HTML interfaces that enhance the user experience. These lightweight technologies simplify the creation of interactive web components without requiring full-page reloads, making the interface efficient and easy to maintain.

       The main advantage of this approach is that it allows CEs to open a separate browser tab for each testing session. Each tab operates independently to manage a different device and test scenario in parallel.

       One of the main challenges was migrating the original JavaFX interface to a web-based HTML interface, while preserving the familiar layout and functionality of the classic ODACE platform. This step was important to ensure a smooth transition for CEs, to make the extended solution more acceptable and maintain continuity in their workflow. the web interface is launched directly when the user starts the platform, without no additional setup. The Java classes manage the interface by map frontend components to backend services. Also the backend pass the CSS and javaScript to the interface components to handle the styling and interaction behavior. This integration between frontend and backend allows for a seamless and responsive user experience.
    b. Backend: The backend is implemented using the Spring Boot Java framework, offering a scalable and robust architecture. This design enables the management of multiple devices and processes simultaneously. We chose Spring Boot because the core of the original ODACE was already built in Java. This allowed us to reuse existing code and components instead of rebuilding everything from scratch. Using Spring Boot Java is important for us to have both Java features to manage ADB communication safely and reliably, besides having the advantage of Spring Boot, which provides powerful features such as dependency injection and RESTful API support.

       Although we updated most of the project files to align with the new architecture and the new requirements, we kept the original automation workflow. This ensured continuity

for CEs, who can continue using the system without retraining, as the testing logic and interaction remain familiar.

We integrated several key Spring Boot features. We used the embedded Tomcat server to able ODACE-MS to run as a standalone application, which is eliminating the need for external server deployment and distribute the resources between the users. We used the Spring MVC framework to handle HTTP requests and responses through RESTful APIs, which manage test execution, session tracking, and communication with connected devices. Moreover, We implemented Spring Scheduler to manage scheduled tasks, such as checking device statuses and performing automated cleanup of inactive sessions.

We also applied dependency injection and component scanning to modularize services such as ADB controllers, Appium managers, and session monitors. This design allowed us to easily maintain or extend components without affecting the entire system. Furthermore, We used REST controllers to exchange data between the backend and other components in JSON format.

2. Database Integration: ODACE-MS integrated PostgreSQL for managing and storing data related to bots, devices, test results, and user profiles. Integrated with the Spring Boot backend by using JPA (Java Persistence API)to manage database operations. This approach not only reduces the need to write raw SQL queries but also improves code readability and maintainability. JPA allows developers to map Java objects directly to database tables (ORM). Additionally, JPA provides database independence, making it easier to switch between database systems without major code changes.

   This database mainly ensures efficient management of session data and supports multi-session operations.

3. Some changes were made to adopt Appium with ODACE-MS requirement. In the classic ODACE, Appium was limited to a single session per user. So, it used the default port and a single set of desired capabilities to build and start the session. However, with ODACE-MS, we needed to manage multiple sessions in parallel.

To test this, we built a mini proof-of-concept application that could run tests on multiple devices at the same time. While we were able to execute tests on different devices in parallel testing, we faced some limitations, such as the same test being executed on all devices, which did not provide the level of independence we needed between sessions. Additionally, Appium would sometimes hang due to incorrect configurations or port conflicts.

To solve this, we redesigned session management in ODACE-MS. For every new session, the platform creates a separate session with its own Appium server port and desired capabilities. We implemented a Java method to automatically assign a free and available port for each new session and start a new Appium server instance using a server port that is assigned to the ODACE-MS server by another Java method and without user intervention. This approach enables the solution to manage multiple independent sessions without conflict, and it also makes the setup process easier for users who may not have deep technical expertise.

Figure 4.2 explains the software architecture of ODACE-MS.

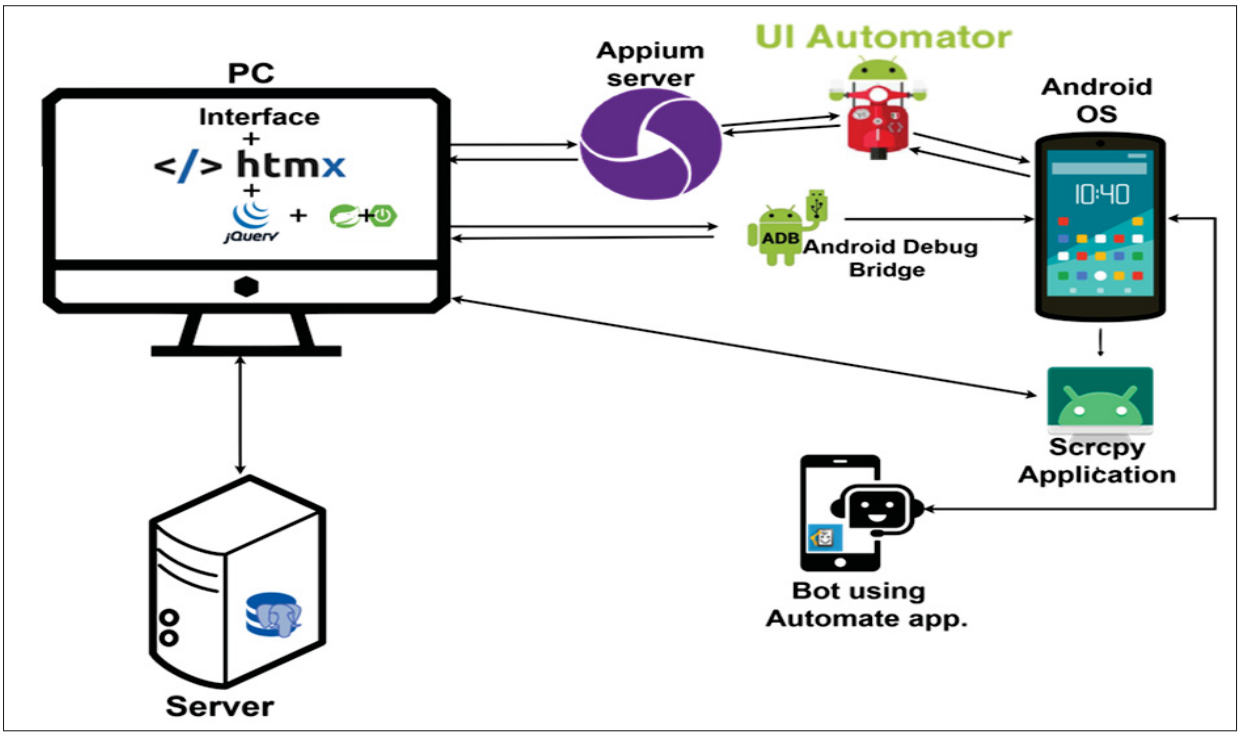

Figure 4.2 Multi-session Software Architecture

4.4 Multi-Session User Interface

The user interface (UI) is designed to reflect the certification context and be user-friendly for certification engineers without the need for special training or prior programming expertise.

In ODACE-MS UI, the main sections and layout have been preserved to ensure an easy transition for certification engineers who are used to using the previous ODACE interface. This approach minimizes the need for additional training and ensures that users can easily adopt the new UI.

We redesigned using Spring Boot's capabilities, we transitioned from the previous FXML interface 4.3 to a web-based HTML interface accessible through a browser. This redesign enhances the user experience by making the system more user-friendly and flexible. The web-based interface allows users to open multiple tabs simultaneously when they run multiple ODACE-MS sessions.

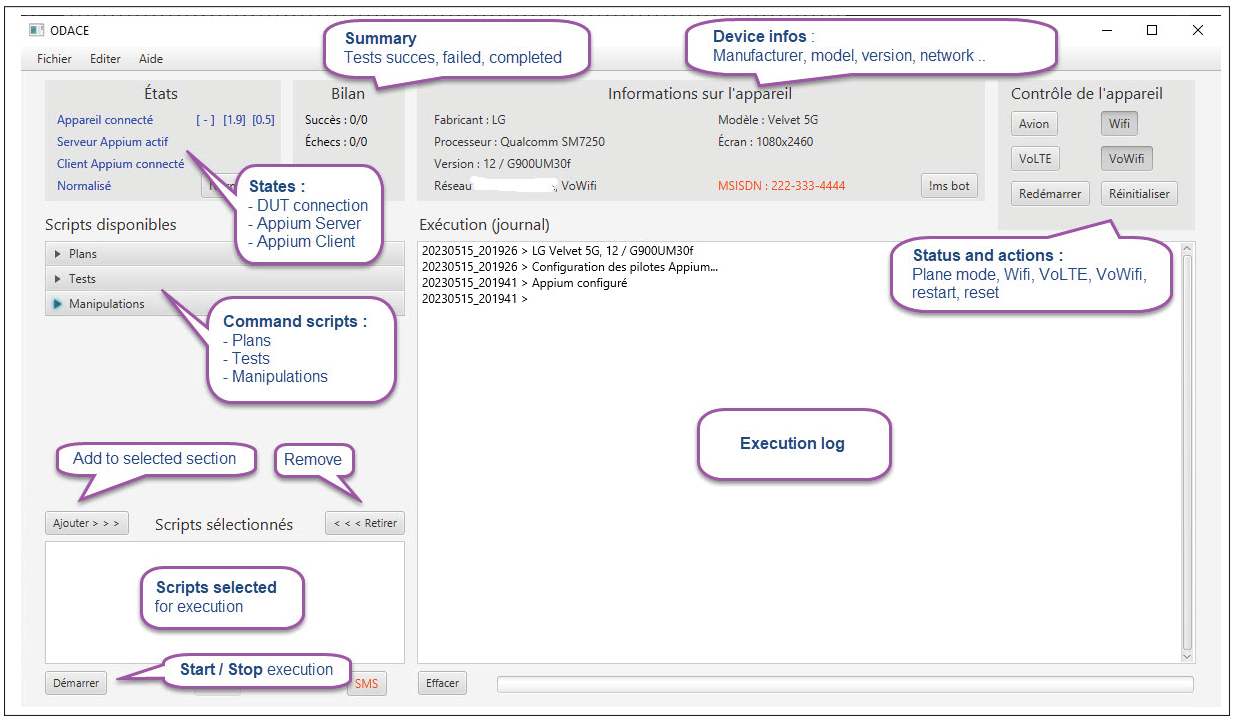

Figure 4.3 ODACE User Interface

As shown in Figure 4.4, the redesigned UI is organized into several key top and middle parts. The top part includes:

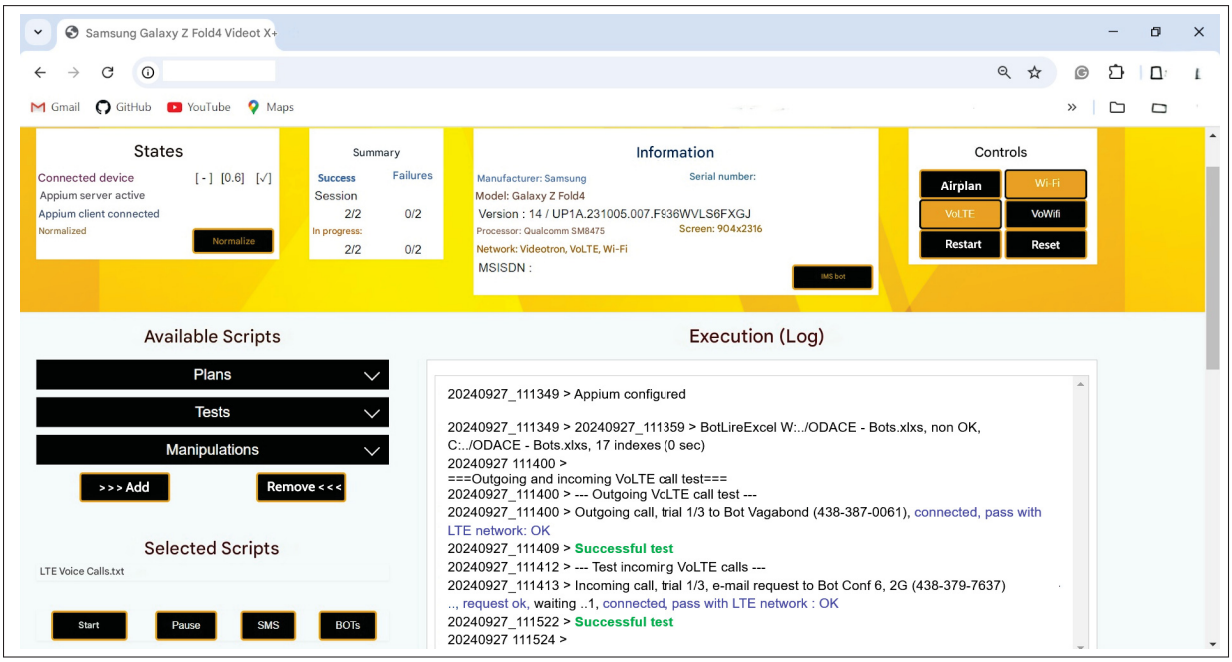

Figure 4.4 ODACE-MS User Interface (translated from French)

- **Tools and Device Status:** Located in the top left corner, this section provides real-time updates on critical system components, including the ADB connection, the status of the Appium server (PC), the Appium client (device), and the device normalization status (e.g., language and other settings required for automation).
- **Test Results Counters (Summary):** The second part of the top row displays counters summarizing the results of the executed tests, which provide users with a quick overview of progress.
- **Device Information:** The third section on the top row presents detailed information about the connected device, such as brand, model, software version, and network connection status.
- **Status and Actions:** The rightmost section of the top row includes buttons for executing basic device functions, enabling quick interaction and control.

The central section includes:

- **Script Management:** The middle row includes an accordion menu, which gives access to the different script categories, including Plans, Tests, and Manipulations. Users can select scripts for execution. The selected scripts are displayed in the Scripts Selected section below.
- **Execution Log:** Occupying the rest part of the interface, this section displays real-time progress and detailed logs of ongoing tests.

This structured layout improves the overall efficiency of device certification tasks and ensures that it is easy to use for both novice and experienced users.

However, we face challenges with multi-session testing. Specifically, CPU usage becomes 100% when running two or more sessions simultaneously, which causes significant delays. We addressed this challenge by optimizing the code by minimizing the number of ADB shell commands opened and closed during the tests, We now open a single ADB shell session then execute different commands within it and save the results in parameters for easy access and reuse and make sure to close the session once close the application session. This approach eliminates the need to repeatedly re-execute ADB shell commands, which previously consumed significant resources. Moreover, when we were digging to see which uses the most CPU and memory, we discovered that "adb devices" commands use a lot of resources, especially since it had to be executed every three seconds to verify device connectivity and ensure safe test execution. To overcome this, we replaced it with the more lightweight adb track-devices command, which continuously monitors device state changes and provides real-time notifications when a device is connected, disconnected, or authorized. This optimization significantly reduced CPU usage and resolved the delays in multi-session scenarios.

4.5 Conclusion

To meet the evolving needs of the certification engineers and improve the usability of the platform, we proposed adding a new feature to allow the user to test up to 8 devices in parallel within a user-friendly interface and environment, which significantly reduces testing time and increases efficiency. These addressed our second research question: RQ2: Can parallel automated testing be effectively applied to speed up the certification process? The answer is Yes, by enabling multiple sessions to run concurrently on different devices, ODACE-MS reduces overall certification time. The parallel execution architecture ensures efficient use of resources and avoids delays caused by sequential testing.

This Chapter presents one of the new capabilities of ODACE-RMS. In the next chapter, we will discuss the second feature design and highlight the platform workflow.

CHAPTER 5

REMOTE AND MULTI-SESSION CAPABILITIES

5.1 Introduction

The need for remote features has become more important, especially with the growing trend of hybrid work in companies. This challenge was clear with the original ODACE platform, which cannot be done remotely and often causes delays until the engineers can physically access the devices, especially the devices that can't leave the operator's premises for confidentiality. This need encouraged us to find a solution that allows certification engineers to interact with devices remotely, improving flexibility and efficiency.

In this chapter, we aim to achieve the third research objective: implementing remote automated testing. To implement this remote feature, we explored technologies that can share USB connections over the network. After evaluating different options, we decided to use USB-over-IP technology, which allows remote access to physical USB devices as if they were connected locally. We propose the final platform of this work is called ODACE-RMS obtained when we add this new remote feature to ODACE-MS.

5.2 Remote USB-over-IP Hardware Architecture

To support the Remote feature in ODACE-RMS, we integrated a new component into the multi-session architecture.

1. USB-Over-IP Server: This component is the core for the remote testing feature in our solution. It enables devices connected to physical USB ports to be accessed over the network as if locally connected.

   Before adopting the USB over IP solution, we explored other options such as using a physical server that could be accessed remotely via VPN to run the tests. However, this approach had several limitations. First, it was limited by the number of available ports, and being a centralized solution, it required a server with high CPU and memory capacity to

handle multiple tests and users. Additionally, if one CE was already using the server to start running tests, others had to wait till the CE finished setting up the test and started it, which caused delays and reduced overall productivity.

These challenges made the option complex and inefficient to use. What we needed instead was a system that could distribute resources across different PCs, allowing multiple users to test independently. This is why USB over IP became the best solution for us. By connecting a device to a port on the USB hub and choosing it from the user PC, ODACE-RMS, which is installed locally on the CE's machine, detects the device as if it were physically connected. This setup enhances user independence and flexibility, even when multiple users test remotely using the same hub.

In our ODACE-RMS solution, we chose the DIGI AnywhereUSB Plus server[1], which supports USB-over-IP and is reliable for multi-device environments. This server acts as a bridge between the DUTs and remote engineers. We connect the devices to the DIGI server using USB cables. Then, from the engineer's PC, the corresponding ports can be mapped through the DIGI software.

Once connected, ODACE-RMS interacts with the devices as if they are physically attached to the engineer's PC. For example, running the ‘adb devices‘ command will list both local and remotely connected Android devices, allowing seamless integration into the testing workflow. The number of users who can test at the same time depends on the number of server ports. Each port can also support multiple devices using USB hubs, which gives the platform more scalability and allows parallel testing across many devices and users. Figure 5.1 shows the design of the remote feature components and the connection between them.

Figure 5.2 shows the hardware architecture components.

[1] More details available at https://www.digi.com

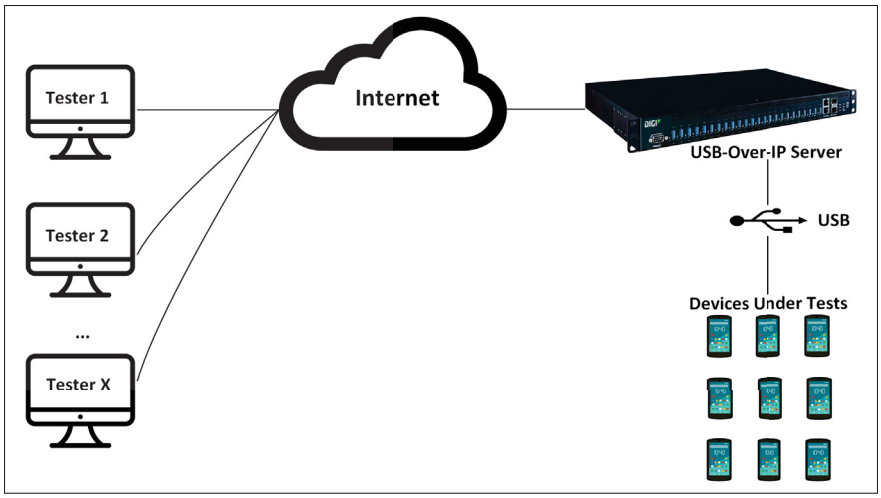

Figure 5.1 ODACE-RMS Remote Components Management Design

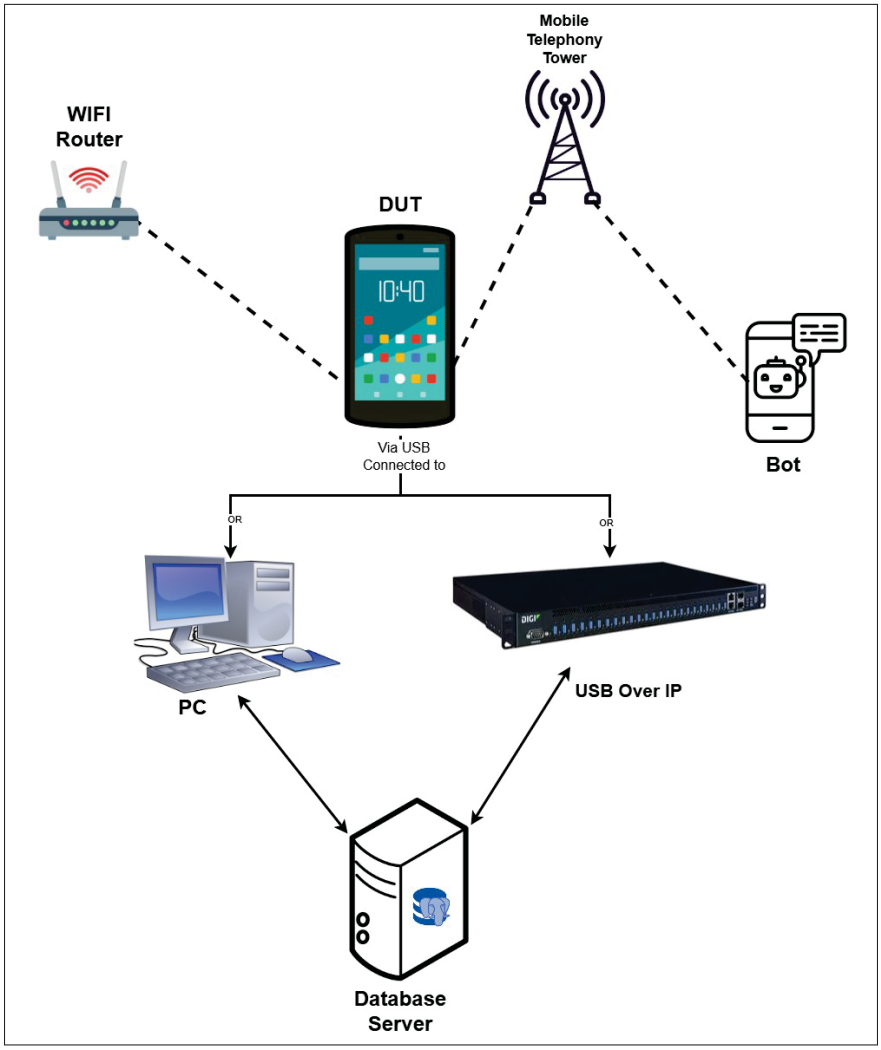

Figure 5.2 ODACE and ODACE-RMS Hardware Architecture

5.3 Remote USB-over-IP Software architecture

1. Digi Remote Manager: ODACE-RMS use Digi Remote Manager to ensure remote device management and connectivity. This component provides functionalities such as streamlined deployment, asset management, automated updates, and real-time alerts, Digi International (2023). It connects with the DIGI hub that provides USB over IP technology, facilitating remote device management and connectivity.

   For a PC to control DUT via USB, Android security mandates that USB access permission is allowed from the device via an on-screen pop-up, as shown in Figure 5.3.

Figure 5.3 USB Debugging Notification

Remote access to the device is blocked until that is granted, which was a big challenge for us when implementing the remote feature. We initially tried using TeamViewer's Organization solution to overcome this, but it has several complications. First, the solution is not free, which limited its accessibility. Second, authorization still had to be done manually the first time a device was connected. On top of that, we had to install the application on each DUT, which added extra setup steps and made the overall solution more complex.

Then we introduced another solution to solve that challenge and complete that step without manual intervention. An Automate application program which automatically clicks “Allow” upon the appearance of the “Allow USB debugging?” screen prompt. The automated workflow is shown in Figure 5.4.

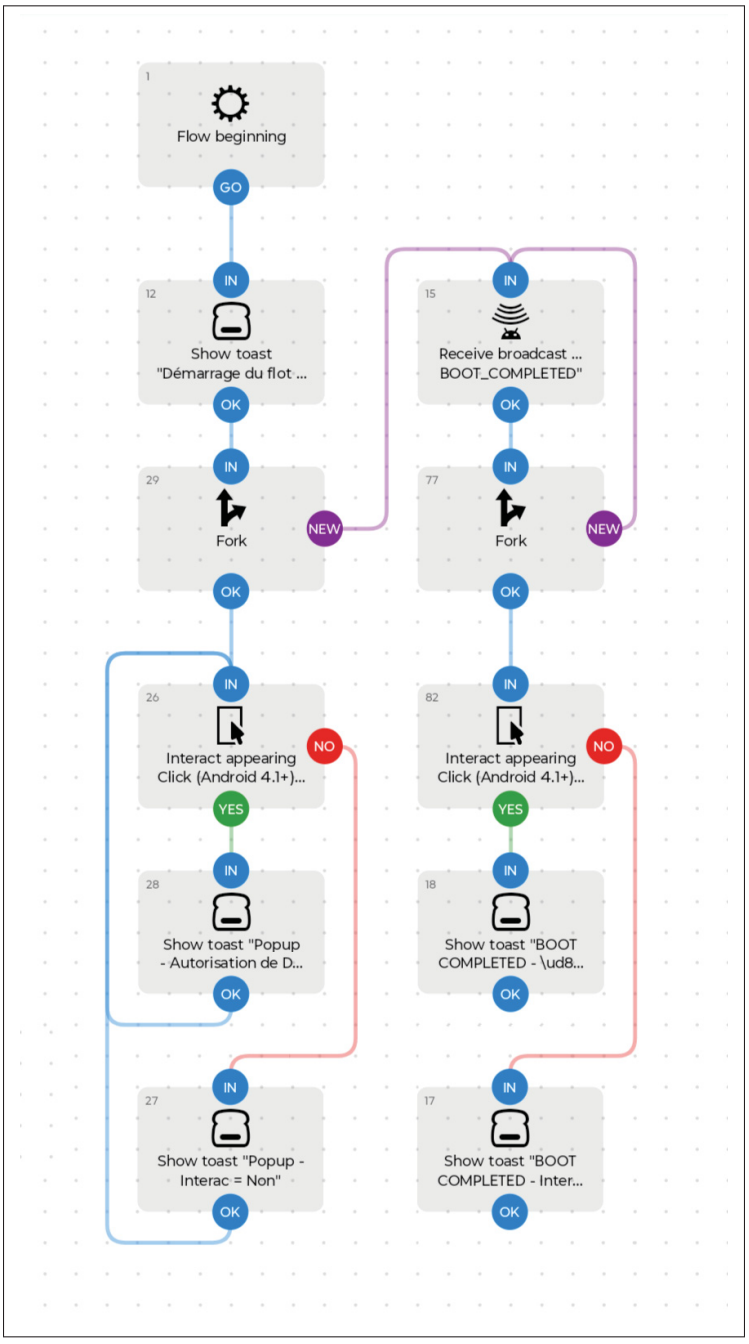

Figure 5.4 ODACE USB connection workflow

Figure 5.5 explains the software architecture of ODACE-RMS.

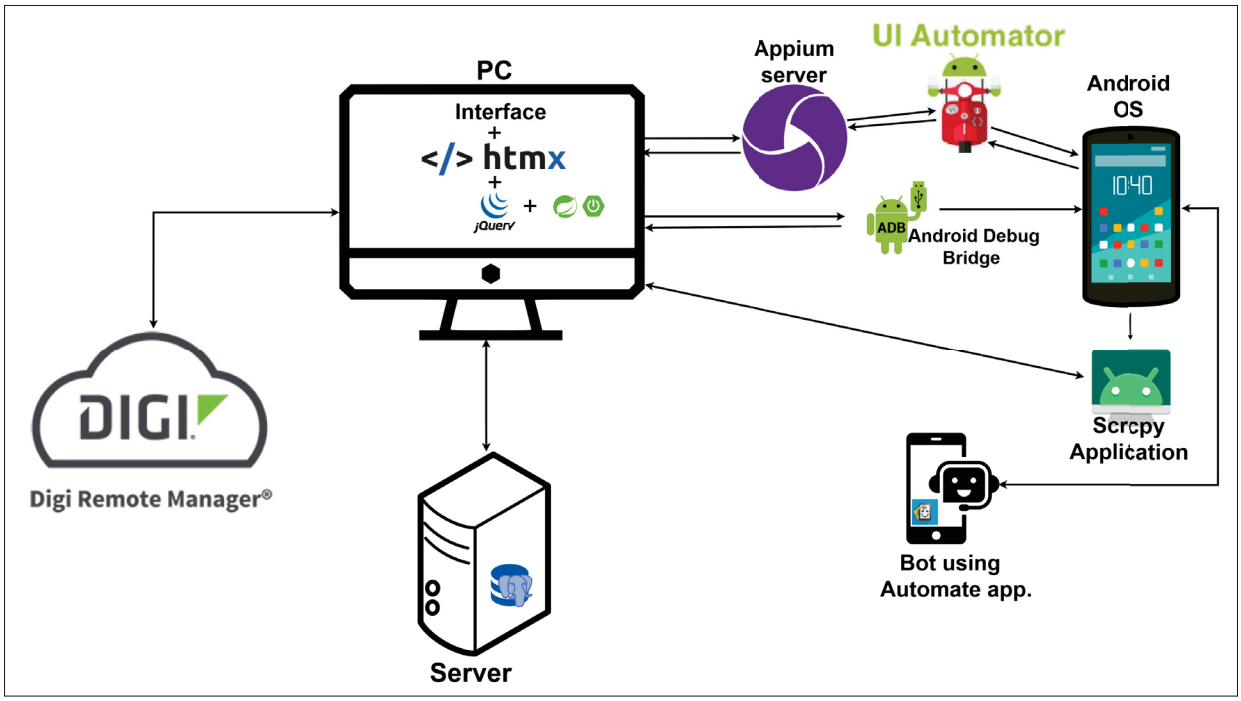

Figure 5.5 ODACE-RMS Software Architecture

5.4 ODACE-RMS Workflow

The general automated certification testing process unfolds as follows:

1. **Preparation of the DUT:** Developer Mode is activated, allowing DUT control over the USB connection to configure the device according to specific test requirements such as Airplane mode, language settings, and activation or deactivation of certain features.
2. **Preparation of the certification engineer computers:** Install the Appium tool and Screen Copy (ScrCpy) application. Appium allows the reading and manipulation of DUT screen elements over a USB connection. ScrCpy is a screen mirroring application showing the DUT screen on the certification engineer's computer monitor.
3. **USB Connection:** A USB cable connects the DUT to the PC.

4. **Running the automation platform:** the solution includes the scripts of Java, a graphical user interface (GUI), and all necessary configuration and command files, which will be elaborated upon in subsequent sections.
5. **Test Selection:** Within the interface, certification engineers can choose from various test plans and technologies to initiate testing as per their requirements.
6. **Result Analysis:** After testing, the results are analyzed and copied to a repository. If any software issues are identified, the user collects logs and creates a bug report, or if necessary, specific functions are manually rechecked to ensure that.

While the previous flow detailed the ODACE workflow in a local setup, we now turn to highlight the ODACE-RMS workflow. The following outlines its operation:

1. **Device Connection:** The ODACE-RMS testing process begins when a user connects a DUT to the PC and starts the application. For the local version, the DUT is connected directly to the PC via USB. In the remote scenario, the DUT is already connected to a USB-over-IP server in the lab.
2. **Device Detection and Info Collection:** ODACE-RMS monitors the PC's USB ports and uses ADB commands to detect connected devices automatically. Once detected, the platform collects the main details (model, OS version, software version, etc.) and then stores them in a database.
3. **Device Selection and Reservation:** If only one device is detected, ODACE-RMS automatically assigns it to the active UI tab. If multiple devices are connected, the user manually selects one. The selected device is then reserved in the database and disabled in other tabs to avoid conflicts.
4. **Appium Configuration:** To minimize delays, ODACE-RMS collects necessary information from shared files, such as bot data, to prepare for the tests efficiently.
5. **Test Preparation:**
    a. To save time, ODACE-RMS retrieves supporting information (e.g., bot data, Clippy Phone) from shared files before starting tests.

b. Whether using USB or USB-over-IP, ODACE-RMS behaves the same. All data collection and actions are performed using ADB, ensuring consistency in both local and remote setups.

c. When started, ODACE-RMS displays all available devices (local and remote). If a user selects a remote DUT, the platform reserves the device until the user releases it. By default, the system supports up to eight parallel devices per user. This limit can be changed by modifying the `ADB_LOCAL_TRANSPORT_MAX_PORT`. However, the default limit is practical for our solution since users typically test three to four devices simultaneously. For remote testing, the maximum number of users depends on the port numbers of the hub.

6. **Test Plan Execution:** ODACE-RMS offers users various test execution options. Users can select predefined test Plan scripts, which consist of a set of tests and manipulations corresponding to specific test types (e.g., VoLTE, VoWi-Fi, 5G) and levels (e.g., Basic, High-level, Regression, or Complete) as detailed in Section 2.1. For example, users can choose plans to test LTE Basic, VoLTE Complete, or Wi-Fi Calling Basic. Alternatively, users can execute specific actions by selecting individual Test and Manipulation scripts.
7. **Real-Time Monitoring:** Every three seconds, ODACE-RMS checks for new devices, disconnections, or status changes. It also verifies radio tech, call status, mobile data, and Wi-Fi. Additionally, the proposed solution verifies the status of the Appium server, ensuring that it remains active and that the DUT's Appium client is connected during the test execution.
8. **Script Execution and Reporting:** Upon script selection, the user clicks "Execute." ODACE-RMS performs required tasks by using the appropriate methods. Upon completion, it exports a report summarizing the outcome. The system then returns to wait for new commands and tests to execute.

Figure 5.6 presents the process of the platform.

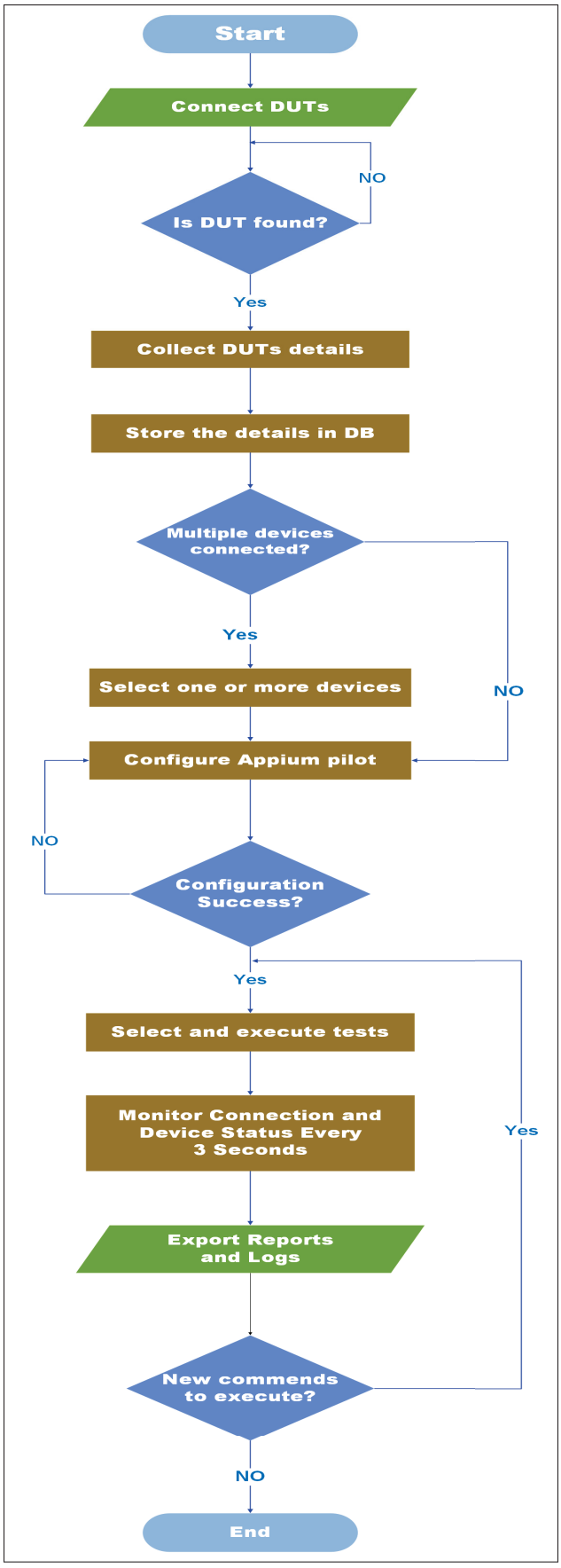

Figure 5.6 ODACE-RMS Workflow

5.5 Case Study - Mobile Originating Call

One of the most important functions that needs to be verified is the Mobile Originating (MO) call test. This test verifies a device's ability to initiate and complete outgoing calls. This fundamental test ensures proper network registration, call setup, voice transmission, and call termination functionality from the device under test.

To validate the MO call function on a new software version, the CE performs the following steps:

1. **Preparation of DUT:** Update the DUT with the new software version, insert an operator SIM card, activate USB debugging, and connect the device to the PC.
2. **Launching ODACE-RMS:** Start ODACE-RMS on the PC to initiate the Appium server and verify the presence of the DUT.
3. **Device Profile and Status Collection:** ODACE-RMS continually collects and displays the DUT's profile and status on its user interface, logging this information simultaneously.
4. **Activation of Appium Client:** ODACE-RMS prepares the DUT for testing by activating the Appium client.
5. **Test File Selection and Execution:** The user selects the necessary test files, adds them to the script list and initiates the testing process.
6. **Call Test Procedure:** ODACE-RMS conducts the call test by randomly selecting a BOT (Breakout Tester) and controlling the DUT to initiate a call. It monitors the time and waits for the call to be established.
7. **Handling Failed Call Attempts:** If a call setup fails due to BOT unavailability or malfunction, ODACE-RMS tries with another BOT. Three unsuccessful attempts with different BOTs resulted in a test failure.
8. **Successful Call Handling:** Upon successful call connection, ODACE-RMS checks the Voice technology on the DUT and then terminates the call.
9. **Test Outcome Determination:** The test is declared a PASS if the call setup, voice technology verification, and call termination are successful; otherwise, it is marked as FAIL.

This process ensures that the call function is thoroughly tested and malfunctions are detected for investigation by engineers who will create anomaly documents and report to the manufacturer when necessary. The test cases in the remote testing feature remain the same as those in local testing. The only difference between local and remote testing is the method of connection. Local devices are connected directly via USB, while remote devices are connected using USB-over-IP technology from the DIGI hub. Despite this difference in connection methods, the execution process is exactly the same for both scenarios.

Figure 5.7 describes the testing process.

5.6 Conclusion

To meet the evolving needs of the certification engineers and improve the usability of the platform, we proposed adding new feature which provide the user more flexibility and practicality by use USB-over-IP technology to control the devices remotely, which make the engineers able to work even if he is not able to have the device physically for the different reasons. Additionally, the user is able to test up to 8 devices in parallel within a user-friendly interface and environment, which significantly reduces testing time and increases efficiency. These addressed our remaining research question: RQ3: Can the proposed framework improve testing behaviour and offer more flexibility for certification engineers? Yes. ODACE-RMS introduces a remote multi-session feature to allow engineers to test devices from different locations.

These improvements will be further detailed in the next chapter, which presents a comparative analysis of ODACE and ODACE-RMS results, supported by figures and tables, and discusses the resulting outcomes.

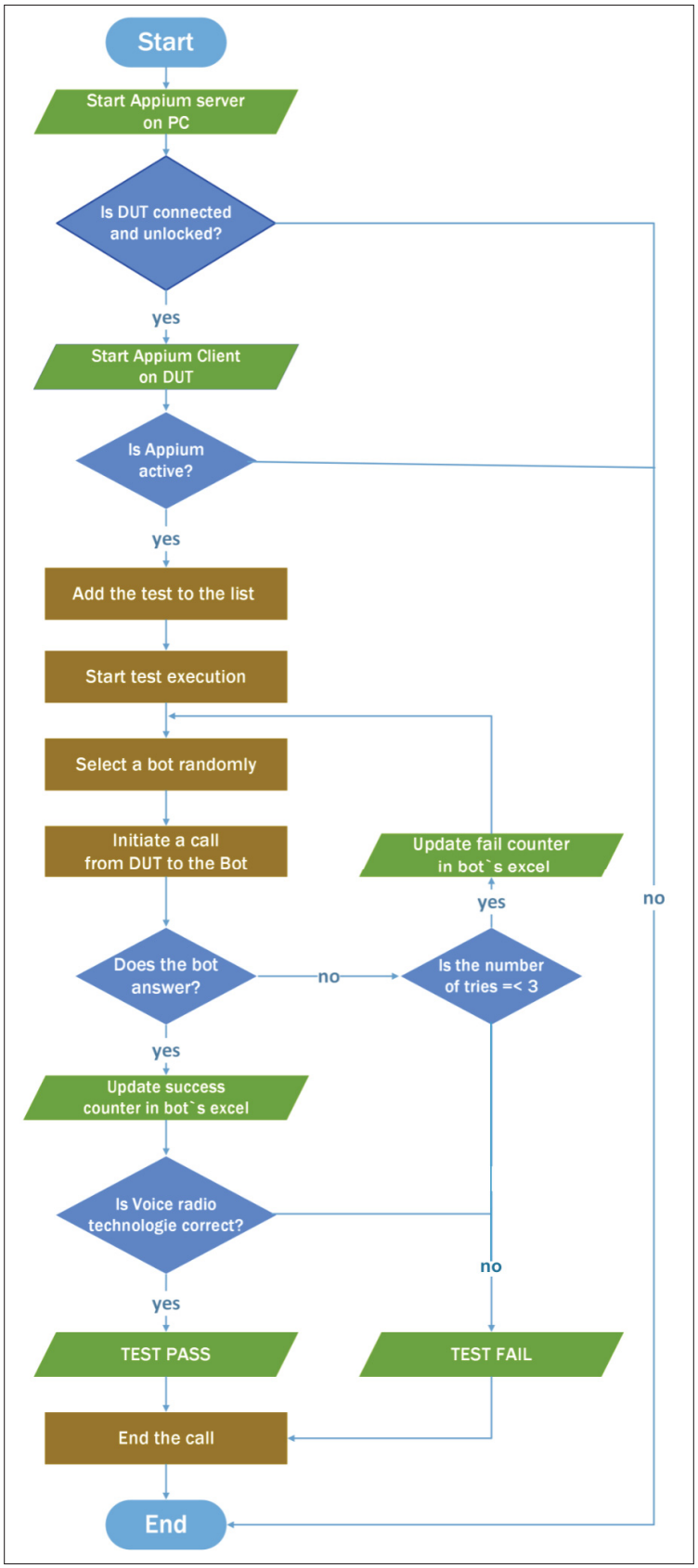

Figure 5.7 Case Study: Mobile Originating call test flow chart - ODACE-RMS

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Introduction

The previous chapters covered the proposed solution and highlighted the importance of automating the certification process. Such automation has the potential to revolutionize the workflow by saving engineers time and making the platform more accessible to all users, including interns and new employees, allowing them to integrate into the team more quickly. To evaluate the real benefits and effectiveness of the platform, this chapter presents and discusses the results collected in collaboration with Vidéotron, based on data from the certification of wireless devices department spanning 2022 to 2025. The discussion will focus on the impact of automation, as well as the effectiveness of the remote and multi-session features.

6.2 Automation Results

To assess the proposed automation solution in the certification process, we evaluate its performance through a series of benchmarks, comparing its execution times, accuracy, and reliability against manual testing processes in previous certification scenarios. This evaluation focuses on the automated solution through the ODACE platform. We, therefore, tracked the number of tests executed by the certification of wireless devices team in Vidéotron during each quarter from Q1 2022, before ODACE introduction, until Q3 2023. Automated tests were gradually introduced from Q2-2022 to Q1-2023. During the total period, 39135 tests were performed, of which 13406 were with ODACE. On average, 5528 tests are performed every quarter, of which, since the complete introduction of ODACE, 2815 are automated. Figure 6.1 shows the number of tests performed quarterly; during the last quarters of 2023, ODACE succeeded in automating 52% of all tests executed.

To evaluate ODACE, we compare the test execution durations of manual testing with automated testing. The time required for preparation steps has been excluded since it is always the same.

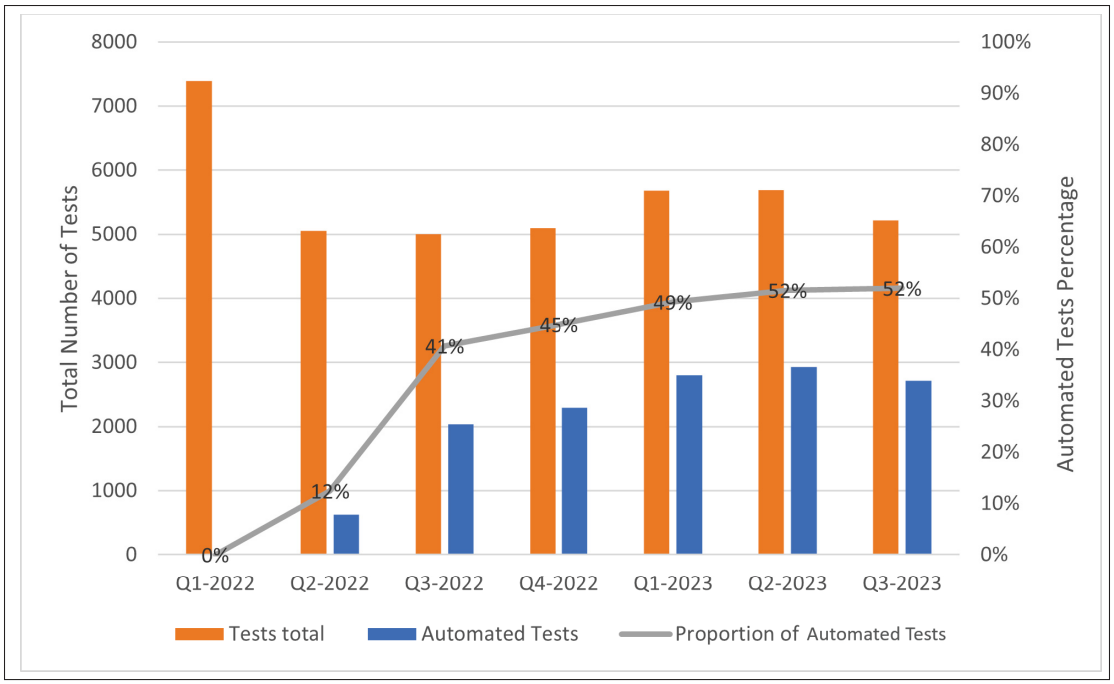

Figure 6.1 Quarterly Numbers of Certification Tests at Vidéotron

Table 6.1 shows the manual and automated time of ODACE quarterly. We clearly show that the time required for automated tests is less than for manual testing execution by 19% to 23%. The average test execution time is 16.7h/month for each certification engineer and is reduced to 12.9h/month when the same tests are done by ODACE. This reflects the time-saving by the tool, even if we ignore the engagement time (defined as the time when the CE is engaged in the testing process) and assume that the user should monitor the devices, whether manually or through automation.

Table 6.1 Manual Time vs. Automated Time

Quarter/year	Manual Time (h)	Automated Time (h)
Q1-2023	56.1	45.2
Q2-2023	53.9	41.5
Q3-2023	48.2	36.9
Sum	158.2	123.7
Average	52.7	41.2

Figure 6.2 shows the times elapsed to execute predefined 'automatable' tests when performed manually or using ODACE during the period covered.

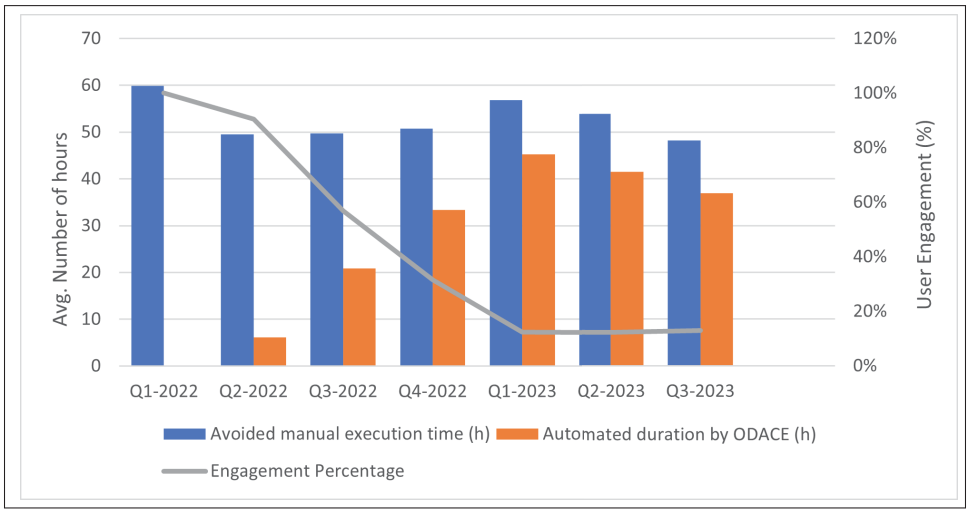

Figure 6.2 Quarterly Time of Certification Tests with User Engagement

Another evaluation presented by Figure 6.2 compares the user engagement time, which is one of the most important aspects of evaluating the solution. As discussed previously in Table 6.1, ODACE cuts test execution time by 21.8%, but it also allows the certification engineer to do other tasks while ODACE executes the tests. In fact, the certification engineer engagement time for manual testing is 100%, while automated execution cuts that to 11%, meaning a reduction of 89% in terms of engagement time.

6.3 Remote and Multi-Session Results

In this part of the evaluation, we focus on determining how the remote multi-session feature impacts time efficiency, operational flexibility, and CPU usage among various browsers.

To evaluate ODACE-RMS's performance, we designed different scenarios to highlight improvements in process times, remote capabilities, and overall efficacy. Figure 6.3 presents these scenarios, each based on more than 50 test cases. We then collected the average execution time in seconds for comparison. All tests were conducted in the same environment while varying the number of devices to test the multi-session capability. The results for the classic ODACE curve

were obtained by multiplying the execution time of a single device by the number of devices, as the classic version does not support parallel testing (testing sequentially). For example, the test locally takes approximately 81.7 sec. to complete. Without multi-session support, testing four devices sequentially would require around 326.8 seconds in total. In contrast, with ODACE-RMS's multi-session feature, the same tests can be executed in parallel, reducing the total execution time to 86–91.7 seconds. This is a significant time-saving. To further evaluate the remote feature, we test ODACE-RMS under different connectivity conditions. First, direct the USB connection to the PC. In the second case, remote access through a DIGI hub on the same network (geographically close). Finally, remote access from outside Montréal while the devices remained in the lab (over 20 km away). These scenarios covered various aspects, including testing Appium, ADB commands, calling features, SMS, network performance, and other test cases relevant to the certification process.

The results showed consistency in the execution time as more devices are added with ranging between 81.0 and 84.0 seconds for up to four devices, because of parallel execution. In remote testing scenarios from within Montréal, execution times increased slightly due to network overhead with ranging from 85.6 to 90.0 seconds. Moreover, when devices are tested from outside Montréal (over 20 km), execution times increase further to between 86.1 and 95.0 seconds, primarily due to added network latency. Despite these increases, the delay introduced by remote testing remains within an acceptable range of 2% to 7% compared to local execution.

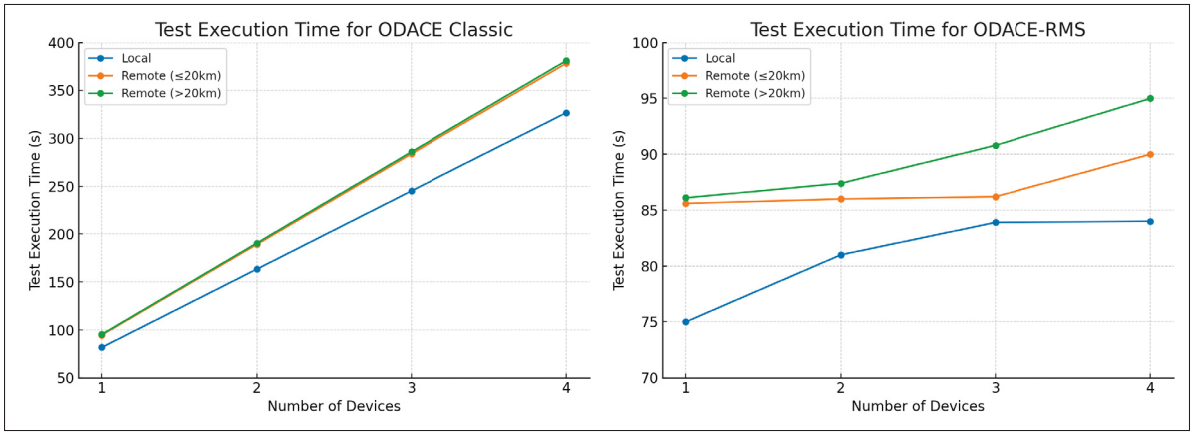

Figure 6.3 Test Execution Time for ODACE vs. ODACE-RMS Across Different Scenarios

6.4 Qualitative Results of ODACE-RMS

In this part of the evaluation, we focus on its impact on usage. We assess its effectiveness in terms of time efficiency, resource usage, and overall utility based on engineer feedback and the survey outcomes.

- **Time Efficiency:** We assess test completion times in specific quarters using ODACE-RMS compared to the original ODACE and Manual testing. The results show reductions due to simultaneous multi-device testing. Figure 6.4 illustrates the time savings ODACE-RMS (2 devices) achieved compared to manual and ODACE for the same number of tests.

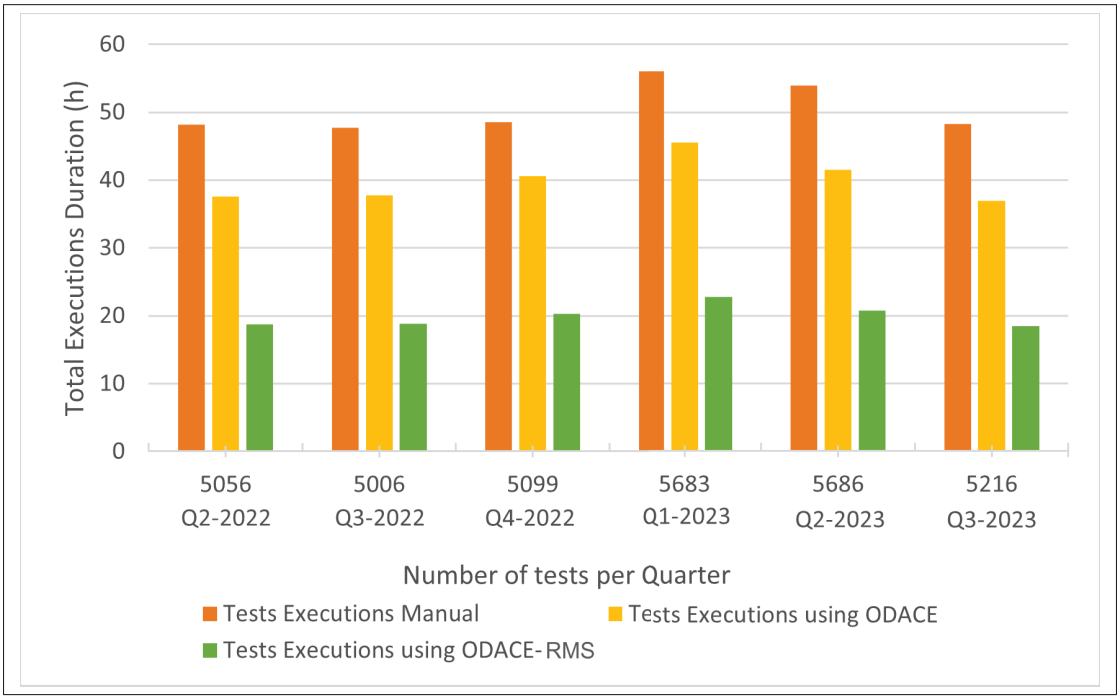

Figure 6.4 ODACE-RMS Improvement Time

- **Resources Efficiency:** We execute a series of tests to compare resource usage across three major browsers (Chrome, Edge, and Firefox). We evaluate that for single-device and multi-device usage. All other applications and browser tabs were closed to isolate the resource consumption of the ODACE-RMS system. A custom JavaScript collected system resource data every 20 seconds, measuring CPU (system and process) and memory usage (which reflects overall system consumption). This evaluation was necessary to suggest the

best browser to use. All three browsers show high CPU usage during the first 50 seconds of startup. After that, usage drops and stabilizes below 30% for system CPU and below 5% for process CPU. ODACE-RMS scales efficiently with minimal overhead, with memory usage increasing by only around 1.6% for Edge, 3.4% for Firefox, and 2.5% for Chrome, which consumes slightly more system memory than other browsers.

Figure 6.5 shows the results for single-device usage, where all browsers have good performance with little distinction for Edge, as it is more stable than the other browsers. Firefox is the most lightweight in terms of memory and CPU use, while Chrome is more resource-intensive initially. All browsers stabilize quickly.

However, in Figure 6.6, we can see that Edge proved better results in the multi-device scenario, consuming the least memory and maintaining a balanced CPU utilization compared to Chrome and Firefox. Also, when we add a device, the CPU load is negligible after initialization, regardless of DUT count. Memory usage grows by 1.6% to 8.8% with the additional DUT.

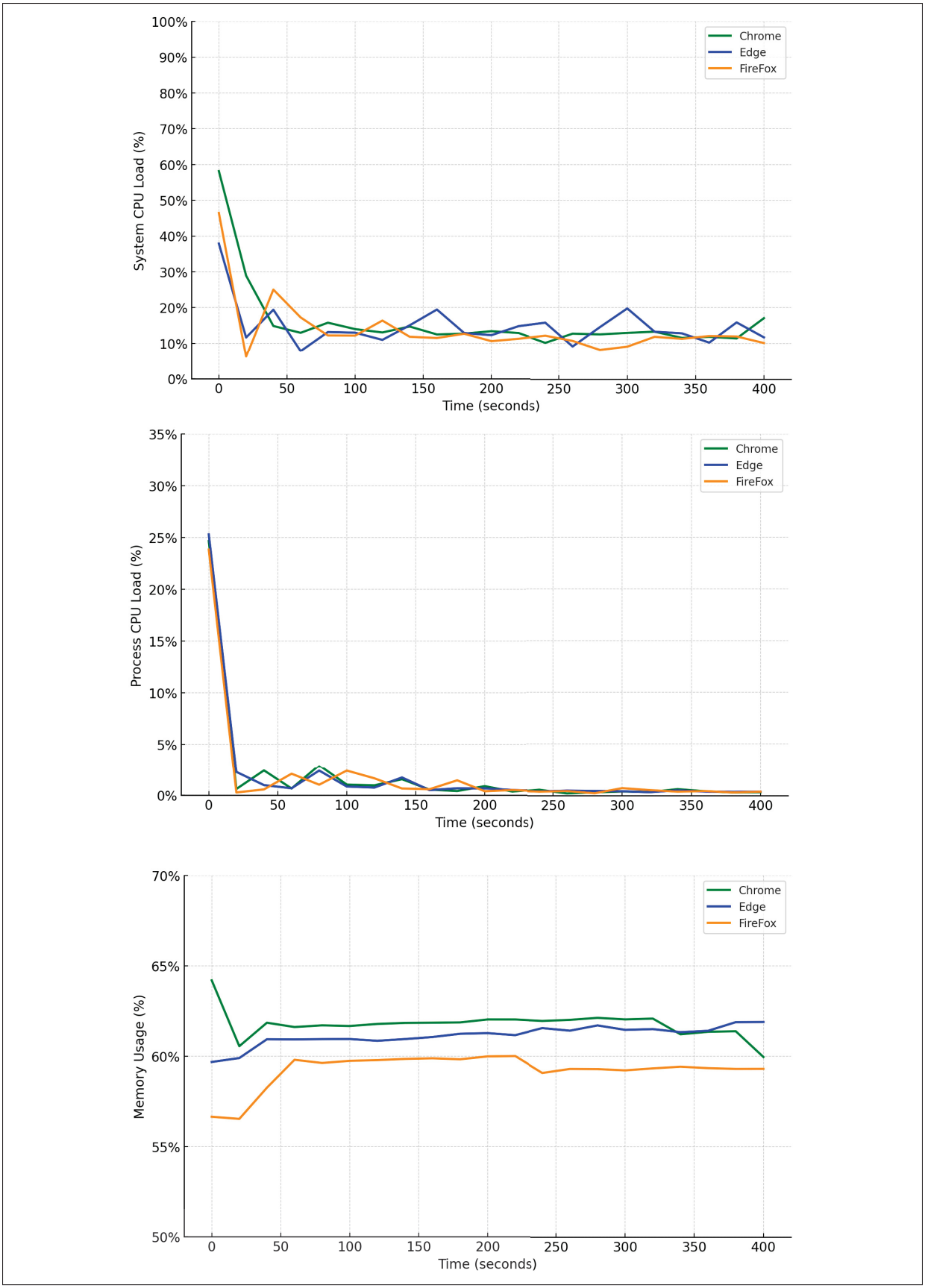

Figure 6.5 Resource Usage Across Different Browsers - Single DUT

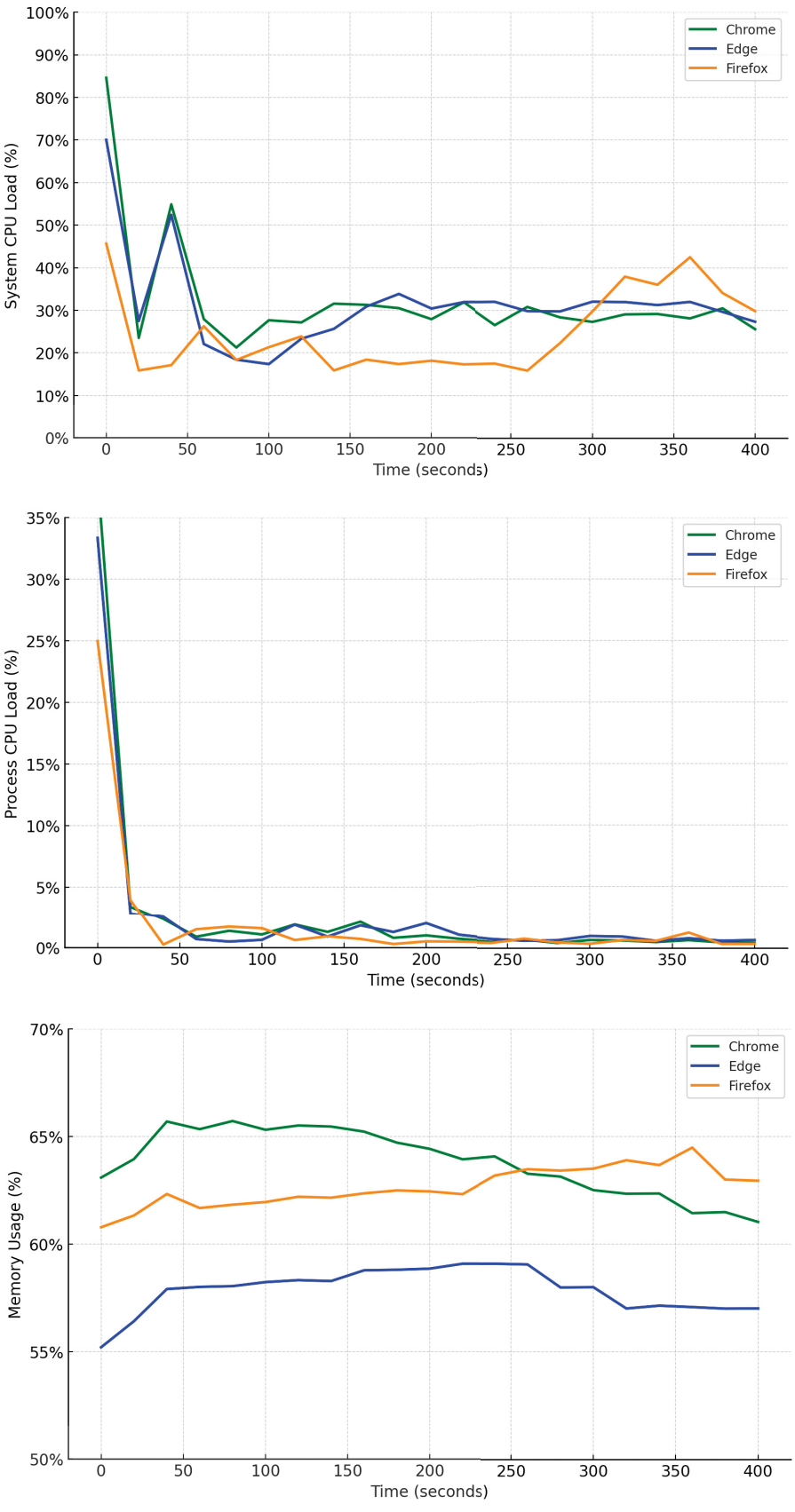

Figure 6.6 Resource Usage Across Different Browsers - Multi DUT (Two Devices)

- **Efficacy:** ODACE-RMS allows users to complete certifications without physical device access. We expect the multi-device feature to boost the number of tests conducted daily or weekly, enhancing performance efficiency.

6.5 Survey and Evaluation of the Remote Feature

To evaluate the efficacy of ODACE-RMS, we surveyed certification engineers at Vidéotron. Participation was voluntary and anonymous, with responses collected from all engineers in the department.

The questionnaire comprised 10 questions. Results indicated that engineers rated the difficulty of lab-only tests (without remote capability) at an average of 3.5 out of 5, which is around 70%. This difficulty rating explains the unanimous support (100%) for hybrid certification testing (local and remote). Additionally, 67% of engineers reported a commute time of 30-60 minutes each way, while 33% exceeded one hour to arrive at the office, highlighting potential time savings weekly with remote testing. Furthermore, 90% agreed that remote capabilities reduce the complexity of cables/devices.

These results indicate strong preference and flexibility gains with ODACE-RMS's remote features as shown in Figure 6.7.

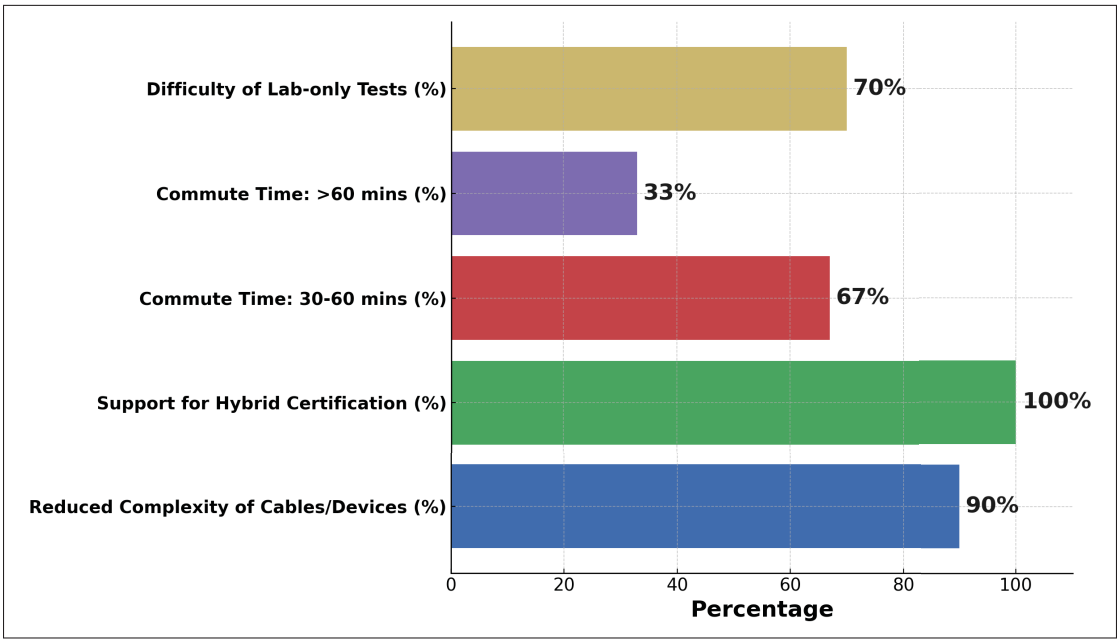

Figure 6.7 Survey Results showing Preferences For Remote Certification With ODACE-RMS

6.6 ODACE-RMS Automation Discussion

While the ODACE-RMS platform has shown clear benefits by reducing user effort, enhancing time efficiency, and decreasing execution times to save certification time, certain challenges remain unresolved. For example, achieving consistency across various device vendors, models, operating systems, and software versions is particularly challenging, making it difficult to maintain a universal code base. As a result, supporting new Android releases oftentimes requires manual updates to the ODACE-RMS code. Another challenge is that specific tests, such as emergency call functionality and audio quality verification, cannot be fully automated and still require human intervention. Additionally, the increased complexity introduced by new features makes platform maintenance more challenging. Although remote testing offers flexibility, some tests still should be done in person, and device issues during remote sessions may necessitate a technician's presence in the lab. To manage this, the Vidéotron lab rotates one CE daily to handle such cases. Remote testing via USB-over-IP may also cause some delays from 7.5% to 11% in some cases when we compare the local execution time with remote execution time (distance of more than 20km) due to the network latency, which could impact the overall testing time.

The testing environment and framework infrastructure can be costly and challenging. Maintaining automation tools and keeping them up to date can also be complex.

6.7 Conclusion

This chapter analyzed the platform's results and discussed the challenges of the solution. The findings demonstrate that the solution meets the main and sub-objectives of the research. Between 2022 and 2024, ODACE successfully automated 52% of the execution tests. As a result, engineers' performance significantly improved, and execution time was reduced by 21.83%. Additionally, the platform saved 89% of the engagement time of the engineers, which clearly shows time and effort savings.

The new feature that enables testing multiple devices in parallel also proved effective. Users experienced time savings with an acceptable delay that did not exceed 7%, even when tests were performed remotely. Moreover, Resource usage remained within acceptable limits—for instance, memory usage increased by no more than 8.8% when additional DUT was tested in parallel.

Finally, a conducted survey indicated that the remote testing feature provides engineers with greater flexibility and supports their certification tasks more efficiently.

CONCLUSION AND FUTURE DIRECTIONS

7.1 Conclusion

The manual certification process traditionally requires few days for each new software update. In this report, we introduce ODACE, a platform designed to address this challenge by automating a part of the mobile device certification process. One of ODACE-RMS's significant strengths is its user-friendly design, which makes it accessible to new employees and interns who may lack technical expertise. By reducing the time needed for routine tasks, ODACE-RMS improves the learning experience of interns during their brief training periods while allowing companies to reallocate resources toward more critical tasks. This approach improves overall efficiency. The results showed that ODACE-RMS significantly reduces time and effort. Additionally, ODACE addresses several challenges highlighted in related studies, including reducing execution time, providing a friendly interface, and supporting comprehensive history and log files for retrospective analysis and evaluation.

The implementation of ODACE has successfully reduced both time and effort involved in testing. Specifically, ODACE automates up to 52% of the certification tests, achieving a 20% reduction in test execution time per engineer each quarter, and decreasing engineer engagement time to just 11% instead of 100%.

To further improve efficiency, and reduce more time. We developed ODACE-RMS, an extended version of the platform that implements a multi-session solution. ODACE-RMS enables parallel testing on multiple devices. On average, ODACE-RMS achieves a 75% time savings when certification engineers (CEs) test four devices simultaneously. Moreover, ODACE-RMS aligns with the remote work trend by allowing tests to be performed without physical device interaction. Our evaluation shows that these features increase user satisfaction and enhance the certification process, making ODACE-RMS a timely and flexible solution.

7.2 Future Directions

Some potential options for improving research directions to enhance ODACE-RMS's capabilities further include integrating the operator's network tracing application into ODACE-RMS, which would expand testing capabilities. Furthermore, using AI-powered image recognition presents a valuable enhancement for managing OS software version updates, effectively addressing challenges related to device compatibility and ensuring consistency across various vendors and operating system versions. In addition, developing support to include iOS devices would significantly extend the platform's applicability across multiple testing environments.

BIBLIOGRAPHY

Abran, A., Moore, J. W., Bourque, P., Dupuis, R. & Tripp, L. (2004). Software Engineering Body of Knowledge. *IEEE Computer Society, Angela Burgess*, 25.

AbuSalim, S. W., Ibrahim, R. & Wahab, J. A. (2021). Comparative Analysis of Software Testing Techniques for Mobile Applications. *Journal of Physics: Conference Series*, 1793(1), 012036.

Alotaibi, A. A. & Qureshi, R. J. (2017). Novel Framework for Automation Testing of Mobile Applications Using Appium. *International Journal of Modern Education & Computer Science*, 9(2).

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D. & McDaniel, P. (2014). FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. *ACM SIGPLAN Notices*, 49(6), 259–269.

Berihun, N. G., Dongmo, C. & Van der Poll, J. A. (2023). The Applicability of Automated Testing Frameworks for Mobile Application Testing: A Systematic Literature Review. *Computers*, 12(5), 97.

Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. *Future of Software Engineering (FOSE '07)*, pp. 85-103. doi: 10.1109/FOSE.2007.25.

Binder, R. (2000). *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Professional.

Boshernitsan, M., Doong, R. & Savoia, A. (2006). From Daikon to Agitator: Lessons and Challenges in Building a Commercial Tool for Developer Testing. *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pp. 169–180.

Cambridge University. [Accessed: 2025-05-28]. Certification. Retrieved from: https://dictionary.cambridge.org/dictionary/english/certification.

Chaves, L. C., Oliveira, F. C. M., Tiago, L. A. & Castro, R. G. V. (2024). Robert: An Automated Tool to Perform Mobile Application Test. *Proceedings of the 2024 10th International Conference on Computer Technology Applications*, pp. 33–36.

Cui, J. (2024). Application of a Mobile Automation Testing Framework: Evidence and AI Enhancement from Chinese Technological Companies. *Journal Academi*.

da Silva, G. & de Souza Santos, R. (2023). Comparing Mobile Testing Tools Using Documentary Analysis. *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–6.

Dahiya, R., Ali, S. et al. (2019). Importance of Manual and Automation Testing. *CS & IT Conference Proceedings*, 9(17).

Digi International. [Accessed: 10 April 2024]. (2023). Digi Remote Manager User Guide. Retrieved from: https://www.digi.com/resources/documentation/digidocs/90002383/default.htm.

Frankl, P. G. & Weyuker, E. J. (1993). Provable Improvements on Branch Testing. *IEEE Transactions on Software Engineering*, 19(10), 962–975.

Godboley, S., Dalei, D., Sadam, R. & Mohapatra, D. P. (2023). Agile GUI Testing by Computing Novel Mobile App Coverage Using Appium Tool. *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pp. 1026–1029.

Gomez, L., Neamtiu, I., Azim, T. & Millstein, T. (2013). RERAN: Timing-and Touch-Sensitive Record and Replay for Android. *2013 35th International Conference on Software Engineering (ICSE)*, pp. 72–81.

Halani, K. R., Kavita & Saxena, R. (2021). Critical Analysis of Manual Versus Automation Testing. *2021 International Conference on Computational Performance Evaluation (ComPE)*, pp. 132-135. doi: 10.1109/ComPE53109.2021.9752388.

Hanna, M., El-Haggar, N. & Sami, M. (2014). A Review of Scripting Techniques Used in Automated Software Testing. *International Journal of Advanced Computer Science and Applications*, 5(1).

Hanna, S. (2008). *Web Services Robustness Testing*. (Ph.D. thesis, Durham University).

Jaffar-ur Rehman, M., Jabeen, F., Bertolino, A. & Polini, A. (2007). Testing Software Components for Integration: A Survey of Issues and Techniques. *Software Testing, Verification and Reliability*, 17(2), 95–133.

Jain, A. & Sharma, S. (2012). An Efficient Keyword Driven Test Automation Framework for Web Applications. *Int. J. Eng. Sci. Adv. Technol*, 2(3), 600–604.

Kaasila, J., Ferreira, D., Kostakos, V. & Ojala, T. (2012). Testdroid: automated remote UI testing on Android. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, (MUM '12). doi: 10.1145/2406367.2406402.

Kemp, Simon. [Accessed: 30 October 2024]. (2024). The global state of Digital in 2024 — DataReportal – Global Digital Insights. Retrieved from: https://datareportal.com/reports/digital-2024-october-global-statshot.

Kim, H., Choi, B. & Yoon, S. (2009). Performance Testing Based on Test-Driven Development for Mobile Applications. *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, pp. 612–617.

Lamy-Poirier, J. (2023). Breadth-first pipeline parallelism. *Proceedings of Machine Learning and Systems*, 5, 48–67.

Le Traon, Y., Baudry, B. & Jézéquel, J.-M. (2006). Design by Contract to Improve Software Vigilance. *IEEE Transactions on Software Engineering*, 32(8), 571–586.

Li, A. & Li, C. (2022). Research on the Automated Testing Framework for Android Applications. *International Conference on Computer Engineering and Networks*, pp. 1056–1064.

Li, J. & Cao, H. (2023). Design and Implementation of API Automation Testing System for Mobile Hybrid Mode Based on Appium Technology. *Proceedings of the 2023 7th International Conference on Electronic Information Technology and Computer Engineering*, pp. 1478–1484.

Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Che, X., Wang, D. & Wang, Q. (2024). Make LLM a Testing Expert: Bringing Human-Like Interaction to Mobile GUI Testing Via Functionality-Aware Decisions. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13.

Lukić, N., Talebipour, S. & Medvidović, N. (2020). Remote Control of iOS Devices via Accessibility Features. *Proceedings of the 2020 ACM Workshop on Forming an Ecosystem Around Software Transformation*, pp. 35–40.

Lyu, M. R. et al. (1996). *Handbook of Software Reliability Engineering*. IEEE Computer Society Press Los Alamitos.

Mojahed, S., Drouin, R. & Sboui, L. (2024). ODACE: An Appium-Based Testing Automation Platform for Android Mobile Devices Certification. *2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 301-308. doi: 10.1109/ICSTW60967.2024.00060.

Motwani, A., Agrawal, A., Singh, N. & Shrivastava, A. (2015). Novel Framework for Browser Compatibility Testing of a Web Application Using Selenium. *International Journal of Computer Science and Information Technologies*, 6(6), 5159–5162.

Rao, A. K., Prasad, S. & Rao, E. (2012). Quality Benefit Analysis of Software Automation Test Protocol. *International Journal of Modern Engineering Research*, 2(5), 3930–3933.

Salam, M. A., Taha, S. & Hamed, M. G. (2022). Advanced Framework for Automated Testing of Mobile Applications. *2022 4th Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, pp. 233–238.

Samad, A., Nafis, M. T., Rahmani, S. & Sohail, S. S. (2021). A Cognitive Approach in Software Automation Testing. *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*.

SEGRON. [Accessed: 31 January 2025]. (2024). Automated Testing Framework (ATF). Retrieved from: https://segron.com/products/automated-testing-framework/.

Sinaga, A. M., Wibowo, P. A., Silalahi, A. & Yolanda, N. (2018). Performance of Automation Testing Tools for Android Applications. *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp. 534–539.

Sinaga, A. M., Pratama, Y., Siburian, F. O. et al. (2021). Comparison of Graphical User Interface Testing Tools. *Journal of Computer Networks, Architecture and High Performance Computing*, 3(2), 123–134.

Singh, S., Gadgil, R. & Chudgor, A. (2014). Automated Testing of Mobile Applications Using Scripting Technique: A Study on Appium. *International Journal of Current Engineering and Technology (IJCET)*, 4(5), 3627–3630.

Suomalainen, J., Julku, J., Vehkaperä, M. & Posti, H. (2021). Securing Public Safety Communications on Commercial and Tactical 5G Networks: A Survey and Future Research Directions. *IEEE Open Journal of the Communications Society*, 2, 1590–1615.

Thant, K. & Tin, H. (2023). The Impact of Manual and Automatic Testing on Software Testing Efficiency and Effectiveness. *Indian Journal of Science and Research*, 3(3), 88–93.

Tran, Hieu Minh and Ninh, Tuan Duc and Tran, Thinh Duc and Van Ngo, Vuong and Nguyen, Linh Duc. (2023). Automation Testing with Appium Framework in IP Multimedia Subsystem. *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 579-582. doi: 10.1109/ICTC58733.2023.10392322.

Vajak, D., Grbić, R., Vranješ, M. & Stefanović, D. (2018). Environment for Automated Functional Testing of Mobile Applications. *2018 International Conference on Smart Systems and Technologies (SST)*, pp. 125–130.

Verma, N. (2017). *Mobile Test Automation With Appium*. Packt Publishing Ltd.

Wang, J. & Wu, J. (2019). Research on Mobile Application Automation Testing Technology Based on Appium. *2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pp. 247–250.

Wang, W., Li, D., Yang, W., Cao, Y., Zhang, Z., Deng, Y. & Xie, T. (2018). An Empirical Study of Android Test Generation Tools in Industrial Cases. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 738–748.

Yoon, J., Feldt, R. & Yoo, S. (2023). Autonomous large language model agents enabling intent-driven mobile gui testing. *arXiv preprint arXiv:2311.08649*.

Yu, S., Fang, C., Du, M., Ding, Z., Chen, Z. & Su, Z. (2024). Practical, Automated Scenario-based Mobile App Testing. *IEEE Transactions on Software Engineering*.

Zhu, H. (1996). A Formal Analysis of the Subsume Relation Between Software Test Adequacy Criteria. *IEEE Transactions on Software Engineering*, 22(4), 248–255.