BIRZEIT UNIVERSITY

Master Thesis

React Native Testing: Pruning GUI Model Approach

Author:

Rand Ibrahim (1195280)

Supervisor:

Dr. Samer Zein

February 21, 2023

Abstract

Nowadays, the world is facing an explosive growth of mobile application development field, which is increasingly considered an important role in everyone's life. People are migrating to smartphone mobile devices to accomplish their daily activities while working, playing and communicating with others. The mobile software technology comprises a wide variety of platforms, technologies, and architecture choices.However, because of the constant changes on software application and the increase evolving in this field, developers are supposed to speed up the development process to satisfy the customer's needs and provide robust applications within a short time period. Therefore, cross-platform development technology aims to overcome these difficulties, where instead of building separate application for each platform, a single application that can be run on multiple platforms is developed.

Model-based testing is a testing technique in which the test cases are derived from a model that describes the different aspects of the system under test. Using this technique in cross-platforms, especially the React Native framework increases the efficiency and makes it faster and easier to identify and find defects and bugs can be quickly. However, once the complexity of the program increases, the model-based testing complexity will remarkably increase. Therefore, this study proposed the **R**eact **N**ative **A**bstract **S**yntax **Tree** Pruning (RN-AST pruning) framework, which based on the idea of pruning the original GUI model of the application that was built using static analysis, to keep only the impacted regions from internal code changes. Thus, help and guide testers while testing. Our study was tested on a mobile application that was implemented to illustrate the idea of the framework, and it was evaluated using a case study evaluation with the help of mobile developers and engineers to ensure that the proposed framework guide testers in the process of detecting bugs and defects in React Native applications.

Results shows that RN-AST Pruning framework idea is useful and provides the test engineers with the affected files and paths that need to be tested. Moreover, it identifies exactly the changes occurred in each file and categories them to updates, placements, and deletions based on the differences between the original version and updated version of the source code, which provides the test engineers with accurate results.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I am extremely grateful to my supervisors, Dr.Samer Zein for his continuous support, and patience during my master thesis preparation. His immense knowledge and advises have encouraged me all the time. I would also like to thank the volunteers that helped me in evaluating the framework. Finally, I would like to express my gratitude to my family, my husband and my friends for their tremendous understanding and encouragement in the past few months.

Chapter 1

Introduction

In this chapter a general overview about mobile application development, the need to have a cross platform, and the important of having a robust testing process are presented. The motivation behind this research is also discussed, as well as, the research problem. Finally, the research questions are identified, and report structure is presented.

1.1   Overview

Nowadays, with the heavy reliance on technology, mobile application development is evolving rapidly and moving quickly toward being a mainstream. Mobile devices and smartphones are becoming an integral part of our daily life. Almost 60% of population are accessing the internet using their mobile devices, which increases the need to support the ongoing development in this area. Theoretically, this seems to be easy, however technically this is complex because of the rapid development nature and imposed limitations for the mobile devices [2], especially that the user expectations about the mobile applications are remarkably high[3].

In general, the diversity of mobile platforms makes mobile development process quite complex and expensive, particularly with the need to build the application for each mobile operating system. This raises the necessary to have a cross platform development to contribute in solving this problem.

Cross platform mobile applications are those applications that are built to run on multiple mobile platforms and operating systems like Android and IOS. Such applications give developers the ability to launch software simultaneously on various platforms, makes the development process faster than before because as a developer you need to deploy only single script to run against different platforms. Moreover, it offers the opportunity to reach wide range of audience. Furthermore, using cross platform saves money and time, where the time to market

will be reduced, which increases the application revenues.

Despite the huge advantages, using cross platform technologies still has downsides and limitations. The largest risk is the maturity of this technology [4], as cross platform development is still relatively young. IOS and Android support was released in 2015, the documentation and improvement continues to evolve. Moreover, some features of IOS and Android still aren't supported and different practices are still under process.

React Native is a cross-platform mobile development framework uses JavaScript to build user interfaces, but instead of targeting the browser, it targets mobile platform [4]. The "bridge" in React Native is responsible in invoking the rendering APIs, where Objective-C API is used for IOS and Java API for Android platform.

React Native has several advantages [5]. The biggest advantages of using React Native is that developers do not need to build a separate mobile application for each platform, but a single code-base can be used to build a mobile application on multiple platforms, which save cost and time. Since React Native is a JavaScript framework, then there is no need to rebuild the application to see the modifications, which actually increases the productivity and reduces the compilation time. Another advantage is that React Native applications are built by rendering "native" mobile UI, so smoother and better application run. Moreover, React Native is an open-source framework, which means that support is widely available online and there exists plenty of ready-made and reusable libraries that facilitate the application development process. In addition to above advantages, using React Native ensures the user interface and user experience consistency between the different platforms.

Testing process is important in determining the quality of the software. It consumes 40-50% of the development efforts and sometimes more for software that required high level of reliability [6]. In general, software testing represents the final review of specification, design and coding. It is usually done to detect system defects, increase the reliability of the application, and find gaps and missing requirements in comparison to the actual requirements. Moreover, testing reduces the cost of changes and required maintenance if required.

Existing mobile application' upgrade and maintenance are grown rapidly, which increase the need to have a sophisticated and cost-effective software testing to ensure that systems satisfy the customers' needs after change[7]. Despite the great development in mobile testing techniques, testing process still needs to be carefully implemented to assure the quality of the system of non-trivial complexity; especially those used critical domains like health, banking and payments.

It is true that testing process became more efficient and more productive, especially with the use of different mobile automation testing tools. However, with the constant changes on software applications, testers face more challenges in maintaining test suites and keeping these test scenarios up to date, especially when using exploratory testing, which is based on the testers experience and information about the system to be tested. This is due to the large number of test cases and the required time to implement and maintain them. For instance, when the GUI elements change, the test cases usually fail to run because these test cases can't cope with the changes in the source code.

Many research projects have been conducted in different aspects related to the field of cross platform, especially React Native field. According to the testing field, Nader at al [8] have focused on the different challenges facing the cross-platform development and testing process. Yepeng Yao and his colleague [9] proposed an automated, distributed and cross-platform testing framework for GUI-driven application. Moreover, an automated testing tool for cross-platform called MobiTest was proposed by Bayley [10] and his colleagues. Wang et al [11] addressed the ability of migrating GUI test cases from one platform to another by proposing a novel approach called "TestMig" responsible for GUI test migration from ISO and Android. However, to the best of our knowledge, non of these studies covers the static analysis and GUI model pruning to facilitate the testing. Therefore, in order to improve the testing of React Native applications and to guide the testers while testing, RN-AST Pruning framework aim to extend and adapt the proposed approach by Reis and Mota [12] to acclimate with React Native frameworks' proprieties.

This approach depends on joining the manual and automation testing by applying the static analysis on the source code, to provide a GUI-Model of the impacted regions after doing changes, and then in order to keep the internally affected and changed elements in the source code, the resulted GUI model will be pruned using reach-ability analysis. Doing so, will contribute in facilitating the testing process and guide testers while testing, so widens the spread of React Native framework.

1.2 Motivation

Currently, there is a large involvement and movement in the field of mobile development. Developers are racing to build and provide robust software that satisfies the customers and users expectations and needs. Such competition between developers and companies increases the difficulties and challenges in proving themselves and staying in the competition zone. This

actually requires them to keep their application up to date, make modifications and mainte-nance in a fast manner to enhance the performance and efficiency of the provided applications. React Native development has proved its efficiency in providing and building applications that can be run on multiple mobile platforms like IOS and Android, which are the largest mobile platforms around the world. Testing these applications is a crucial phase in the development life cycle to ensure that the application achieve what is expected and please its users.

Testing process has different approaches, model-based testing is one of them. Where, in short, can be defined as using an application model that describes its behavior to test the application and ensure it is as required. Therefore, from that point, this research discusses using such technique in testing cross-platform applications, especially those implemented using React Native framework. However, instead of using the entire model to test the application, why not to prune it to show only the parts that are affected due to different code changes or new enhancement. Pruning the model will help testers, guide them through the testing process, facilitate the testing process, make it easy to keep the applications updated and launch that applications to the market faster than before.

1.3   Research Problem

Despite the great adoption of cross-platform development and the rapid evolvement in this field, particularly React Native framework, there still a lack in the existence of frameworks that assist the test engineers in testing the graphic interfaces in mobile applications and provide the testers with a subset of test cases to test instead of testing the whole test cases. In general, it is very hard and time consuming to maintain all test cases and keep them up to date in the short time between the rapid releases, which increases the overhead on the testers and complicates the testing process. Accordingly, there is a need to provide a framework that can assist the testing process using model-based testing by pruning the entire model to help testers and guide them in reaching the modified GUI parts of the application impacted from the internal changes on the source code. Doing so, will provide the testers with a guiding plan and a road map to test the application, so reduce the testing process complexity, reduce the paid effort, time and cost and it will ensure that the customer's needs and requirements are achieved.

1.4 Research Questions

As mentioned before, the main goal is to facilitate the GUI testing process, assist and guide test engineers by pruning the application model to keep only the affected GUI parts that are related to the internally changes in the source code.

Accordingly, the following research questions are formulated:

RQ1: How to detect the GUI elements and prune the GUI model?

RQ2: How to calculate and classify the code differences and changes between the last two versions of the application?

RQ3: How to build the list of paths that contains the changes file?

RQ4: How effective is the build framework in detecting changes and results that satisfy the test engineers?

1.5 Report Structure

This research report will be organized as follow: chapter 2 is the background chapter that describes the main concepts in this research. Chapter 3 provides an overview about the latest researches done in the field of model-based testing and identifies the gaps in the existing studies and theories. Then, the followed methodology and its phases are explained in more details in chapter 4. Finally, a brief conclusion that summarizes the research and the future plan were presented in chapter 6

Chapter 2

Background

2.1 Overview

In this chapter, a technical overview about cross-platform technology and model-based testing will be introduced. Moreover, the static and dynamic analysis will also be explained in more details to provide the reader with a general overview about these concepts.

2.2 Cross-platform Development

From its name, cross-platform development is the practice of building software that is compatible with more than one type of hardware platform. Cross-platform application can run on Microsoft Windows, Linux or even mac OS [13]. In this type of development, as a developer you can build the code-based once and run it on many platforms. Different languages can be used to build and create a cross-platform application like JavaScript or C#.

Creating cross-platform mobile application makes it easy to access wide range of audience, since it runs on different platforms like Android operating system and IOS, which are the most famous platforms. Moreover, cross-platform by default deals with the differences between IOS and Android, which helps in building consistent application on both platforms.

In general, there exists several frameworks that help in building hybrid applications that work on most mobile platforms. However, despite the variety of cross-platform framework, it is important to know and realize that there's no framework fits and ideal for everyone. The choice of framework depends on project and the aim of each project. For example, Flutter is a popular framework that is widely used by many developers for building mobile, web, and desktop apps from a single code base. It uses *Dart* as the programming language and it is used in eBay, Alibaba, Google Pay, ByteDance apps.Furthermore, Ionic framework is

another example of cross-platform framework that is used to build hybrid mobile and desktop applications using a combination of native and web technologies. Ionic is based on a SaaS UI framework that provides multiple UI components for building mobile applications.

Moreover, React Native is the most common cross-platform framework. Below section will give a wide information about this framework.

2.2.1   React Native

React Native is a JavaScript open source mobile application development framework to build native IOS and Android applications. It was classified with the 2nd highest number of contributors in GitHub in 2018. React Native was developed by Facebook Company (Meta now) in 2015 and nowadays, it is adopted by many companies such as Netflix, Dropbox and many other companies. It is based on React, which offers a fresh approach to create user interface with JavaScript. It was actually done to extend the mechanism of React to native mobile application development.

With React Native, the paradigm is "learn once, write anywhere." With this approach, an experienced web React developer can get up and write Android or IOS apps at a much faster pace.

There exist two ways to start building React Native applications. The first is to use React Native CLI and the second is to use the Expo CLI. Using the first option may be time consuming due to the extra complex configuration required to setup the environment to build the react native application. When using React native CLI option you need to be familiar with mobile development because it needs both Android and iOS emulators, so this means it requires Android Studio and XCode to run the application, which increases the complexity. While the second option is to use Expo, which is a set of tools and services that allow you to develop apps for both Android iOS without having to deal with Java-Kotlin and Swift native code. To build the application using Expo you will only need a recent version of Node.js and a phone or emulator. When run the application,Expo generates a QR code for the application and to run the application on Android or iOS devices, you only need to scan the generated QR code using Expo Go application, which should be installed on the device. Therefore, you do not need an emulator or a Mac to run React Native apps with Expo.

Using React Native provides multiple advantages [5]. For example, it offers a *hot reload feature* that increases the development process speed by showing the changes made on the application screen immediately without rebuilding the application, which increases the productivity and

reduces the compilation time. In addition, reusability is considered one of the biggest benefits of using React Native, where instead of building and creating separate mobile application for reach platform, the developers can adapt one created structure to be used in the other platform, since React Native has pre-developed components in its open-source library that can be used by developers instead of writing it from scratch. This feature allows the developers to reuse almost 90% of the codes on both operating systems platforms. Furthermore, React Native is considered an open-source platform, which means that it is free to access and modify the source code. It also has a big support community can help in case developers face struggles or any obstacles in solving problems. In term of cost, it is significantly cheaper to create an application with React Native, since only a single development team directly creates the application instead of having two teams. Thus, easier to manage and track the progress and less required resources to build the mobile applications. Moreover, having single development team has a positive effect on communication. Furthermore, an important point to take into consideration from business perspective is the consistency between the versions of the Android and IOS applications. Using React Native ensures that the user interfaces and user experience is the same on the different platforms.

However, React Native still has a dark side and a set of challenges as well. For instance, it was developed and presented to the market in a short time and it is still fresh and immature, which causes low application performance and may harm the application if compared with native applications. Due to this issue, developers might face various issues when it comes to packages compatibility or debugging tools, which will negatively influence the development process. Moreover, due to the its immaturity, React Native still lacks some components and others still underdevelopment, which may obstructs the development process if developers want to build an application from scratch. At the core of React Native technology, developers must have a concrete knowledge about the web and native technologies and they must have the ability to work on JavaScript, project configuration UX guidelines, etc. Therefore, it is not an easy to build a cross-platform team.

In order to understand the technical part of React Native, there is a need to recall the *Virtual DOM (Document Object Model)*, which is one of React's features. [4].

The Virtual DOM in React acts as a layer between the developer's description about how things should look like (developer code), and the work done to render the application onto the page. In general, to render interactive user interfaces in a web browser, the browser's DOM must be edited and this is actually an expensive step since writing on DOM impacts

the performance. Therefore, instead of rendering the changes on the page directly, React makes the necessary changes by using an in-memory version of the DOM and re-renders the minimal amount of changes[4].

In addition to the performance benefits that the Virtual DOM offers, its real optimization lies in the power of its abstraction. It acts as an abstraction layer between the developer's code and the actual rendering.

Figure2.1 shows how React Native works. As we can see the React native is made up of two sides, the JavaScript side and the native side. The native side could be Objective-C/Swift for iOS or Java/Kotlin for Android or other like web or desktop. The figures shows that the Bridge component is responsible for allowing both sides to talk to each other, where instead of rendering to the browser's DOM directly, React Native recalls Objective-C APIs for rendering IOS components or Java APIs for rendering Android components.

Mainly, React components uses *render* function to return markup to describe how components should look like. In React for the web, this markup is directly translated to the browser's DOM. However, in React Native the markup is translated to suit the host platform based on the bridge component. As a result, React Native has the ability to target other platforms by just writing the bridge component.
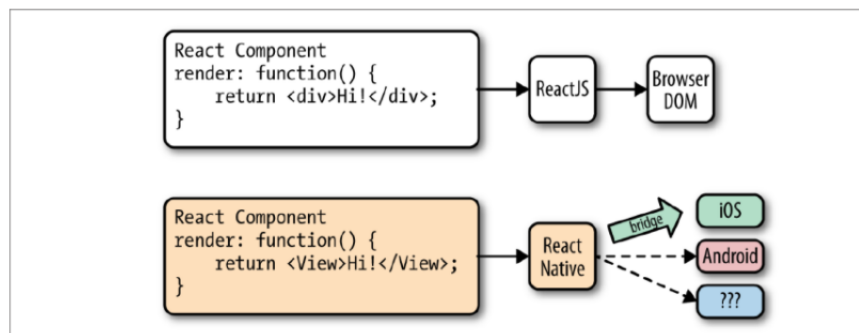
FIGURE 2.1: Rendering React to different targets

In general, when React runs in the browser, the render life-cycle begins by mounting the React components. Then, handling the rendering process of components. According to the rendering process, the developer return HTML markup from the *render* method of the React component. Thus, React renders the component directly into the page.

For React Native, the life-cycle is similar to React life-cycle [4], but the rendering process has some differences because React Native depends on the *bridge*. The bridge is responsible for

translating the JavaScript calls and then invoking the API of the host platform, which may be Objective-C for IOS and Java for Android .

Views are the basics building blocks of the user interfaces in both Android and IOS. In general, the views are small rectangular elements on the screen used to display text, images or respond to user input. In Android development, views are written in Kotlin or Java; and in IOS Swift or Objective-C are used to write the views. However, in React Native these views can be invoked with JavaScript using React components, and at run-time React Native will be responsible for creating the corresponding Android and IOS views for those components, which are called *Native Components* by the help of bridge.

In general, there exists a set of essential, ready-to-use Native components comes with React Native and can be used to start building the application [5]. These components are called the *Core Components*. Core components make it easy for anyone to dive into React Native Applications development.

2.3 Model-Based Testing

As the applications' code-base expands, small errors and edge cases can cause larger failures and lead to bad and unsatisfied user experience, which influenced the application negatively. Therefore, testing the application before releasing is important to prevent fragile programming and make sure that the application is working as expected. Moreover, it ensures that our code will work properly in case we added new features or make changes to existing features. Thus, testing process has many values and good tested application increases the quality of that application and makes it reliable and easy to use, which encourage people to use it, so increase the overall revenues.

In general,manual testing is no more accurate to test the different mobile applications and software testers can no more rely on it to provide a good application. Therefore, it has become a crucial need to have an automated testing. Model-based testing is considered one of the best approaches to ensure that the application works as required and satisfies the audience needs.

From its name, model-based testing (MBT) is simply a testing technique in which the test cases are derived from a model that describes the functional aspects of the system under test. MBT describes how the system would react and responds to an action and see it the application responds as excepted. It basically can't be used and introduced suddenly in the system,

instead it has to be done gradually. Figure 2.2 taken from [14] illustrates the different stages of model-based testing.
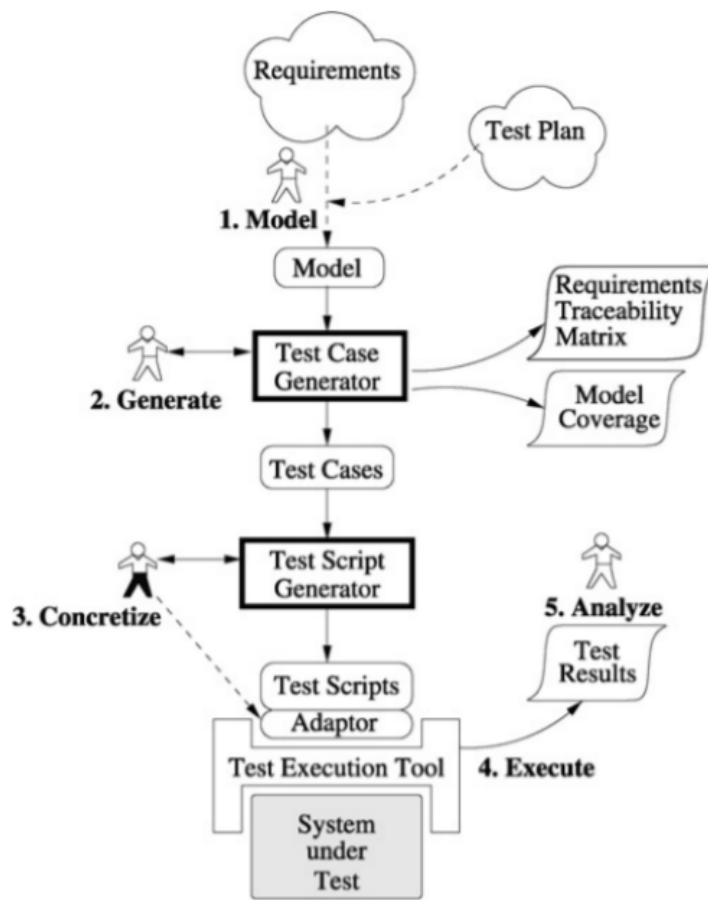
FIGURE 2.2: Model-Based Testing Process

Model-based testing has five stages:

- **Model**: this step aims to build an abstract model to represent the different aspects of the software under test. Different ways can be used to model the software like using the UML or the Finite State Machines.

- **Generate**: where a set of abstract tests are obtained and generated from the model generated from previous step.

- **Concretize**: in this step the abstract tests are converted to a set of test scripts to make them executable.

- **Execute** the set of test scripts on the software to be tested

- Perform an **analysis** to the test execution results and make the corrective action

Basically, it is important to know that the effectiveness of the generated test cases depends on the correctness and completeness of the generated models.

This kind of testing approaches has different advantages. It optimizes the software testing time and allows the developers to focus of writing models that cover the different system requirements. Moreover, it reduces the maintenance cost and it generates minimal number of test cases that ensure and validate the software functionalities, provide more code coverage and increase the efficiency of the testing process and less error-prone. However, MBT still has difficulties and disadvantages; it actually requires a skilled testers team to focus on building a testable software and models that describe the real-life user experience, which means a steep learning curve. Another challenge in MBT is that it needs a deep understanding of the application architecture.

In general, there exists several type of models that describes the different aspects of the system behavior. For example, the *State Transition Diagram*, this model helps in estimating the testing results based on the input selected. Moreover, various combinations of the input result in a corresponding state of the system. This model illustrates that the software can, at any point in time, be in a specific state from a finite set of possible states. The *Unified Modeling Language (UML)*, it is a modeling language that is used to create visual models that describe the different system behaviors. Another model type is the *Data Flow Model*, it is a visual representation of the information flows within a system. This type of modeling shows how data enters the system and how it leave, it also shows what changes the information and where data stored. Form Data flow model we can determine the scope and boundaries of the system. Another type of models is the *Control Flow Graph*, it actually represents the flow inside a program unit, and it also shows the paths to be traversed during the program execution. A *Decision Tree Model* is another model, where it represents the different actions to be performed based on given conditions using if-then-else and switch statements. Using this type of models helps the testers in searching the effects of combinations of different inputs. Moreover, it also proves its effectiveness in dealing with complex business rules.

2.4 Code analysis

In general, developers aim to write a bug-free code that meets the requirements and design specifications and prevent security issues. Therefore, in order to ensure that these three goals have been met a code analysis should be conducted.

There exists two types of code analysis, the static code analysis and the dynamic code analysis.

Both play an important role in the development and testing process. Below each type was taken into account separately.

2.4.1 Static Analysis

The importance of software testing process was described previously in this work, where it play a crucial and important role in providing a good quality software products. The main objective of software testing is to apply a set of techniques and strategies to detect errors and failures in the software either in real or simulated environments[15]

In general, code review had proved its efficiency in producing a good quality and reliable software. Therefore, this approach was adopted by many development teams and projects. Static analysis approach is an approach to review the source code of the application, make sure it applies specific rules and adheres to industry standards; it is applied without the actual execution of the program as opposite of dynamic analysis which is done by executing the programs [16]. Theoretically, this analysis type can examine the source code itself or the complied form of the source code, where during the static analysis, the code is transformed into some intermediate model or abstract representation model to match come recognized code patterns [17]. Static analysis is usually performed early in the development before testing begins, and that what makes it different from dynamic analysis, which identifies defects after running the application and during the testing process. However, there might be missed errors and defects in dynamic analysis that static analysis can find.

Figure 2.3, illustrates the different steps of static analysis process [18]. As seen, the process starts by the *lexical analysis*, which means trying to understand the code lines by breaking it down into smaller chunks *"tokens"*.

These tokens could be any valid entity in the programming language like literals, variables, operators and function calls. This phase ignores and discards those characters that do not contribute in the semantics of the program like white-space, comments, etc.

The second phase of the static analysis is the *syntactic analysis*. At this step, the parser takes the tokens that we already have from the previous phase and validates that the sequence of these tokens conforms to the grammar. Then, it organized them in an *abstract syntax tree (AST)* that represents a high-level structure of the program. Note that, since there exists several programming languages, the AST representation for each language may differ from the other.
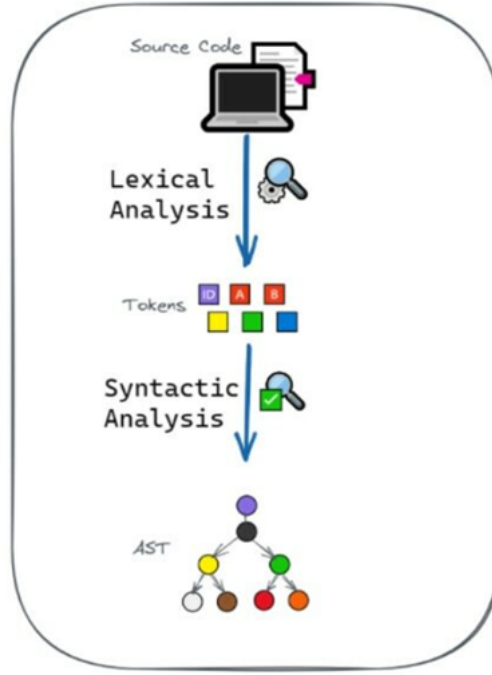
FIGURE 2.3: Static Analysis Steps

In this research approach, static analysis will be used to build the **abstract syntax tree** for the mobile application.

Abstract Syntax Tree (AST)

Abstract Syntax Tree is a tree-structural representation of source code that describes the code snippets of a specific programming language [19], where every node in the tree has at least the type it represents like *Literal, VariableDeclarator, MemberExpression*, etc. It is widely used in compilers to represent the structure of the source code, since it is the result of syntax analysis phase of the compiler.

During the process of building and generating the AST, some parts of the code that don't affect the semantics of the code are discarded while other are preserved . The preserved information are those that are vital to the AST purposes, for example:

- Variable types, its location

- The order and definition of executable statements

- Identifiers and the assigned values

- The left and right components of binary operations

The AST Explorer is a tool helps in visualizing the ASTs in different popular programming languages like JavaScript, Python and java. Below is a screenshot of the produced AST for a simple JavaScript code 2.4.
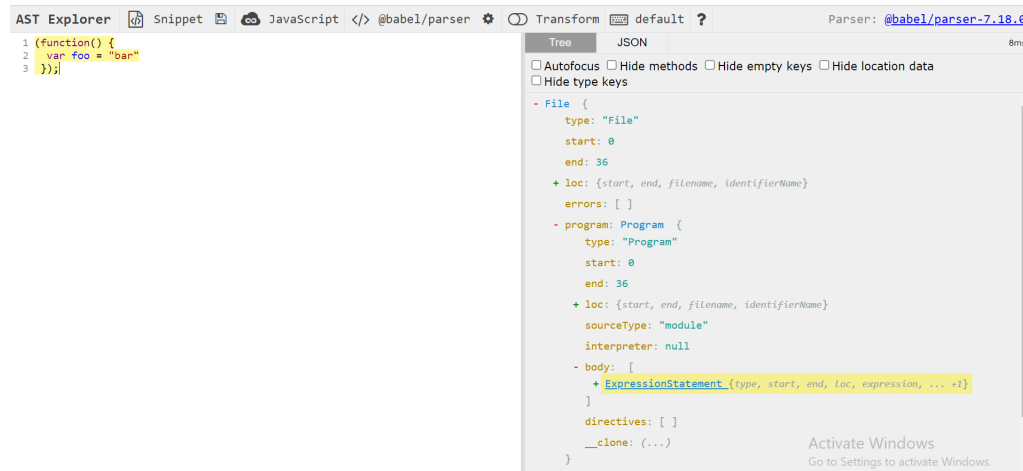
FIGURE 2.4: AST for simple JavaScript function

One thing to note is that there is no "one" AST format. They might differ based of the programming language and the used parser.

2.4.2   Dynamic analysis

The alternative for static analysis is the dynamic analysis, which can be defined as the process of testing and evaluation the code while software is running [20]. This type of analysis is widely used by developers that are under pressure to deliver clean and safe applications faster, where dynamic analysis tools help them in debugging the running threads and processes of the applications, it also help them in finding performance problems and memory leaks during the application execution, so the ability to find the impact on the reliability of the application. It is important to know that both analysis approaches are complementary to each other, since no single analysis approach can find every error. Dynamic analysis is used to reveal too complex defects and vulnerabilities that are hard to be discovered by static analysis. While static analysis examines all possible execution paths and variable values not only those invoked during execution.

Chapter 3

Literature Review

3.1 Overview

This chapter provides an overview of the current knowledge in the field of model-based testing in mobile applications graphical user interfaces; it identifies the relevant theories, studies and determine the gaps in the existing researches.

Writing good literature review does not mean only summarizing the sources, instead it analyzes, synthesizes and evaluates these knowledge areas to give a clear and comprehensive overview about the main topic of the research. In this study, 22 papers were selected based on the selection criteria explained below. These papers grouped into two categories. The first group explains the automatic model-based testing and the other group talks about maintaining the graphical user interfaces.

The rest of this chapter is organised as follows. Section 3.2 explains the search methodology, the used terms, the search databases and the selection criteria. The remaining sections provide a more comprehensive overview about the different categories found when analyzing and synthesizing the study papers.

3.2 Search Method

During this research a thorough and rigorous search method has been conducted to provide a clear vision about the searched topic , reduce the bias ,increase the transparency, and enable us to identify the gaps in the existing researches. It requires a careful and structured search process with a good search terms in a reliable databases

The following subsections will clarify our search terms and the sources databases and the search result.

3.2.1   Search Key wording

In general, finding good search terms is an expensive process that worth spending some time on it because search terms are very crucial and important in determining the search results and outputs. Researchers are supposed to find the most relevant terms and keywords to cover the different aspects of the research topic. Finding the appropriate search strings needs some creativity and ability to consider all possible synonyms, related words and abbreviations. Since my main topic is related to GUI model-based testing for cross-platform mobile application and GUI ripping, my search was centered around these fields. Therefore, here are some search words and strings that I used during my search:

- Mobile model-based testing

- GUI ripping

- GUI model generation for mobile application

- Mobile application static analysis

- GUI testing

- Mobile Test automation

- GUI models maintenance

- Static analysis for model-based testing

3.2.2   Source Databases

According to the databases to search in, it is important to get and search in a comprehensive and trusted list of sources, so that ensure covering most of the searched areas and fields without missing relevant studies. The internet is full of different resources to search in. However, in this research the search process was conducted mainly using the Google Scholar web search engine, which provides a broadly search for scholarly literature. The literature studies were adapted from the following databases, which are the most reliable and trusted to search in.

- IEEE Xplore

- ACM Digital Library

- ResearchGate

- SpringerLink

3.2.3   Selection Criteria

As mentioned in the above sections, the search process was done using different data sources and using different search terms and strings. hundreds of studies related to our main topic were found and different strategies were used to select the most appropriate studies. Moreover, snowballing process were used, which broadened the scope of our search.

In general, to start the filtration process, after taking the search output, we applied the inclusion and exclusion criteria that is illustrated in the below table 3.1. After that, we skim read the resulted papers, where we read the abstract and the introduction and based on them and if related to our main topic we continue the reading process to gain a more comprehensive overview about each study. Following search protocol, we end up with about 22 strongly related papers. These papers were read more carefully, then we dumped the information into a table contained the: **title, year, priority, research problem , methodology, Result and General notes(if needed)**

Inclusion Criteria	Exclusion Criteria
Published within 5-6 years	Old studies
Study size >= 4 pages	Full text cannot be obtained
Focusing on GUI ripping and mobile Model based testing	
English	

TABLE 3.1: Inclusion/Exclusion Criteria

After reading the resulted studies, we categorize them into 2 categories, each category is explained in more details in the below sections.

3.3   Automatic Model-Based Testing

Generally, GUI testing is an important activity aims to detect faults in the application interfaces, which may lead to error in the application. The testing process for applications with graphical user interface is know to be complex because it is usually hard to predict the human interaction with the application and because of the infinite number of possible event sequences.Different testing approaches were used to handle the testing process. Model-based testing is one of the techniques used for GUI testing, it is about automatic generation of test cases from the model of the application under test without relying on the traditional testing

techniques like manual scripting or capture and reply. Model-based testing has the ability to accomplish the testing tasks in a more efficient and cheaper way if compared with traditional testing techniques.

Manual construction of the application model is tedious and error-prone. Moreover, depending on automatic generation methods considered a challenging process. Many studies and tools were proposed as automatic model-based testing tools, that help in building the model dynamically by exploring the execution environment of AUT.

Ibrahim and his colleagues in their study [21] proposed a hybrid technique to support the reverse engineering of the GUI model of the mobile application. The basic idea of their technique was to use both static and dynamic analysis; the GUI Information was extracted using the static analysis for the byte-code of the application and then a dynamic crawling was done to reverse engineering the GUI model of the application. Their static analyser took the application APK as an input, started the analysis process to end up with a window transition graph (WTG), which is made of nodes(GUI Widgets) and edges(Events). This graph then entered the dynamic crawler to extract the GUI widgets and its related events to produce the GUI state model as an output. They aimed to clarify the GUI behavior using an effective and high quality model.

Using both approaches (static and dynamic) enhanced the scope and the completeness of reverse engineering process, where it exploited the strengths in each approach. For example, the static analysis based on the idea of extracting accurate and complete information about the application by analyzing its source code or binary code without execution. However, it was difficult to gain a comprehensive information about the behavior of the application GUI. Therefore, the dynamic analysis was done to provide information about the behavior of the application.

They made a prototype called *AMOGA* for their study that used the hybrid approach to generate a model to describe the behaviour of a mobile application. This model can be used to generate test cases to test that application.

In another study that discussed the importance of model-based testing for mobile application is the ORBIT tool[22]. This work is an automated GUI-model generator for mobile applications. The proposed work used the static analysis on the mobile application source code in order to extract the different events and actions supported by each GUI widget. Then, these events were exercised on live using a dynamic crawler to identify the GUI behavior of the application. Identifying the different actions and events in the static analysis involved

three basic steps: (1) Identify where the action is registered or instantiated, (2) Locate the GUI component on which the event is fired, (3) determine the component identifier to help the dynamic analysis in recognizing the component and firing the action. One of the limitations of this approach compared with AMOGA, is that its static analysis less comprehensive. It does not capture menus and dialog and it actually does not take into account the event handlers UI effects and the triggered callbacks. The hybrid approach[21] was evaluated on a real mobile applications and it obtained satisfactory results and the generated model covered most of the application behaviours.

MobiGUITAR [23] is another automatic model-based testing tool that is based on reverse engineered mobile model. Its idea was inspired by the reverse engineering for desktop applications. However, different challenges appeared between the mobile and desktop fields. For example, mobiles are state sensitive, which means that the stateless event-flow graph (EFG) used in desktop is no longer good for mobile application. Moreover, the test adequacy criteria based on EFG is no longer available, instead a criteria that takes the state-based life cycle of Android applications is needed. Therefore, MobiGUITAR was proposed to overcome these challenges. This tool is made of three primary steps: (1):*Ripping* to create the state-machine model by dynamically traverse the application GUI. (2): Test cases and event of sequences *generation* based on the model and the test adequacy criteria. (3): *Execution* to replays the test cases.

Another study was made by Huang and his colleagues [24] about reverse engineering using the static analysis in extracting the different GUI models for Android application. In their study, they represented the analysis results using the IFML (Interaction Flow Modeling Language) to help human in understanding the models and facilitate the modification processes. The study based on the importance of extracting the GUI objects, the interaction events, window transactions and the GUI constraints in order to understand the application behaviour and enhance the model-based testing where more events can be triggered and different input values can be used due to more extracted GUI elements. Below table 3.1 compare their results with MobiGUITRA[23] results. Table indicates that the new study covers and contains more GUI elements compared with MobiGUITAR tool.

Activity	MobiGUITAR Our Tool	View	Event	Transition	Constraint
connectbot.HostListActivity		11	1	–	–
		13	8	8	0
connectbot.PubkeyListActivity		6	3	–	–
		9	2	4	3
goodweather.MainActivity		53	9	–	–
		55	14	13	2
goodweather.SearchActivity		11	4	–	–
		8	3	3	0
k9.Accounts		13	3	–	–
		22	8	10	4
k9.AccountsSetupBasics		14	3	–	–
		10	8	9	3
ringdroid.RingdroidSelectActivity		22	5	–	–
		15	13	10	2
ringdroid.ChooseContactActivity		6	0	–	–
		5	4	3	0

FIGURE 3.1: Comparision between extended IFML models and MobiGUITAR tool

Chuanqi Tao and Jerry Gao [1], discussed the rapid evolution of mobile and wireless technology, which brought new challenges and issues in automatic mobile testing process. One of the biggest issue is the lack of mobile test scripting techniques and tools that can deal with the diversity of mobile test environments and devices. Therefore, they introduced a new tool based on GUI ripping to facilitate the validation of numerous mobile applications. They provided a large-scale automation solution by incorporating different open-source technologies like Appium and Selenium. Their approach can increase the test coverage by allowing the parallel execution of test scripts on multiple mobile devices running on different platforms.

The GUI ripper in the proposed approach automatically analyzed the mobile application using the different random techniques and exploration strategies like Depth First Search(DFS) and Breadth First Search(BSF). GUI ripper is able to extract the different GUI widgets and its properties from the GUI window to end up with generating a GUI tree, which is made of nodes, each represent a GUI window or activity. From this tree, we can generate the Event Flow Graph (EFG) that made of nodes each represent a GUI event that contains the matching GUI widget ID from the GUI tree, and the edges between these nodes represent the "follow" relationship between these nodes. Test cases can be generated based on the EFG, which maximize the degree of application UI coverage.

Below figure 3.2 is a high level infrastructure diagram that describes their proposed approach. As we can see there is a test automation web application where users can upload the mobile (Android) APK file. From this web application, users can view the test execution and the test summary through MAC terminal. After upload the APK file, the test automation server start the ripping , analyzing dependencies and generating the test cases. These test cases

are then passed to the test runner that drive it to the selenium grid. These test scripts are executed using the running Appuim nodes on different mobile devices or emulators that are already registered.
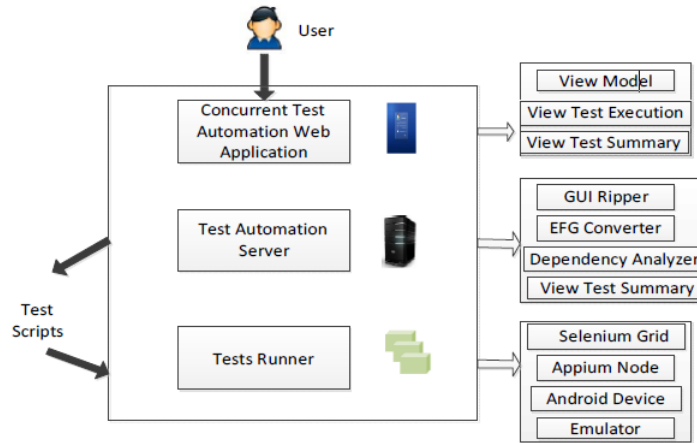

FIGURE 3.2: High level infrastructure diagram [1]

As previously mentioned, mobile development process is full of challenges. One of these challenges is the huge diversity of hardware and software. Moreover, event-driven programming, which is based on user interactions and external events that the program must react to. In addition, the hug development in the mobile frameworks and platforms is considered a new challenge in the development process. These challenges actually affects the mobile development negatively and leads the programmer to write an error-prone code, because of the different considerations that must be taken into account when developing mobile applications. Furthermore, as the competitions between mobile development companies are significantly increasing, developers are supposed to pay more effort to build an application that considers all states of the applications, its context and its external environment. Although, the testing process is developed in recent years, defining new test suits is still a difficult process that requires more efforts, especially that there is no single model that covers the different aspects in mobile applications like the domain, context, usage and the related information in GUI. Thus, the manual creation for each model is time-consuming. Therefore, Santiago and his colleagues in their study [25] proposed a multi-model representation that used the static and dynamic analysis to extract the different information of the application. Their approach helped in automatic extracting of augmented models that combined the different information needed in the testing process, in another words, single model that combined and synthesized relevant information from different models. They evaluated their study on Android applications and they found that the proposed model can be used to generate test cases using the

environmental variables from the context model, generating the inputs for test cases using the domain attributes, and using the GUI information to increase the functionality testing coverage.

Generally speaking, the increasing number of mobile applications with rich GUI causes a growing need to have automatic techniques of GUI testing. A new study made by Gennaro and Imparato [26] to improve and enhance the quality of android testing process and made it more secure and reliable, they combined the GUI ripping technique with the input perturbation testing.

Their basic idea was to explore the GUI application and create a model, then use that model to generate the perturbed text inputs. They discussed one of the challenges in current GUI ripping tools, which is providing test inputs for different application fields without human instructions. This actually affects the code coverage and does not guarantee testing all paths of the application. Therefore, they presented a new tool called *SlumDroid* , which is a modified version of GUI ripper[27], that was responsible for interacting with the user interface and emulating the user interactions and extracting the executable tasks that were stored in the task list. Moreover, it used heuristics to reduce the redundancy in GUI explorations by determining the similarity between current GUI states and the previously visited. The produced GUItree and the captured screenshots from the ripping, entered a separated tool called *GUIAnalyzer* to perform the input perturbation. GUIAnalyzer had the ability to show the different characteristics of GUI input fields like the input type, so it used the regular expression perturbations to generate text inputs for these fields. GUIAnalyzer produced XML format file with a list of input fields and the associated perturbed inputs to be used in the coming test sessions. Their approach was evaluated using case study approach, where they applied it on ten Android applications. The overall result was satisfying and it showed that using the perturbed inputs increased the code coverage between 3% and 70%.

Another automated model-based approach was made by Tianxiao and his partners [28]. Their approach introduced a new and fully automated model-based testing that increased the efficiency of mobile application testing. They got benefits from the testing run-time information to optimized the application model dynamically, which improved and enhanced the model precision and code coverage compared with existing approaches that guide the testing process with static GUI models, which means that the model does not evolve its abstraction during the testing process..

In general, existing testing tools construct the GUI-based model for the application by mapping the different GUI actions to the model actions and the GUI views to the states, where each transition between states is labeled with model action. All existing model-based approach applied the static abstraction/mapping using heuristics rules during the testing process. However, the process of mapping the GUI actions to model actions is a critical and challenging step. On one hand, if the model is overly fine-grained then a state explosion may occurs which will affect the process of exploring the testing model. On the other hand, if the model is overly coarse-grained then the knowledge on model action won't gather sufficiently. Generally, inefficient mapping occurs when mapping multiple GUI actions having different behaviors to the same model action, so the model action don't replayed as expected during the model construction.

The proposed approach is called *APE*. It provided an effective dynamic model mapping based on the run-time information. It refined and evolved the model by looking for the more suitable mapping that balance between the size and precision of the model. This approach if compared with existing techniques, instead of using the static mapping and operating on a fixed granularity, it *dynamically* detect the granularity as required, which caused a higher code coverage and more reliable application. APE dynamic mapping was represented with a decision tree, which improved the testing efficiency.

APE was evaluated on 1316 popular applications and results showed an outperformance in testing coverage and crash detecting process compared with the state of the art Android GUI testing tools. Results showed that APE provides 26-78% more activity coverage, 17%-22% more method coverage and 14%-26% in instruction coverage.

In a study conducted by a group of researchers in China *Stoat* novel guided approach [29] was presented that aimed to test the application functionalities from the GUI model and to validate the different user/system interactions, since existing model-based testing tools coverage is still limited because of the incomplete UI exploration, which cause inefficient testing results

The proposed approach operated in two phases. First, it took the application as an input and generated a stochastic model that described the GUI interactions by employing a dynamic analysis technique and static analysis to explore the different apps behaviours and construct the model. Second, the testing process was guided using Markov Chain Monte Carlo (MCMC) sampling to generate the tests from the model to detect more bugs by traveling on less paths.

Stoat was evaluated on 1661 Android popular applications and results showed a high coverage and effective testing compared with existing testing tools.

Most of the mobile testing techniques generate test cases based on only the different GUI events, without supporting the external events. However, in order to ensure that these applications are working correctly and perform its functionalities properly; one of the recent studies made by Asmau et al [30], proposed a testing approach for mobile applications that took into account the two type of events. The *GUI events*, which were identified using the static analysis of the application bytecode, and the *context events* from analyzing the manifest.xml file.

External events are generated by the operating system as a response to incitement from external sources. For example, receiving SMS or missed call. Android operating system handles the external events using the Intent messaging object, which facilitates the communications between the different components of the application. The permission-based model is used by Android operating system to control the behavior of the application when accessing sensitive data and to notify the user about the dangerous behavior of the device. Generally, each mobile application should declare its permissions by listing them in the manifest file.

A previous study [31] was made by a group of researchers to identify the context events by analyzing the application manifest file without taking into account the source code. Therefore, their approach was not comprehensive enough. However, in this proposed study, the authors considered analyzing the application source code in addition to the application permissions file to provide a more comprehensive information about the context events.

In this apporach, in addition to analyzing the application manifest file, static analysis was used to identify the different application events by analyzing the application byte-code. They used the GATOR toolkit to handle the byte-code analysis process, where the GUI, event handlers and the callback results were analyzed to determine the GUI and system events. The result of analyzing was a Window Transition Graph (WTG), that made of nodes describing the application's windows and activities, and edges describing the events between these windows. This approach was applied on real applications and it showed an increase in the code coverage, which enhanced the application quality.

Static analysis was also used to construct the model in the proposed approach by S.Yang at al [32], where they used static analysis generate a model indicating the behavior of Android's application. They proposed a particular form of GUI models, which is the window

transition graph (WTG) that represents the GUI window sequences and the associated events and callbacks. This model can be used to increase the application understanding and facilitate the testing process. The WTG is made of nodes that represent the different windows in the application, and the edges that represent the transitions triggered by callbacks in the UI. These callbacks can be event-handling callbacks or window-life cycle callbacks.

The good analysis and representation for these callbacks has a critical role in the WTG construction. Moving between Android windows is done using these methods and during the transition additional callback may occurs, which may cause a complicated interleaving between these callbacks. Therefore, the researchers addressed this problem by the help of window stack, which stores the currently alive activities. It captured the additional windows categories, and it modeled the changes to the window stack. For example, single transition in the WTG can have multiple complex effects on the window stacks which are all part of the same WTG edge. Therefore, they gave more attention for careful modeling of the window stack changes and the related callbacks, which helped them in identifying the valid and feasible paths in the WTG. Static analysis was used to construct the WTG of the application. It takes as input all run-time windows, the different widgets in these windows, the different widgets events and their event handler callbacks. Given these input, the construction proceeds in 3 stages. The first stage is constructing the initial edges explained using the trigger event labels. The second stage extended these initial edges to include the push/pop sequences and callback sequences. The final stage is the backward traversal of the graph to determine the correct target nodes of edges.

Their approach was evaluated using a 20 open source android application to determine the overall cost of applying the previous stages and to evaluate the precision to manually constructed model. Results show the effectiveness of their algorithm and approach.

Generally, application has different number of GUI windows that must be adapted to different versions of the mobile, which actually increases the testing process complexity. Many testing approaches and techniques have been proposed to help in testing such cases. Some techniques got benefit from the GUI layout to reduce the obsolete GUI events [?], instead they developed new strategies to guide the path exploration process. However, the exploration process lacks the management criteria. Model-based testing approaches came and actually, it developed a number of exploration techniques like the Depth-First search (DFS), the Breath-First search (BFS) and the hybrid exploration to the model. Despite that, it is still a challenging process

and test the application precisely and completely due to the non-deterministic events, which interrupt the application exploring process, and the state explosion problem that affects the testing performance and increases the complexity. Therefore, Tianxiao Gu el al [?, aim-droid]roposed the *AimDroid* approach, which aimed to improve the model-based technique performance and maximize the coverage while limiting the length of the test sequences and reducing the unnecessary transition between activities.

AimDroid explored the application using multi-level method. It first discovered the unexplored activities using the BFS algorithm. Then, it isolated the discovered activities into a cage and it exploited these activities intensively with a learning reinforcement using fuzzing algorithm to help in discovering new activities. Thus, simplifying the search complexity and manage the consumed time on each activity (collection of widgets).

The proposed approach is made of various components implemented in a client-server style. The server is responsible on guiding the exploration process, while the client is an Android device or and emulator that is connected to the server. AimDroid takes the APK file to end-up with different reports for program diagnoses. The testing process is divided into episodes, each focuses on a single activity to explore. This activity is isolated in a cage, which will block the transitions to any other activity from that activity. Each episode is divided into iterations within a period of time. During each iteration, AimDroid builts the different states and actions for the current GUI, then it selects an action based on the learning module, this action is used to generate the different events on the application.

AimDroid has been evaluated on 50 popular real world application. Results showed the out-performance of this approach in code coverage and in detecting crashes compared with other tools.

Jacinto and Alexandre in their contribution[15] proposed an approach to help in exploratory testing by providing a GUI model of the regions that were affected by the internal code changes.

Their idea came from the gap between the GUI elements and the internally changed elements. This gap produced when testing the application by focusing on unstable test scenarios and examining the change requests of the most recent bugs or improvements. The information gathered from these change request or these test scenarios are not accurate and not enough to determine the affected GUI elements. Therefore, the researchers employed the model-based testing to help reducing this gap. They provided a pruned GUI model by keeping the GUI elements that were related to the internally changed elements in the source code. They used

the static analysis to generate the full GUI model of the application; which was made of different window elements that were connected by event elements as responses from user actions, and then this model was filtered using the reachability analysis to end up with a pruned GUI model.

To keep just the affected regions, they calculated the code differences between the application source code and the previous version of the application from the code repository. This process was the hardest part in their work because it was not just getting the differences like Gif diffs, instead a static analysis was used to detect the new and the modified methods.

Their approach was evaluated on five GUI applications and it showed an increase in code coverage reached 60.40%.

It is true that model-based testing approach has approved its effectiveness in testing the functionality of mobile applications and providing satisfied applications in short interval of time. However, in the context of mobile applications testing, model-based approach still faces problems like the testing coverage inadequacy and the large number of generated test cases, which increases the testing time and management efforts. Therefore, Ahmed and his partner proposed a technique [33] to generate minimal number of functional test cases with a maximum coverage of the mobile application. Their approach helps the developers and testers in providing mobile applications in short interval time with less effort.

The proposed approach was made of four steps. The first step was to *identify the application events*, then *classify these events* and *eliminate the reachability events* to finally *test cases generation step*.

The application events identifications was done by extracting all application events form the XML file that contained the already generated event flow graph. After that, these events were classified to identify the different types of event. The *reachability events*; these events contained the open/close menu and window events, the *system interaction events* that used to interact with the application under test and the *termination events* that were used to terminate the modal windows. The third step in their approach was to eliminate the reachability events because they just change the structure of the application not interact with the application. Thus, eliminating these events reduced the testing time and generated minimum test cases that covered most of the application functionalities. Finally, after finishing the elimination process, the approach started the test case generation process from the event flow graph, where the edges in this model between the different nodes represented the test cases. The proposed technique results were compared with other techniques results by applying the

new approach[33] and the previous approaches to an application called "Create Shape, calculator". Results stated the effectiveness of the proposed approach in generating minimum number of test cases with high coverage reached 94.8%.

3.4 GUI Models Maintenance

Farnaz et al in their approach[34] proposed a technique "GUIFetch" that took the application graphical user interfaces sketch, and from the huge number of open source application in public repositories, it identified the applications that are similar to these provided sketch.

The process involved two phases, the *Analysis Phase* and the *Similarity Computation Phase.* The analysis phase responsibility was to take the different application sketches, the transitions between them and a set of keywords, and then searched over the public open source repositories to find relevant applications. GUIFetch removed duplicates applications or those that likely have similar or identical source code. After that, GUI hierarchies with transitions between the GUIs for the user sketch were generated using a prototyping tools called *pencil*. Subsequently, GUIFetch used the static and dynamic analyses to generate the GUI hierarchies for every possible screen of the related application. The similarity computation phase took the source code of the matched, non-duplicate applications, the GUI hierarchies for these applications' screens and those for the user sketch. It also took the transition graph from the analysis phase. In this phase, GUIFetch computed the overall similarity between each application and the sketch. It first calculated the screen similarity. which meant the similarity between each screen in the application and each screen in the sketch. Then it computed the transition similarity in the application and in the sketch to check if the transition defined by the user in the sketch exists in the application. Using the screen similarity and the transition similarity, GUIFetch computed the overall similarity score by adding these scores.

GUIFetch helped the developers in building their GUI applications and assessed them whether there are existing application similar to what they want to develop. It was implemented for Android application and evaluation showed promising results where it showed it effectiveness in helping developers in designing and developing their applications.

Usually when doing regression test and when application evolve, GUI test scripts fail because of the changes in the application. GUI test scripts refer to the exact sequences of events to be executed, and they are very sensitive to the changes occur in the structure or the application workflow. While it is preferable to repair the current test scripts, it is hard and expensive to do so manually. Therefore, there is a need to have an automatic way to maintain these test

scripts after GUI evolving.

Chatem [35] is an automatic GUI test script maintenance technique proposed by a group of researchers for Android applications. It took the GUI model for the base version of the application, the model for the updated version and a set of test scripts for the base version application, then it automatically extract the changes between the two models based on their corresponding Event Sequence Models (ESM). Finally, it constructs alternative test actions and replace the obsolete test cases with the alternatives. Moreover, new test scripts are fenerated to test the new events and widgets that were added to the GUI screens.

In their approach, they used Gator [32] to construct the initial ESM for the application. However, the generated model may be limited and miss feasible behaviors, so they executed the application to confirm the important behaviors of the ESMs. According to extracting the changes between models, they extracted the changes first at the screen levels, then the widget levels and finally at the connection level.

Sebastian Bauersfeld ensures the importance of having a robust and high quality GUI due to the huge evolving in tablets, smart phones and the heavy reliance on them to achieve our daily lives activities. Testing these GUI applications is still a challenge, where the manual testing is expensive, limited and time-consuming process especially when doing regression test. Therefore, Sebastian proposed a new regression testing tools for GUI applications which is called *GUIDiff* [36]. It compared the GUI states between the two different versions of the application under test to end up with a list of detected differences.

The basic idea behind GUIDiff is to run the two versions of the application in parallel and report the differences between the GUI states to the testers. Therefore, the GUI state information should be captured in widget trees. After that, the two versions of the same application are run side by side to notice the differences between the states in the widget tress. Doing so will compare the properties of the same controls against each other.

In GUIDiff the widget trees need to be aligned to find the identical control. Therefore, the same actions on identical control pairs can be executed which allows the parallel execution of the two versions of the application. Moreover, comparing the different properties of the controls like title, color, style, etc helps in detecting and reporting the differences, so helping the testers in observing these differences.

Such approach has many challenges; the process of tree alignment is expensive and may affect the performance and slow down the test execution. In addition, since the tool is semi-automatic where testers are part of this testing process, it would be exhaustive to them to

label every and each difference between the trees.

3.5 Summary

This chapter illustrated and included different topics and subjects. The search methodology, terms and the inclusion and exclusion were explained in details. Moreover, critical literature review was conducted, about 20 papers have been studied carefully. The problem statement and the proposed approaches for each study have been identified, which covers the field area and strengths our knowledge about this topic.

All studies highlighted the importance of providing a good mobile application with a high quality to satisfy the customers' needs especially with the increase of competing companies in the market. As mentioned before GUI testing is a challenging process because of the complexity of predicting the human behaviors while using the application as well as the large number of event sequences that need to be tested before launching the application. Model based testing is one of the approaches that help in GUI testing and that is actually the base of all studies we have already read. The basic idea of this type of testing is generating test cases relying on models illustrated the application. From our elaborating and critical reading we have notice that all studies were evaluated on Android operating systems which is widely used over the world. However, none of these studies dealt with using model-based testing in *cross-platform* applications, which produced a gap and this is where we will elaborate to fill such gap.

Chapter 4

Methodology

4.1 Overview

As mentioned before, our approach is mainly based on generating a pruned model to facilitate the testing process of React Native applications and after investigating the literature review and provide a comprehensive understanding about model-based testing. In this chapter, the methodology and the different phases of RN-AST Pruning framework will be explained in details.
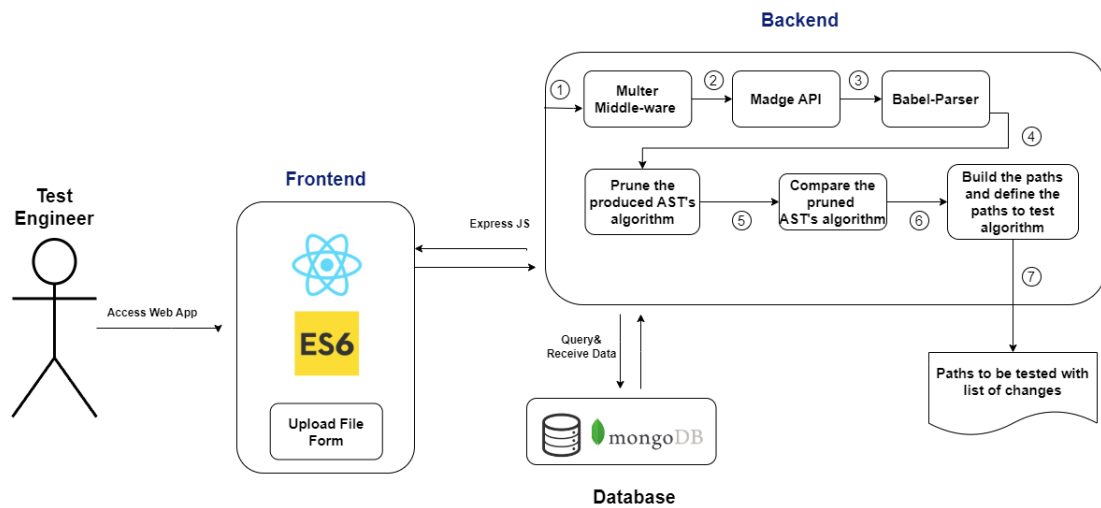
4.2 Solution Approach

FIGURE 4.1: Structure Diagram of the framework

The above figure illustrated the proposed structure for RN-AST pruning framework. As seen, the framework composed of three parties communicate with each other, the test engineer who

uses the front-end interface to connect to the back-end side that sends data to the Mongo Database, which is considered the third party.

The back-end side represents and clarifies briefly the different steps for our framework. First, Multer, which is a Node.js middle ware is used to handle the process of uploading the source code file from the test engineer side. Second, Madge API generates the visual graph of the dependencies of the uploaded source code. This API has different features that facilitate the process of determine the different dependencies between the different modules in the application, finding circular dependencies and providing many useful information. Then, the babel-parser was chosen to parse the uploaded ECMAScript source code. This parser is a JavaScript parser used in Babel compiler and it is heavily based on acorn js parser. The parser produced the abstract syntax tree (AST), which is a tree representation of the abstract syntactic structure of the uploaded source code. After producing the AST, it is time to prune this tree and keep only the GUI elements and this is the responsibility of the pruning algorithm, which will be explained in details in the coming sections. This algorithm will be applied to the produced AST of the uploaded source, which is the original source code and the updated version of that code, which will be saved in certain folder in the device. After pruning the two AST's, the comparison algorithm starts its job by comparing the pruned AST'S to find the GUI elements that differentiate between the two versions of the source code. This algorithm classifies the change between the two versions into three classifications:

- Update: that means same elements but different properties or values.

- Placements: that identifies the new inserted elements in the updated version not in the original version,

- Deletions: that tags the deleted elements which are found in the original source code not in the updated version.

Finally as indicated in step number 7, the paths that contains the changed files are generated and returned to the test engineer who will use them to reduce the testing time and facilitate the testing process.

Below sub-sections describe and explain the different algorithms and steps of our approach in more details.

4.2.1 GUI modeling and representation

As illustrated in the above figure, the test engineer is asked to upload the source code file of the application, which is the starting point for React Native applications, with the use of Multer middle-ware that adds the file object contains the files uploaded via the form to the request object. Then it comes the time of building and modeling the structure of the application by building the component diagram and the dependency graph. This structure, in addition to its importance in helping people in retaining and recalling information longer about the software, it also increases their understanding of the application by providing a general overview about the system. Therefore, everyone has the ability to know the different components of the system, understand the relationships, dependencies between them and the impacts of changing one component on the others. It may also increases the collaborations in improving the system, since providing a visual graph about the system encourages designers and other team members to discuss, find weak spots and enhance the system.

In RN-AST pruning framework, the Madge API -which is a developer tool- was used to generate the **visual** graph of our application dependencies. The Madge API was chosen because it is free, open source, easy to use and it offers the useful needed information like the different dependency paths of our application files and modules as object, the existence of circular dependencies and other useful information.

Madge API takes the uploaded file, produces the dependency graph based on the imports in the file. Then the framework insert the file name and its dependency as an object in the files table in MongoDB and then sends the content of the image as base64 encoding representation to the client side, which on his parts, shows the image of the dependency graph to the test engineer.

Note that the dependencies illustrated between the components in the dependency graph indicate that the functioning of one component depends on the existence of other component. After using Madge API to visualize the dependency graph as shown in figure 4.2, each node in the figure is a source file and each directed edge between these nodes represents the dependency from one file to another. For example, below is a real example of a dependency graph of a registration form build over react native runs over android & iOS. The edges between the files indicates the dependencies between the different files
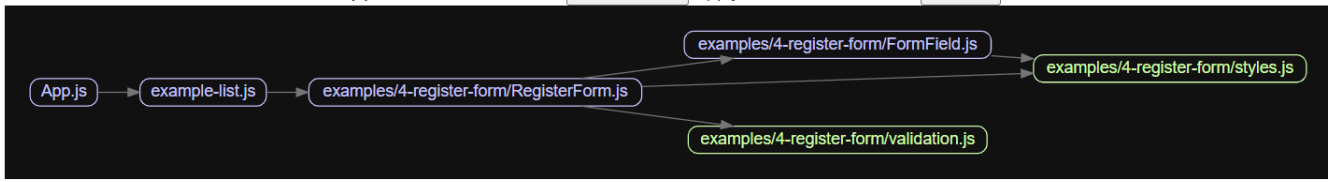
FIGURE 4.2: Dependency Graph Example

4.2.2   Parse the code, detect the JSX elements & prune the AST

By working with parsers, researchers improve the results for various software engineering tasks like: code summarizing bug and malware detection. [37].

Code parsing is the process of breaking up the code sentences or group of words into separate components based on the set of rules and grammars for each programming language [37], where the output of parsing the source code is represented in a tree-like object usually called the *abstract syntax tree.* Parsing plays a key role in the architecture of the compilers, where it considered the first step in compiling process. [38].

Several tools convert source code into a tree structure. In our study, Babel JavaScript Parser was used to produce the AST, do the source code transformations and extract the dependencies of the source code as an object. Babel parser is heavily based on the Acorn and Acorn-jsx parsers and the produced AST conforms to the ESTree specification, which is considered the de-facto community standard for ECMAScript ASTs [39].

Technically, most parsing libraries provides a way to traverse the produced AST that means the ability to visit the different nodes of the tree in order to perform different actions [?]. In RN-AST pruning framework, the chosen parser - Babel parser- offers the Babel-traverse to visit the AST nodes and detect the JSX elements.

NPM was used to install the babel-parser as below 4.3:

```
1  npm install --save-dev @babel/parser
```

FIGURE 4.3: Install Babel-parser

The parser was called in the code to parse the content of the different dependencies files of the uploaded file as below 4.4:

```
1   const babelParser = require("@babel/parser");
2
3   let ast = babelParser.parse(buffer.toString(), {
4     sourceType: "module",
5     plugins: [
6       // enable jsx and js syntax
7       "jsx",
8       "js",
9       ],
10    });
```

FIGURE 4.4: Babel-parser in the code

Both source codes of the original code- which is the uploaded file- and the updated code -which is stored in specific folder in the device- are being parsed using the Babel-parser to produce the original AST and the updated AST. These AST's are pruned to keep only the exported JSX elements for each source code. These JSX elements are the elements that are shown on the user interface of the application. Doing so helps in finding the GUI changes between the two versions of the code.

The pseudo-code for the proposed algorithm for pruning the AST of the source code answers the first question in the research questions section 1.4 and it can be described as shown below:

Pruning algorithm notes: This study based on the idea of using static analysis to parse the JS source codes and check if their exist any changes in the exported JSX elements between the two versions of the source code.

Since the developers using React Native can write JS ode in different patterns and styles, there exists several ways to write the *export*,which is used when we create a JavaScript module and want to export their functions, primitive values or objects to be used in other modules using the import declarations[40]. The basic idea behind imports and exports is to exchange contents between several JavaScript files and to help in splitting the code into multiple files, so achieving the modularity in our design.

There are two type of exporting in React native applications:

- Named Export (Zero or more exports per module) that are used when building a component or components that are imported many times.

36

- Default Export (One per module) that exports a single class, function or primitive from a script file.

RN-AST pruning framework addresses the Default export type that are created by including a *default* tag in the export. Using default export allow the developer to have only one export in the file.

In general, there exists many forms of using "default export" in the modules based on the type of the exported components.

Basically, in the world of React Native, components can be written using functions or classes. Just like their names indicate, a functional component is just a plain JavaScript function that returns JSX. However, a class component is a JavaScript class that extends React.Component which has a render method.

Below are screenshots illustrated the different expressions of exporting in react native that cover the different components types.

```
1  function App() {
2      // function body
3  }
4  export default App;
```

FIGURE 4.5: Export Function

```
1  const App = () => {
2      // body
3  }
4  export default App;
```

FIGURE 4.6: Export Variable

```
1  class App extends Component {
2      // body
3  }
4  export default App;
```

FIGURE 4.7: Export Class

Another expression is to declare function, class components directly in the export default like below:

```
1  export default class App extends Component {
2   render() {
3    // render body
4    }
5  }
```

FIGURE 4.8: Export Class Directly

Functional components can be written in two ways:

regular functions, which can be written as:

```
1  export default function App() {
2      // body
3  }
```

FIGURE 4.9: Export Regular Class Directly

Or arrow functions as below:

```
1  export default App = () => {
2      // body
3  }
```

FIGURE 4.10: Export Arrow Class Directly

Each pattern of these patterns produced different AST structure. Our algorithm detects JSX elements of all above patterns used by the developers.

Algorithm name: AST Pruning Algorithm

Input: The AST of the source code as array (produced by babel parser)

Output: Pruned AST only with JSX elements shown on the screen returned as array.

Algorithm Steps:

1. Get the Abstract Syntax Tree of the uploaded source code from the babel parser.

2. Use babel-traverser to traverse the AST nodes and especially the **ExportDefaultDeclaration** node to check the type of default export.

38

3. Get the first rendered GUI element on the screen based on the used export default pattern:

   (a) When export function as default export after the function declaration, then the algorithm get the name of the exported function and traverse all the **FunctionDecalaration nodes** until the name of the function in the node matches the name of the exported function. Then the first rendered node is the first JSX element stored in the **ReturnStatement** node in the body of the FunctionDeclaration node.

   (b) When export variable as default export after the variable declaration, the steps of getting the first rendered JSX element are as above steps. However, instead of traversing the FunctionDeclaration nodes, the algorithm traverses the **VariableDeclaration** nodes, check if the variable name matches the exported, then get the first JSX element from the **ReturnStatement** node in the body of the matched VariableDeclaration node.

   (c) When export class as default export after the class declaration, the steps of getting the first rendered JSX element are as above steps. However, instead of traversing the FunctionDeclaration or VariableDeclaration nodes, the algorithm traverses the **ClassDeclaration** nodes, get the different class methods, then get the first JSX element from the **ReturnStatement** node from the **render** method.

   (d) When export **regular syntax** function as default export, the first JSX element would be from the **ReturnStatement** node from the body of the **FunctionDeclaration** node.

   (e) However, when export **arrow syntax** function as default export, the algorithm get the **ReturnStatement** node as the first JSX element from the body of the arrow function.

Note: JSX element is syntax extension to JavaScript with the purpose of designing a more concise and easy-to-understand syntax that describes what UI element should look like.

4.2.3 Comparing the JSXElements tree of the original and updated source code

After producing the pruned ASTs of the original source code and the updated version of the source code, it's time to answer the second question 1.4 and compare these two ASTs to find

the set of differences and this is the aim of this phase.

Generally, React is considered one of the fastest JavaScript frameworks. Reconciliation algorithm is considered one of the several reasons for this distinction. This algorithm deals with figuring out how to update the UI of the application effectively and without any delays, by optimizing the process of comparing the current DOM tree and the virtual DOM tree -that contains the new DOM tree with new states and props- to determine which parts need to be changed on the actual DOM.

In our proposed algorithm the strategies followed in the diffing algorithm in React (reconciliation) were taken into consideration while comparing the two pruned ASTs.

As mentioned in the previous step, the Babel-parser was used to produced the AST of the uploaded source codes, and by following the pruning algorithm steps, we end up with two pruned ASTs.Below is the pseudo code for the proposed algorithm of comparing the two pruned ASTs.

Algorithm name: Diffing Algorithm

Input: The pruned AST of the original source code (old AST) and the pruned AST of the updated source code (new AST).

Output: The deletion elements array, the placement elements array and the updated elements array.

Algorithm steps:

1. If the old AST and new AST are empty then return empty arrays

2. If the old AST is empty then push the new AST to the placement array and return it

3. If the new AST is empty then push the old AST to the deletion array and return it

4. Iterate at the same time over the two pruned ASTs starting from the topmost element in the AST.

5. The algorithm generates an id for each exported element -if not exists one- then compare the ids of the element. The elements of the new id's are added to the placements array, while elements with the deleted id's are added to the deletion arrays.

6. Accroding to the elements with similar id's, the algorithm always starts comparing them

(a) If the old element and the new element have the same type, the algorithm looks for the proprieties and the attributes and use Lodash library to do the comparison, and if any difference exists, the old element is pushed to the updated array.

(b) If both have different types and there is a new element, this means pushing the new element to the placement array, which contains the new created elements. In case there is an old elements, this element is pushed to the deletion array that has the elements to be deleted.

By default, when recursing on the original elements and the updated elements, the algorithm just iterates over both lists of elements at the same time and check for differences. For example, when adding an element at the end of specific view, finding differences between the original tree and the updated tree works perfectly.

```
import React from 'react';
import { Text, View } from 'react-native';

const YourApp = () => {
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems: "center"
}}>
      <Text>
        Try editing me! 🎉
      </Text>
    </View>
  );
}

export default YourApp;
```

FIGURE 4.11: Original View

```
import React from 'react';
import { Text, View } from 'react-native';

const YourApp = () => {
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems: "center"
}}>
      <Text>
        Try editing me! 🎉
      </Text>
      <Text>
        This is the new added element|
      </Text>
    </View>
  );
}

export default YourApp;
```

FIGURE 4.12: Updated View

The algorithm will match the two <Text> Try editing me! </Text> and then insert the <Text>This is the new added element</Text> element.

However, inserting a new element in the beginning of the view as figure below 4.13 cause a performance issue, where the algorithm won't realize that the <Text>Try editing me!</Text> element has only changed its place. Therefore, to solve this problem the algorithm supports the **id** attribute. which is unique per element, where it checks the id of the two elements, if they differ, the algorithm checks if the original id exist in the original tree, which is usually not hard process and based on that it move toward other elements in the updated tree and the original tree.

Checking the id attributes enhance the comparing algorithm performance and provide the test engineers with a more clear results.

Note that the algorithm generates an id attribute for elements that has not an id. In order to generate the id, it hashes the type of the element and the index. Therefore, ends with an element with unique id among its siblings.

```
import React from 'react';
import { Text, View } from 'react-native';

const YourApp = () => {
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems: "center"
}}>
      <Text>
        This is the new added element
      </Text>
      <Text>
        Try editing me! 🎉
      </Text>
    </View>
  );
}

export default YourApp;
```

FIGURE 4.13: Updated View

Note that the proposed framework is able to detect the following changes:

- Adding new elements to the screen

- Removing elements from the screen

- Updating the text or proprieties of any element in the screen

4.2.4 Building the different paths of the dependency graph

As mentioned before, the dependency graph aims to provide a general overview about the system and the dependencies between the different files and modules in a clear and uncomplicated manner.

Technically, dependency graph is a collection of entities called nodes; in our case these nodes represent the different files in our system. Generally, nodes are connected by edges that manage the relationship between them [41]. Going through these nodes produces the different paths of the dependency tree starting from the first node, which is the root, and ending with leaves, which are the nodes with no dependencies.

In RN-AST pruning algorithm, the idea of getting the different paths of the dependency graph based on start traversing the graph using the *depth-first-search (DFS)* technique. The DFS algorithm starts at the root node and explores as far as possible along each branch before backtracking to the parent.

In order to show the list of paths and the different changes of each node in the path, we have created a tree component, where each tree path represents the sequence of nodes(files) to be tested by the testers of the application. Each node has a sub nodes that classify the changes

into three types. If the file has changes between the two versions of the application it will be colored with red to facilitate the process of tracking changes. Below 4.14 is an example of the list of files to be tested because of a change in the red colored file.

Path6

App.js

example-list.js

examples/4-register-form/RegisterFormTabs.js

examples/2-login-screen/FacebookLogin.js

FIGURE 4.14: Path to be tested

In the front-side and to show the changes in the files as categories, the framework used the **React D3 Tree** component to represent hierarchical data. It is expected to pass a JSON object with specific format to build this tree. Below figure 4.15 illustrated an example of the passed JSON object to this component, where the component takes the name and children of each node.

```
Image written to C:\Users\rec\React-Native-Applications\upload-main\backend\image.s
{
  key: 'Path6',
  nodes: [
    { key: 0, name: 'App.js', color: 'black', children: [Array] },
    {
      key: 0,
      name: 'example-list.js',
      color: 'black',
      children: [Array]
    },
    {
      key: 0,
      name: 'examples/4-register-form/RegisterFormTabs.js',
      color: 'black',
      children: [Array]
    },
    {
      key: 0,
      name: 'examples/2-login-screen/FacebookLogin.js',
      color: 'red',
      children: [Array]
    }
  ]
}
```

FIGURE 4.15: Data passed to React D3 Tree

After passing the data to the component, it start building the tree. Figure 4.16 shows the tree of the updated elements in a changed file after passing the changes to the React-d3-tree component
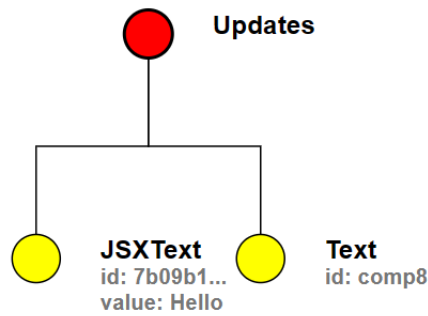
44

FIGURE 4.16: Path to be tested

4.3 Code Implementation Strategies

4.3.1 Modularization

A design pattern concern in splitting the code into multiple files (modules) to facilitate the maintaining process, increase the code reusability and to organize the code, which provide scalable applications. Each module has functions or classes that handle a specific functionality. The functions in one module can be imported and called in other modules, which reduce the code redundancy and make it easier to add new features or edit and remove the existing features.

In the proposed framework, the functions that represent the different algorithms were separated in modules that were imported in the main source code to achieve the modularity pattern. Modularity pattern helps in implementing a scalable Nodejs project by building a reusable, easy to maintain and test modules.

As shown in the figure below 4.17, the different modules of our proposed framework were put in a folder called functions. These functions were exported using the `module.exports` which makes this function accessible outside of the module wherever we require that module. To use this module, the `require()` were used to import it in another module. For example, in `pruneAst.js` module, the `detectJSXelements.js` was imported using the require as below 4.18
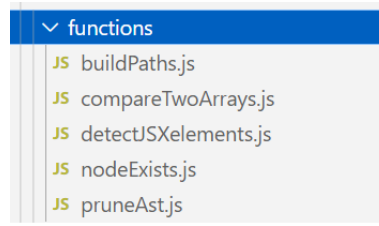
FIGURE 4.17

FIGURE 4.18

4.4   Evaluation

In order to understand the extend by which the proposed framework help and guide testers while testing React Native applications,a user evaluation was conducted to evaluate the user experience and acceptance and measure the effectiveness of the framework in detecting and finding the UI differences between the original and updated source codes. The details of this user evaluation follows.

4.4.1   Application Under Test

In general, the application under test is meant to validate that the aim of the proposed framework is feasible, viable and applicable in practice. Particularly, in this study, it aims to ensure that the framework achieves its goals in detecting the differences between the original source code and the updated version of the code. Moreover, provides the test engineers with a list of paths that may be affected by these changes.

The application is developed using expo framework, which is a set of tools and services built for React Native that allows us to build a natively-rendered mobile applications to run on iOS and Android, so single code base for different platforms.

In order to run the application, all you need is to run the following command, which will start a development server for you :

```
npx expo start
```

This command will generate a QR-code, so to run the application on Android or iOS, the Expo Go application need to be installed on the Android or iOS phone and then connect the phone to the same wireless network as your computer. Use the Expo Go application to scan the QR code from the terminal to open the application under test. Expo GO facilitate the process of running the application on an Android or iOS devices without the need to write additional code snippets.

Basically, React Native has a number of essential and ready-to-use core components to start building the application. Below tree diagram 4.19 illustrated the common core components that are covered by the proof of concept application we have implemented.
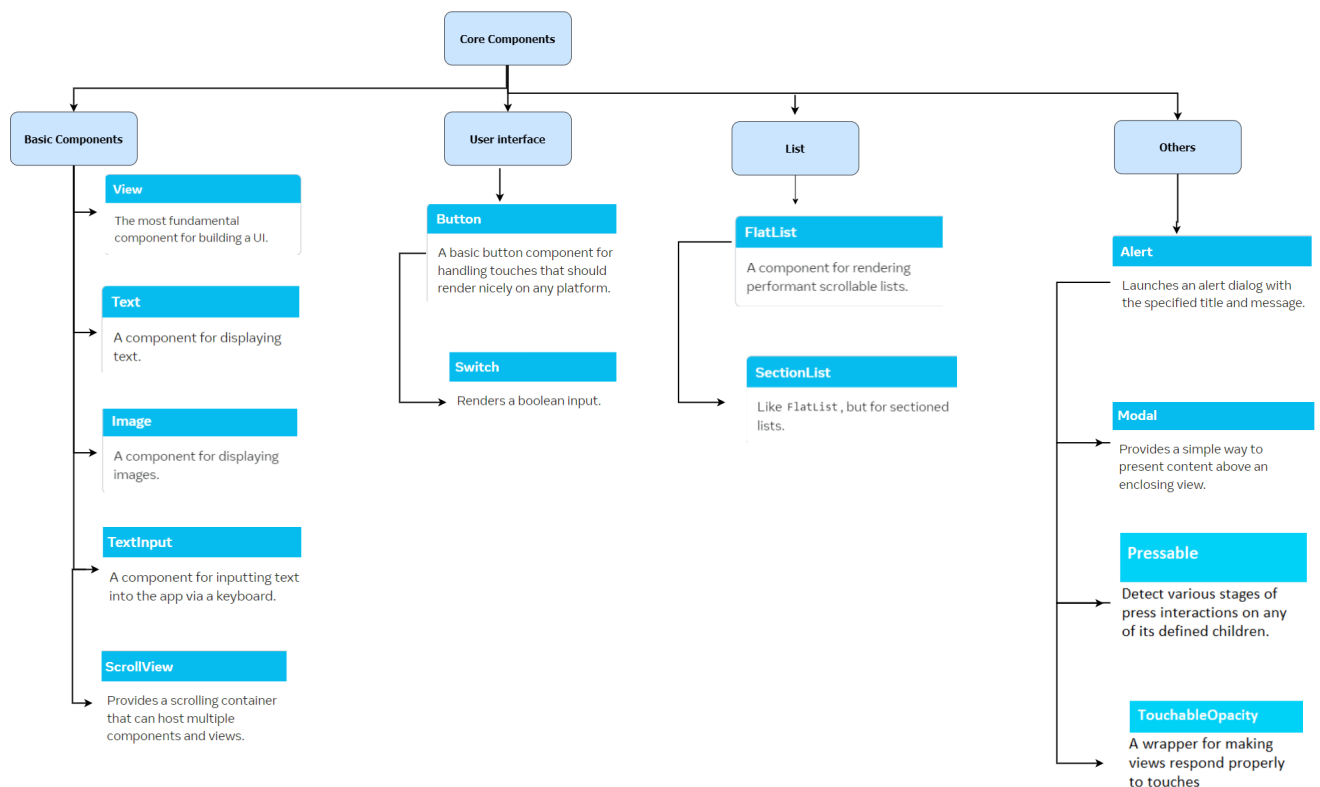

FIGURE 4.19: Covered core components

The developed application is made of a list of pages, each page consists of one or more core components. These pages are treated as modules. Thus, importing one module in another module produces the dependency between these modules. Changing any of these files may affect the main file and this is what the proposed framework is going to detect.

Below is a set of screenshots from the application

Select one page from the list

Level 1   **Hello World**

Level 2   **Facebook Login Screen**

Level 3   **Resturant Menu**

Level 4   **Register Form**

Level 8   **Stop Watch**

Level 9   **BMI Calculator**

Level 11   **Worldwide News**

FIGURE 4.20: Main page of the application
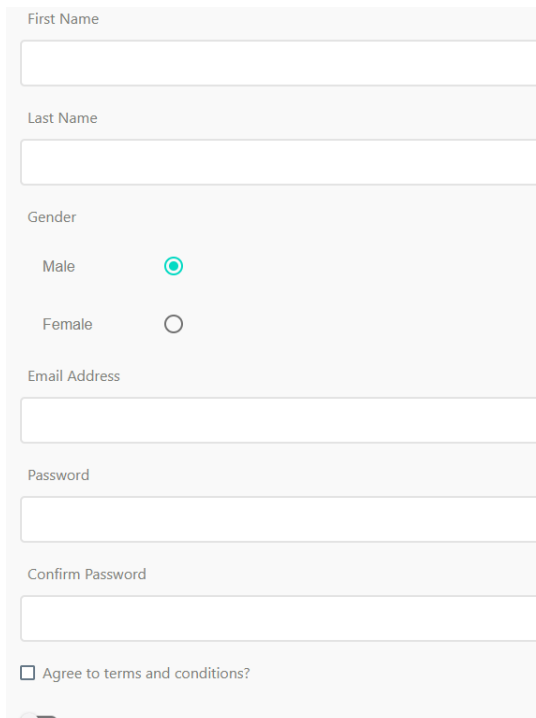
FIGURE 4.21: Facebook Page
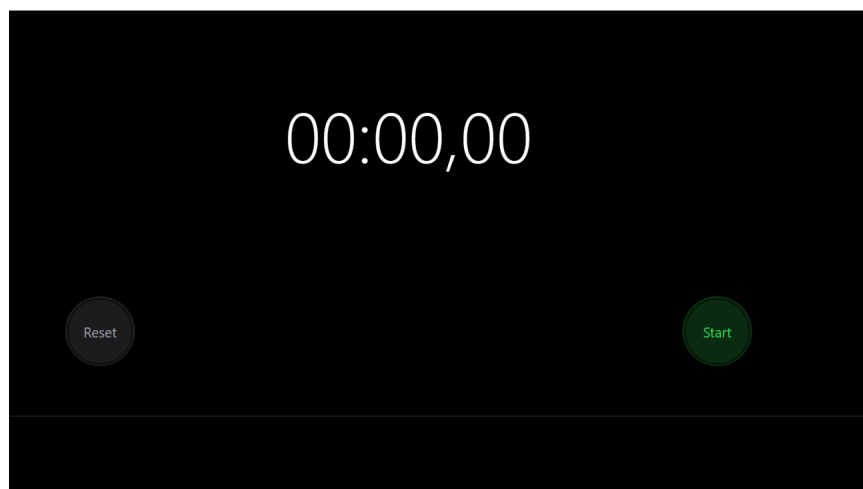
48

FIGURE 4.22: Registration Form Page

FIGURE 4.23: Stop Watch Page

FIGURE 4.24: BMI Page

FIGURE 4.25: World wide news

4.4.2 Experiment Setup and Procedure

This chapter describes in details how the experiment was done, the procedure followed to collect data and people participated in the study.

Participants

Basically, the evaluation of the proposed framework was done with the help of four volunteers that are divided into employees and students. Two of them with a strong React Native

programming background and the others are beginner developers in React native, but a comprehensive tutorial was given to them in order to teach them the basics and increase their knowledge in using and programming with React Native.

Participants were asked to fill a similar questionnaire. The first section in the questionnaire aims to collects data about their qualifications and experience in mobile development. Below table illustrates the general characteristics of the participants as indicated from the collected data.

What is the highest qualification you have?	How many years experience do you have in mobile app development?	How familiar are you with React Native framework?	How many mobile applications do you build using React Native?	How would you rate yourself in building a user-friendly mobile application?
Master Degree	More than 6 years	Familiar	1-3 applications	Good
Bachelor Degree	1-3 years	quite Familiar	1-3 applications	Very Good
Bachelor Degree	1-3 years	Familiar	1-3 applications	Good
Bachelor Degree	1-3 years	Not Familiar	None	Very Good

TABLE 4.1: Participants Characteristics

In general, participants interact with the RN-AST pruning framework using a web interface implemented using React Library. Results and Errors that were generated on the server were sent to the client and displayed to the user.

The proposed framework was build as a web tool to facilitate the process of getting started. It was uploaded to the GitHub repository to simplify the access to the tool from anywhere and anytime. All you need is to install the dependencies and then just get started.

The output sent between the server and the client side was implemented as a JSON object as many web applications use this format for data transmission.

```
setTimeout(() => {
    // Object.assign to append to the object
    jsonObj = Object.assign(jsonObj, { 'treeElements': finalPathsArray });
    res.json(jsonObj);
}, 4000);
```

FIGURE 4.26: Assign JSON Object

Experiment Steps

Below are the steps that were followed to help in evaluating RN-AST pruning framework.

1. The participants were introduced to the framework by reading an instructional tutorial document [1] to reduced the bias between the different participants, to demonstrate the main aim of this framework and the steps of using it with an explanation of the framework's outputs.

2. Multiple online sessions were done with the participants to introduce them with the tool.

3. On completion the learning step, the participants were asked to do some GUI changes to application under use 4.4.1, these changes includes adding new element, deleting or updating exist elements. Note that : two of the participants uses their own computers to access the application and the others uses my computer to do the changes on the original source code.

4. The changes done by the participants were run to ensure that there were no run-time issues

5. Their changes were cloned and copied to the updatedFiles folder in the proposed framework.

6. RN-AST pruning framework starts its job by calling the Madge API to build the dependency graph of the original source code, produce the AST by parsing the different versions of the source code using Babel parser, prune the AST to keep only the GUI expressions and finally compare the pruned AST's to check if the file has changes before building the paths that contain the files with changes.

7. Upon finishing the experiment and showing the results to the participants and making sure that the tool detects their changes, a matching questionnaire was filled with each participants.

Participants Experiments

As mentioned above the participants were asked to do some GUI changes on the application under test in order to test the core components in React Native. Below are some of the changes that were done by the participants.

---

[1]https://github.com/randibrahim/AST-Pruning/blob/main/InstructionalTutorial.pptx

Participants #1 #2

The first and second participants were introduced to the framework and read the instructional tutorial to get the basic idea of the application under test and RN-AST Framework. They used my own laptop to do some code changes on the application. Here are a list of some of their changes on the code

- Add a new **Text** element nested inside **TouchableOpacity** element.

- Update the value attribute of a **Text** element.

- Delete a **View** element, which has different elements inside.

- Remove the animation type attribute for **Modal** element.

- Change the color attribute of a **Button** element and the disabled attribute of a **TextInput** element and a **Switch** element

- Add a new **image** element

Element	Covered
Text	✓
Switch	✓
Image	✓
View	✓
TouchableOpacity	✓
Button	✓
TextInput	✓

TABLE 4.2: Covered Elements by participants # 1 #2

Participant #3

The third participant was introduced to the framework online. The application under test was uploaded to the GitHub platform in a new repository and the participant was able to install the different dependencies of the application under test and run it on his PC.

The participant was asked to do the GUI changes on the code. A new branch [2] was created on the repository that contains the new version of the code. These changes were cloned in my side and the framework was tested on these changes. His changed focused on covering the three categories of changes (updates, placements, deletion). Below are a elements that were covered by this participant.

---

[2]https://github.com/randibrahim/reactNative-App/tree/change-1

Element	Covered
Text	✓
View	✓
TouchableOpacity	✓
Pressable	✓
TextInput	✓

TABLE 4.3: Covered Elements by participant # 3

Participant #4

The forth participant was the most cooperative among them. He was introduced to the framework remotely and he was able to access the application under test using gitHub platform. He run the application on his side and do some changes over the code. Although the poor experience in using react Native, he was able to cover these elements by the changes he did on the code. Note: His changes were uploaded to Google drive [3]. These changes were cloned

Element	Covered
Text	✓
View	✓
TouchableOpacity	✓
Pressable	✓
TextInput	✓

TABLE 4.4: Covered Elements by participant # 4

and tested by the RN-AST Framework to ensure that the changes were detected.

Note that from my own side an extensive test were done to test the tool and ensure that all core components that were not covered by the participants were covered by the test scenarios that I made.

The Questionnaire

In our study, the questionnaire method was used to collect, obtain, summarize useful information from the participants about the proposed framework to support and help the evaluation process.

The questionnaire [4] was made with the help of Google Form and it followed the positive design approach, which was introduced by Sauro et al.**??** that suggests including items with

---

[3]https://drive.google.com/u/0/uc?id=1lUEJIUPazyrU0Oem4NNeefWEFO32CnMhexport=download
[4]https://github.com/randibrahim/AST-Pruning/tree/main/Questionnaire

positive and negative wordings to reduce the responses biases, help analyzing the results faster and avoid accidental errors. The questionnaire questions can be found in appendix A. It has three sections with a total of 16 questions, 9 questions with 5 point Likert scale ranging from Strongly Agree to Strongly Disagree, 3 open-ended questions to capture their opinions of using RN-AST pruning framework in testing React Native Applications, and the remaining questions to capture their background and experience in developing mobile applications.

Chapter 5

Results and Discussion

In this chapter, the results are discussed in depth based on the evaluation chapter 4.4.

5.1 Measurement Results

As mentioned above, the questionnaire method was used to collect data from the participants. Below the results of the questionnaire are presented in an easy way to understand and act on them.

At the end of the data collection process, 4 volunteers participated in the survey. The participants were varied in term of highest qualifications and in their experience in mobile development. Results shows that 25% of them have a master degree and the other 75% have the bachelor degree.
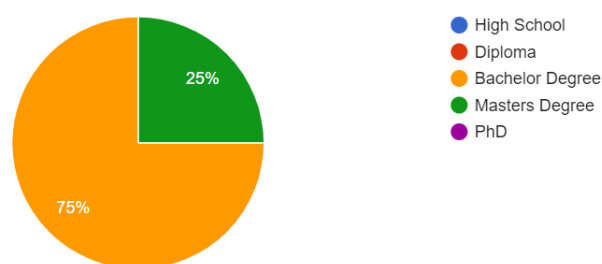

FIGURE 5.1: Highest Qualification

According to their experience in mobile development field, results indicates that only 25% of them are seniors with more than 6 years in this field and 75% are juniors with only 1 to 3 years experience.
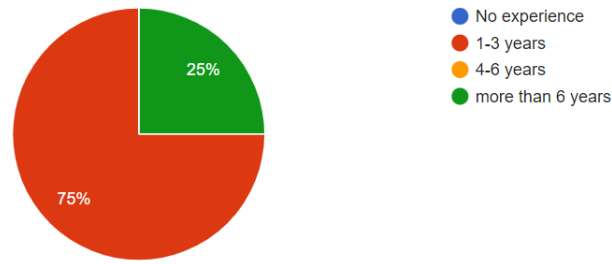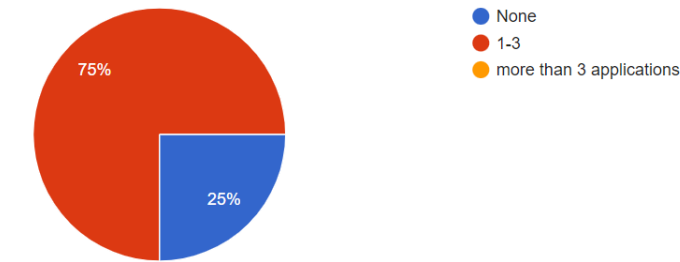
FIGURE 5.2: Mobile Development Experience

FIGURE 5.3: Number of build applications

However, 25% of the participants are not familiar with react native framework and have not build any application using react native, 25% have a little bit experience in using react native framework and 50% heard about it and this can be shown in the bar chart **??**, where 5 means expert and 1 not familiar.
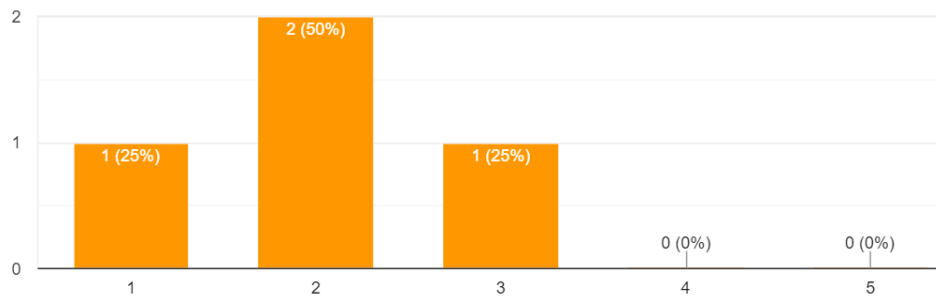

FIGURE 5.4: Familiarity with react native

Since RN-AST Pruning framework aims to detect the GUI changes in react native applications, 50% rated themselves good in building a user-friendly mobile applications and the other half tends to have a very good knowledge in user interface.
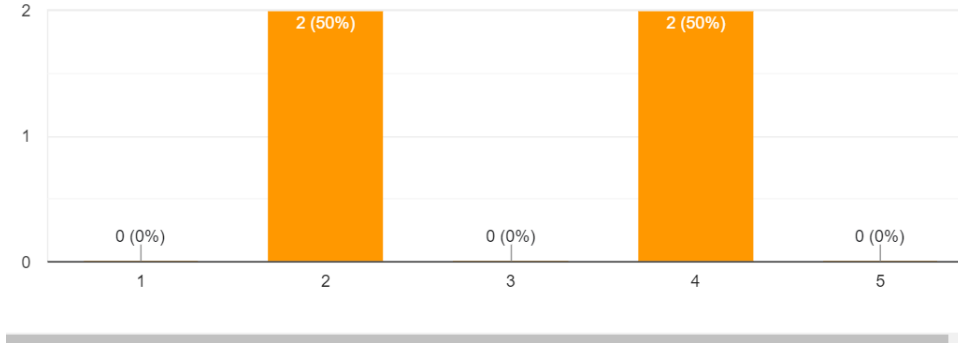
FIGURE 5.5: Build user friendly applications

The second section in the questionnaires aims to evaluate the usability and efficiency of RN-AST Pruning framework. The information gathered from the participants varied greatly in evaluating the usability of the proposed framework. Results were divided into quarters as indicated in the bar chart below, where 5 means totally agree that the framework is easy to learn and 1 means totally disagree5.6.
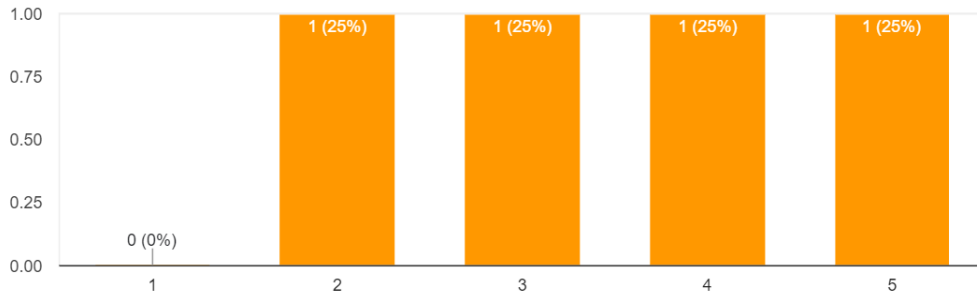

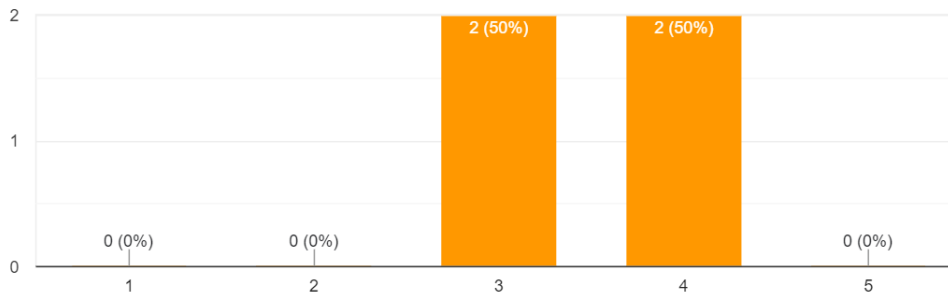
FIGURE 5.6: Easy to learn RN-AST pruning framework

FIGURE 5.7: Easy to use RN-AST pruning framework

Moreover, all participants were neutral in determining the ease of making changes on the original source code as illustrated in figure 5.8
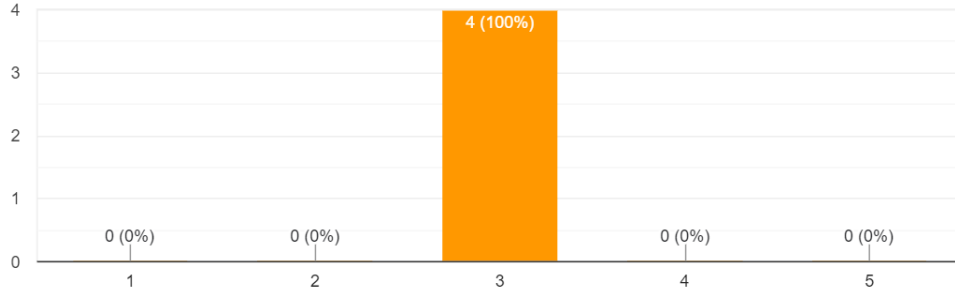
58

FIGURE 5.8: Easy to make changes on original source code

As for the efficiency of RN-AST Pruning framework, almost all participants agreed that the proposed framework was able to provide them with the affected files that needs to be tested as shown in the bar chart 5.9, where 5 in likert scale means totally agree
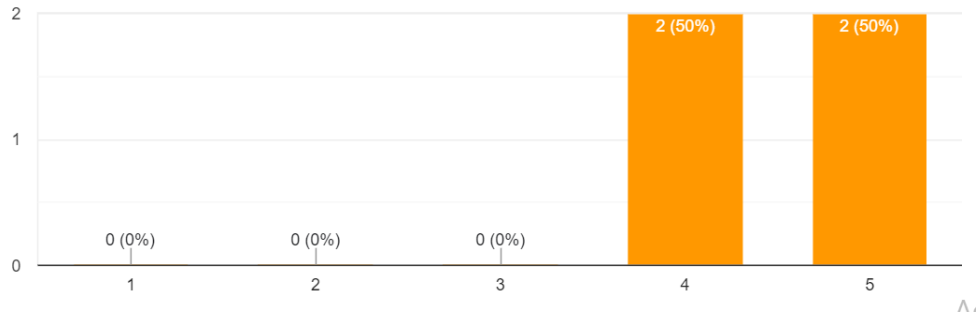
FIGURE 5.9: Provide the affected files

The results provided by RN-AST Pruning framework as illustrated in figure 5.10 were satisfying for about 50% of the participants and the other half of the participants were neutral in determining their satisfaction with the framework results.
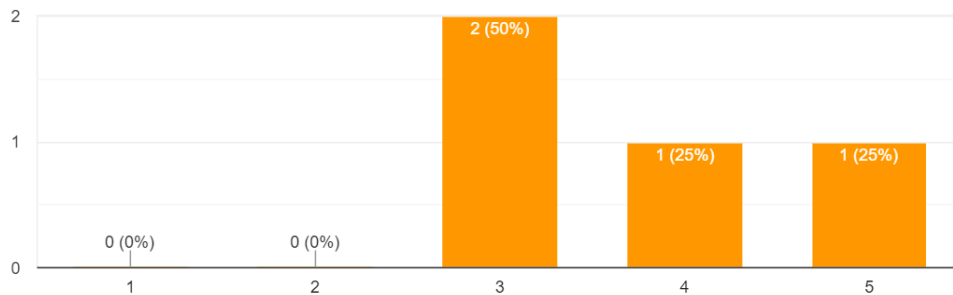

FIGURE 5.10: Satisfy with the results

Despite all, about 75% believe that the idea behind this framework is useful and worthy. The remaining 25% were neutral in this point.
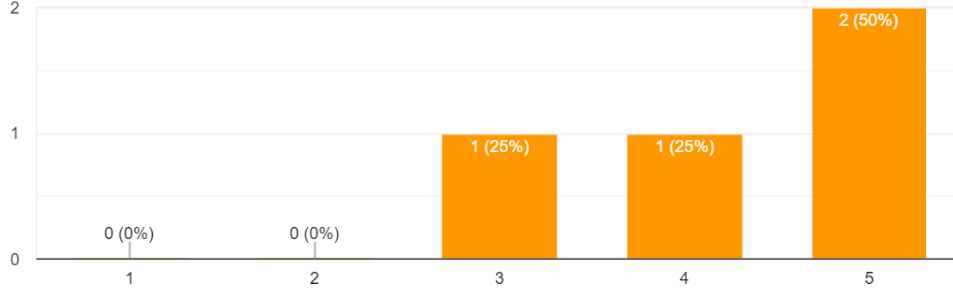
FIGURE 5.11: useful of RN-AST Pruning framework

At the end of the questionnaire, participants were involved in discussion about what may be added to the framework to enhance its efficiency. One of them suggests adding the list of deleted files and new added files -if any-, his suggestion was taken into consideration and the code was modified to reflect his suggestion. This added a new feature to the framework, where in addition to showing the paths of reflected files, it shows the new and deleted files if exists. Moreover, they suggest enhancing the way the changes are illustrated, so the react-d3-tree was used to draw the changed element as a tree. Below are their opinions about the framework:

- *"It helps the computer engineer to identify all the changes that have been made to the source code and also makes it easier for programmers who work in companies to complete the work in less time, less effort and higher efficiency because it explains everything that has been changed on the project they are working on and displays all details"*

- *list of files need testing is easy to have*

5.2   Discussion

5.2.1   How effective is the build framework in detecting changes and results that satisfy the test engineers?

After collecting the results from the participants, below points can be shown:

- All participants agreed that RN-AST Pruning framework provides them with the list of affected files. However, half of them were not satisfied with the way the results are shown, and they found it hard to understand.

- RN-AST Pruning framework idea is useful and can help test engineers by reducing the time and effort required to do the testing process by providing them with the affected files and paths.

From these two consideration the fourth research question 1.4 can be answered positively to indicate the efficiency of RN-AST Pruning framework idea and implementation.

5.2.2 How to detect the GUI elements and prune the GUI model?

According to the first research question presented in the beginning of the study 1.4, RN-AST Pruning framework provides an algorithm that is responsible for parsing the source code and detect the JSX elements. This algorithm is explained in details in 4.2.2, where the basic idea is to use babel parser to parse the react native source code and produce the Abstract syntax tree, then uses the babel-traverse to traverse this tree representation and detect the exported JSX elements that is used to build the user interface.

Applying this algorithm to both versions of the source code produces two pruned abstract syntax trees that are then compared to find the GUI changes between the original source code and the updated source code.

5.2.3 How to calculate and classify the code differences and changes between the last two versions of the application?

RN-AST Pruning framework extends the previous algorithm and provides another algorithm 4.2.3 that is responsible for comparing the two pruned AST to answer the second research question 1.4. The algorithm adapts the heuristics of the reconciliation algorithm followed in react while updating the DOM. When finding the difference, the algorithm classify each difference into one of three types of classifications. The change can be update if the properties or the value of the element is changed between the two versions of the code, placement if the element is new and not exists in previous version of the code, or deleted if the element is not in the new version of the code.

5.2.4 How to build the list of paths that contains the changes file?

Finally, the framework answers the third research question by the algorithm 4.2.4 that traverse the dependency graph using the Depth First Traversal technique and check if each file

has changes before embedding it into the path. The framework also uses the react-d3-tree library to draw the tree hierarchy of the changed files to help test engineers in tracking the changes.

As mentioned above, the idea behind RN-AST Pruning framework is inspired from Reis and Mota approach [15] in aiding the exploratory testing with pruned GUI model to show the impacted regions by the internal code changes. In their approach the authors used static analysis using Soot [1] framework to produce the GUI model of the application that is represented using a direct graph build of small parts such as: *Component, Window and Events.* In order to prune the GUI model and keep only the regions that may be exercised, they used the static analysis to detect the new and modified methods, then they iterated over all events in the GUI model and check if the methods reached by each event belongs to the changed methods, if so they then build the path that arrive at the event from the starting window. While in RN-AST Pruning framework, the babel parser was used to do the static analysis and produce the AST representation of the code that represent the model of the mobile application in our case. The AST is made of nodes, where each node represents a construct occurring in the source code. The babel-traverse is used to iterate over the different nodes in the AST and to build the pruned model that contains only the GUI elements of the application. Since the framework intends to find the impacted regions of the code, then the pruned model for the original source code and the updated version of the source code are compared with each other to determine the updated, new and deleted methods between the two version. Once these methods are known, RN-AST Pruning framework start building and generating the paths from the dependency graph. These paths contains the files that depends on the impacted files and may be affected due to the changes in the source code.

Reis and Mota approach and the RN-AST Pruning framework approach aim to aid the test engineers in the testing process by providing them with the impacted regions due to the changes between the different versions of the code. Therefore, maximize the chances to find bugs and errors within a short period of time.

5.2.5 Threads to Validity

In this section, the factors that may affect the validity of our results will be discussed.

As explained before, RN-AST Pruning framework aims to help test engineers in testing mobile

---

[1]http://soot-oss.github.io/soot/

application written using React Native framework by detecting GUI changes between the original version and the updated version of the source code, and building the affected path of files that need to be tested. Therefore, less test cases to run, less testing time and more efficiency and productivity.

RN-AST Pruning framework has been tested and evaluated by small sample of react native testers and developers due to the lack of people who know react native and able to help us in evaluating the framework although large sample of developers and test engineers were asked to participate in the evaluation process, but only 4 participants were cooperative and helped in testing the framework. Therefore to obtain and get more accurate results, RN-AST Pruning framework has to be tested by a larger sample to cover a large area of GUI changes and more react native components. Moreover, as indicated in 4.4.1 RN-AST Pruning framework covered the core components in React Native, but the increase of using mobile applications and the diversity in application domains may possibly reveal other components to be included in the framework.

One of the feature that can be applied in react native and is not covered by RN-AST Pruning framework is the conditional rendering which means displaying components depending on different conditions. Below figure 5.12 illustrated an example to conditionally render a small block of text.

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

FIGURE 5.12: Conditional Rendering

Moreover, the JSX attribute of type JSX Expression Container is not covered by this framework. This container means put any valid JavaScript expression inside the curly braces in JSX like figure 5.13

```
<Text id="comp9" style={styles.label}>Last Name</Text>
<TextInput id="comp3"
  style={{
    width: 50,
    height: 50,
    backgroundColor: 'red',
  }}
  value="Last Name"
  onChangeText={handleChange}
/>
```

FIGURE 5.13: Conditional Rendering

iOS limitation

As mentioned above, the application under test were build using Expo not React Native CLI due to the obvious strikes that the windows users will face when using the React native CLI, where you need both Android and iOS emulator. This means the as a developer you need to have both Android Studio and XCode to run the application, which are available exclusively on the Mac OS.

In general, the extra configurations required to use React Native CLI, the need to be familiar with Android Studio and XCode and the needs to have Mac OS leads to the decision of using Expo CLI to build the application. When usign EXPO CLI as a developer you only need s recent version of Node.js and a phone or emulator. When run the application, Expo produces a QR code when scanned using an Android device or iOS device the application will be run on these devices smoothly, EXPO CLI facilitate the process of building and testing cross-platform applications.

Chapter 6

Conclusion and Future Work

React Native framework is moving toward stream, it has been increasingly gaining adoption in building mobile applications. Providing robust React Native application become essential especially in light of the great competition between developers and companies in providing competitive applications that satisfy the users needs and expectations. Therefore, the testing process of these applications to ensure its effectiveness is crucial and very important.

] "In general, model-based testing proved its efficiency in testing cross-platform applications. However, because of the increased complexity, model-based technique becomes difficult to conduct. Therefore, in this study, RN-AST Pruning framework was proposed. The basic idea behind this framework is to enhance the testing process, help test engineers by reducing the number of test cases to run, therefore, reduce the required time and effort needed to complete the testing process. The framework works on pruning the abstract syntax tree generated by code static analysis to keep only the GUI elements. Then compare the pruned AST of the original source code with the pruned AST of the updated version of the source code. GUI changes are categorized into updates, placements and deletions. The framework then build the paths that contains the changed files. Each path has a list of changed files or the files that may affected due to the dependency with that file. Thus, reducing the number of tested files and the number of test cases to run.

In previous chapters, cross-platform development and model-based testing were discussed in more details. A comprehensive overview about static analysis and React Native framework were provided. Moreover, the previously proposed and published papers and studies related to our field were summarized and synthesized to give an idea about what has been conducted in the previous decades. Finally, the different phases of RN-AST Pruning framework were explained in more details to help gaining a comprehensive understanding about the framework.

A proof-of-concept mobile application was implemented and presented to be used by a group of mobile application developers. In addition, the framework was evaluated using a case study evaluation conducted on a group of 4 developers with different qualifications and experiences. Participants used RN-AST Pruning framework to detect their changes on the proof-of-concept mobile application and to list them the affected files that need to be tested by the test engineers. Results show that the framework was able to provide them with the changes they have applied to the mobile application code. Moreover, they praised the framework and believe that it is useful in helping test engineers in their testing process.

Due to the lack of time and to improve this work, some next steps need to be conducted in the near future:

- Increase the number of covered components

- Support the conditional rendering

- Enhance the framework interface to more user-friendly theme.

- Evaluate the framework with the help of larger sample of developers and testers.

- Integrate our results with model-based GUI test case generation tools to get more accurate and systematic results.

Appendix A

Questionnaire

About you

1. **Gender**

   ☐ Male

   ☐ Female

2. **What is the highest qualification you have?**

   ☐ High School

   ☐ Diploma

   ☐ Bachelor Degree

   ☐ Master Degree

   ☐ PhD

3. **How many years experience do you have in mobile app development?**

   ☐ No experience

   ☐ 1 - 3 years

   ☐ 4 - 6 years

   ☐ More than 6 years

4. **How familiar are you with React Native framework?**

   Not Familiar ☐—☐—☐—☐—☐ Expert

5. **How many mobile applications do you build using React Native?**

   ☐ None

   ☐ 1 - 3

☐ More than 3

6. **How would you rate yourself in building a user-friendly mobile application?**

   Poor ☐—☐—☐—☐—☐ Excellent

RN-AST pruning Evaluation

7. **It was easy to learn how to use RN-AST pruning tool**

   Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

8. **It was easy to use RN-AST pruning tool**

   Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

9. **It was easy to make changes on the original source code**

   Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

10. **RN-AST tool provides me with the affected files that need to be tested**

    Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

11. **The results provided by RN-AST pruning tool were satisfying and easy to understand**

    Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

12. **RN-AST pruning tool idea is useful for test engineers**

    Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

13. **You would recommend RN-AST pruning tool to a friend**

    Totally Disagree ☐—☐—☐—☐—☐ Totally Agree

Opinions and Future Work Recommendation

14a. **Please specify what you like and dislike the most in using RN-AST tool**

   _____

   _____

   _____

14b. **In your opinion, what needs to be added to the RN-AST pruning tool**

   _____

Bibliography

[1] C. Tao and J. Gao, "On building test automation system for mobile applications using gui ripping.," in *SEKE*, pp. 480–485, 2016.

[2] D. Hartmann, stead, "Cross-platform mobile development."

[3] C. R. Raj and S. B. Tolety, "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," in *2012 Annual IEEE India Conference (INDICON)*, pp. 625–629, IEEE, 2012.

[4] B. Eisenman, *Learning react native: Building native mobile apps with JavaScript.* " O'Reilly Media, Inc.", 2015.

[5] M. Walburg, "Is react native good? advantages and disadvantages." `https://binarapps.com/is-react-native-good-advantages-and-disadvantages/`, 2021.

[6] L. Luo, "Software testing techniques."

[7] J. J. Marciniak, *Encyclopedia of software engineering.*

[8] S. N. L. I. Nader Boushehrinejadmoradi, Vinod Ganapathy, "Testing cross-platform mobile app development frameworks (t)."

[9] X. W. Yepeng Yao, "A distributed, cross-platform automation testing framework for gui-driven applications."

[10] F. D. H. R. M. C. Bayley, Ian, "Mobitest: a cross-platform tool for testing mobile applications,"

[11] X. W. Xue Qin, Hao Zhong, "Testmig: migrating gui test cases from ios to android,"

[12] A. M. Jacinto Reis, "Aiding exploratory testing with pruned gui models,"

[13] mDevelopers, "What is cross-platform development?." `https://mdevelopers.com/blog/what-is-cross-platform-development`, 2021.

[14] M. Utting and B. Legeard, *Practical model-based testing: a tools approach.* Elsevier, 2010.

[15] J. Reis and A. Mota, "Aiding exploratory testing with pruned gui models," *Information Processing Letters*, vol. 133, pp. 49–55, 2018.

[16] A. S. Gillis, "static analysis (static code analysis)." https://whatis.techtarget.com/definition/static-analysis-static-code-analysis, 2020. Accessed: 2022-12-20.

[17] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java Virtual Machine Specification, Java SE 8 Edition.* Dian zi gong ye chu ban she, 2016.

[18] Rahul, "on introduction to static code analysis." https://deepsource.io/blog/introduction-static-code-analysis/, 2020. Accessed: 2022-12-20.

[19] T. Gedeon, K. W. Wong, and M. Lee, *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part III*, vol. 11955. Springer Nature, 2019.

[20] Perforce, "What is dynamic analysis?." https://totalview.io/blog/what-dynamic-analysis/, 2020. Accessed: 2022-12-20.

[21] I. A. Salihu, R. Ibrahim, and A. Mustapha, "A hybrid approach for reverse engineering gui model from android apps for automated testing," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 3-3, pp. 45–49, 2017.

[22] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated gui-model generation of mobile applications," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 250–265, Springer, 2013.

[23] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "Mobiguitar: Automated model-based testing of mobile apps," *IEEE software*, vol. 32, no. 5, pp. 53–59, 2014.

[24] A. Huang, M. Pan, T. Zhang, and X. Li, "Static extraction of ifml models for android apps," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pp. 53–54, 2018.

[25] S. Liñán, L. Bello-Jiménez, M. Arévalo, and M. Linares-Vásquez, "Automated extraction of augmented models for android apps," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 549–553, IEEE, 2018.

[26] G. Imparato, "A combined technique of gui ripping and input perturbation testing for android apps," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 760–762, IEEE, 2015.

[27] A. Memon, I. Banerjee, B. N. Nguyen, and B. Robbins, "The first decade of gui ripping: Extensions, applications, and broader impacts," in *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 11–20, IEEE, 2013.

[28] T. Gu, C. Sun, X. Ma, C. Cao, C. Xu, Y. Yao, Q. Zhang, J. Lu, and Z. Su, "Practical gui testing of android applications via model abstraction and refinement," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 269–280, IEEE, 2019.

[29] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, "Guided, stochastic model-based gui testing of android apps," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pp. 245–256, 2017.

[30] A. Usman, N. Ibrahim, and I. A. Salihu, "Test case generation from android mobile applications focusing on context events," in *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, pp. 25–30, 2018.

[31] K. Song, A.-R. Han, S. Jeong, and S. D. Cha, "Generating various contexts from permissions for testing android applications.," in *SEKE*, pp. 87–92, 2015.

[32] S. Yang, H. Wu, H. Zhang, Y. Wang, C. Swaminathan, D. Yan, and A. Rountev, "Static window transition graphs for android," *Automated Software Engineering*, vol. 25, no. 4, pp. 833–873, 2018.

[33] A. Mateen and K. Abbas, "Optimization of model based functional test case generation for android applications," in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, pp. 90–95, IEEE, 2017.

[34] F. Behrang, S. P. Reiss, and A. Orso, "Guifetch: supporting app design and development through gui search," in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, pp. 236–246, 2018.

[35] N. Chang, L. Wang, Y. Pei, S. K. Mondal, and X. Li, "Change-based test script maintenance for android apps," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 215–225, IEEE, 2018.

[36] S. Bauersfeld, "Guidiff–a regression testing tool for graphical user interfaces," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 499–500, IEEE, 2013.

[37] I. Utkin, E. Spirin, E. Bogomolov, and T. Bryksin, "Evaluating the impact of source code parsers on ml4se models," *arXiv preprint arXiv:2206.08713*, 2022.

[38] S. Setunga, "Everything you need to know about tree data structures." https://medium.com/swlh/writing-a-parser-getting-started-44ba70bb6cc9, note = Accessed: 2022-12-20, 2020.

[39] T. Seckinger, "Compile-time abstraction of javascript mocking libraries powering a domain-specific language for interaction testing,"

[40] R. H. Khan, "Export default in react." https://www.delftstack.com/howto/react/export-default-in-react/, April 2022. (Accessed on 01/14/2023).

[41] TK, "Everything you need to know about tree data structures." https://www.freecodecamp.org/news/all-you-need-to-know-about-tree-data-structures-bceacb85490c/, note = Accessed: 2022-12-20, 2017.