POLITECNICO DI TORINO

Master's Degree Course in Computer Engineering

Master's Degree Thesis

Hybrid Mobile Apps: Development and Testing Challenges

Supervisor
Asst. Prof. Luca Ardito

Correlatore:
Dr. Phd. Riccardo Coppola

Candidate
Sevil Coskun
Student ID: 250910

Academic Year 2018 - 2019

Abstract

Context: The mobile market has grown exponentially in the latest years and multiple platforms are available for the deployment of mobile applications. Hybrid mobile apps, which can run on both Android and IOS devices, are an important solution for developers who aim to release their apps on both markets. Testing mobile applications, and especially their Graphical User Interfaces (GUIs) is one of the most crucial aspects of mobile development and requires an important effort from developers.
Goal: The objective of this thesis is to understand the development frameworks and testing approaches for Hybrid mobile apps, and analyse their main challenges, in the light of a literature review and an exploratory experiment with an existing open-source hybrid app.
Method: Firstly, a Systematic Literature Review has been conducted over various digital libraries. The main research questions of the SLR revolved around hybrid development frameworks, mobile testing frameworks compatible with hybrid apps, and challenges in both of them. Secondly, it involved a Source Repository Analysis, conducted on open-source Android projects, to analyze the diffusion of hybrid applications and the number of tested ones among them. Lastly, an exploratory analysis has been conducted by applying testing tools on a hybrid app that was developed with Phone- gap.
Results: Result of the review shows that many development frameworks and testing tools have been released during the last years: Phonegap is the more adopted a development framework, and Appium is the most used testing tool for open-source projects. Throughout the exploratory study, it has been found that automated testing tools show several crucial issues when applied to real-life applications. More specifically, Appium is perceived as the most adaptable tool among those considered. Robotium, UI Automator, and Selendroid all exposed compatibility issues with cross-platform applications developed with the PhoneGap framework (now available as Cordova).

Conclusion: Hybrid apps can be developed with many frameworks but all of them expose issues that are discussed in the present thesis. According to the development framework selected and the application type, several alternative testing tools can be leveraged. Existing testing tools, even, exhibit many drawbacks and hybrid application testing is proved to be very prone to compatibility issues. The results of the thesis confirm that the fast-growing technology and the diversity of development patterns for Android should be coupled with parallel advances from the testing community to ensure better testability for hybrid mobile applications.

Acknowledgements

The candidate warmly thank her parent(Munevver and Bora) and sister, Deniz, for all love and their encouragement, moral support, personal attention and care. Especially thank Cem to be with her all time and support her when she feeling lost and depressed, takes care her, and love her without any conditions.

Express candidate sincere gratitude to Asst. Prof. Luca Ardito and Asst. Riccardo Coppola, for allowing her to conduct this research under their auspices. She is especially grateful for their confidence and the freedom they gave her to do this work also for the confidential information they have kindly provided by her, and for the useful discussions they have allowed candidate to improve her thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Mobile Devices and Operating Systems

Nowadays mobile phone usage reached very high numbers, according to Statista[**1**] values; in 2019, the number of mobile phone users is forecast to reach 4.68 billion. According to these mobile phone numbers it is obvious that the number of mobile applications increased very sharply. In 2017, 178.1 billion mobile apps and in 2018, 205.4 billion mobile apps were downloaded to consumers connected devices [**2**]. As it seems from the numbers, in one year, the number of downloaded applications increased very rapidly. Recently, the most used mobile phone operating systems are Android which is Linux based operating system[**9**], IOS which is developed by Apple[**9**], Windows which is based on Microsoft operating system[**9**] and they have different mobile applications stores (e.g., Google Play Store, the Apple App Store, and the Windows Phone Store) which offer free and paid mobile applications to users [**3**]. Therefore, it has very big impacts on the software engineers' life because creating a mobile application for different devices with different purposes has some challenges such as time and cost. Hence, there is a big effort to analyze and categorize mobile applications for different user desires. Therefore it is fundamental for researchers to try to find optimal solutions to make developers life easier and increasing the quality of mobile applications for everyone's' point of view [**3**][**4**][**5**].

1.2 Mobile App Development Frameworks

In the high-level perspective, mobile applications could be created with three different development techniques as Web, Native, and Hybrid [**3**][**4**][**6**]. It is an inevitable fact that each different development technique needs different requirements like application design, development platform and language, testing strategies.[**10**] Previously, web-based technologies were popular and accessible from browsers with written by HTML, CSS programming languages [**5**]. Then the smartphone era exploded, and native mobile applications added to web-based versions. Later, a combination of web and native techniques creates a hybrid development technique try to decrease effort for cross-platform Mobile applications.

Web based mobile applications run on the web servers and accessible via mobile web browsers and is highly portable across multiple mobile platforms[**3**]. Users do not need to download any application to their devices. Web application offers interaction with users and websites like gathering information [**7**][**8**]. To create a web-based mobile application, HTML, CSS, JS are basic necessary programming languages and there is no need SDK(Software Development Kit), just some templates and frameworks enough such as Angular, React, Vue[**7**]. To test a web-based mobile application, there are some testing tools such as Selenium, Appium, UIAutomator[**7**]. Web-based mobile applications generally categorized as like the website and they are generally informational and provides additional functionality and interactivity with users, so they do not need to use much functionality of the mobile phones. For example, Wikipedia is a website; it provides information. Facebook is a web app that is more interactive. Web applications have also mobile application version but if a user does not want to download that mobile application, a user can reach a web application version from his mobile phone browser such as Chrome, Safari. Firefox. Therefore, web applications do not need to be downloaded like native mobile applications do [**9**].

Native mobile applications made for specifically smartphones, tablets, and any smart devices with running only a specific operating system[**10**][**11**]. Therefore, it can reach all software and hardware components like GPS, camera, memory, and utilize the full capacity of the device of the mobile hardware, which has better performance because of easy touching hardware [**3**]. It will be supported by the original device operating system. Native mobile application development relies on different development environments, testing tools and programming languages that are original for the device, the operating system. To create a native Android mobile application; some of the necessary programming languages are Java, Kotlin; IDEs are AndroidStudio, Eclipse, DroidScript; testing tools are Selenium, Appium, Espresso[**7**][**20**].On the other hand, to create a native IOS mobile application; some of the necessary programming languages are Objective-C, Swift; IDEs are XCode, Appcode, SublimeText; testing tools are Selenium, Appium, XCTestUI [**5**][**7**][**8**]. As it seems from the information above, there are many different programming languages, IDEs, testing tools. Additionally, creating a native mobile platform application has some challenges about cost, the experience of the developer, and time because software developers need many different languages, tools, and platform to create a mobile application and different tools and background for testing [**6**].

Hybrid mobile applications are a combination of native and web techniques as well [**3**][**5**]. Hybrid uses a common code base to simultaneously deliver like native apps to multiple platforms. These apps can access the hardware features of the device and are also distributed using the platform's app store. Hybrid apps are created using Cross-Platform Tools (CPT) [**4**]. This approach combines native code with an independent code that is suitable for multiple platforms and allows creating cross-platform mobile apps. These applications fit any platform and function identically across different ones. There are some examples of hybrid mobile applications such as Instagram, Uber, Gmail, and Twitter. To create hybrid mobile application C#, HTML, CSS, JS is necessary programming languages and for some example of creating environment PhoneGap, Titanium, Xamarin, Cordova, Iconic then for testing tools are possible such as Selendroid, Appium, Espresso, XCTestUI, Calabash, Robotium, and UIAutmator [**7**][**8**].

For Multi-platform mobile application developments, there are some challenges generally in design and implementation life cycle like compatibility of operating systems features, version differences.[**3**] Cross platforms have four different types as Library, Framework, Platform, and Product/Service to converts source code native binaries.[**6**] There are some drawbacks, such as poor performance, poor native user interface, limited hardware accessibility [**6**]. Additionally, user expectations about the applications are remarkably high. It is a real challenge for the application vendors to provide versatile applications in this competitive market in a short time. The challenge is even more if the application is targeted for multiple platforms[**10**].

1.3 Mobile Application Testing

The quality and reliability of the applications are the key factors that contribute to their success[**12**]. Therefore, testing is one of the most important parts of the software development life cycle. From a high-level point of view, there are functional and nonfunctional testing categories. Functional testing can be defined as activities that validate service functions, integration of the systems behaviors, external system behaviors, and user interfaces[**13**]. Functional testing types can be categorized as; Unit testing, Unit integration, System, System Integration tests in the V model. For mobile apps, unit and unit integration tests focus on white-box tests, which run either on the mobile device or the back-end server[**12**]. System and System integration tests focus on the black-box testing strategy[**14**]. According to Liu Z., Hu Y., and Cai L., a quality model of the mobile application is categorized under eight test elements; Function, Interface, Security, Reliability, Usability, Performance, Resource, Compatibility. Those test elements have sub-test elements which are a more detailed version[**21**]. Furthermore, there many different papers that describe non-functional test categories in many different categories too but in general, types can be categorized as; Performance, Usability, Quality of service, Reliability/Availability, Security, Memory and Energy, GUI, and so on[**13**][**14**]. Generally, those tests are maintained manually means that a human is responsible to conduct tests by hand. However, manual testing of mobile applications may be a very costly activity both in terms of time and resources, as well as an extremely boring, repetitive, and error-prone activity. Not surprisingly, a growing interest in mobile testing automation techniques and tools has been demonstrated by the industry[**15**].

1.4 GUI Testing in Mobile Applications

Recently, almost any software application has a graphical user interface (GUI), in light of that, the user interface is the easiest way for the interacting user and it is the easiest way for users to control software systems by displaying visual widgets, buttons, etc[**16**]. Therefore, GUI has a huge impact on the software, and according to researches, as much as 60% of the lines of code are related to GUI[**17**]. In the functional testing phase, overall system and integration testing of software under test (SUT) involve testing via the GUI testing and the test designer develops test cases, each modeled as a sequence of user events, and executes them on the SUT via the GUI, either manually or automatically. Manual GUI testing is the old and most used technique to assess the behavior of GUIs by a human who directly interacts with GUI with specific events or test cases[**16**]. Events include clicking on buttons, selecting menus and menu items, and interacting with the functionality of the system provided in the GUI[**18**]. In practice, unit and unit integration(white-box) testing are easier to translate as automated testing but system testing(black-box) is more difficult to translate as automated testing due to GUI for the end-users. Hence, some of the failures of the software can only be caught when testing trough GUI, so GUI testing is a very good method to estimate visible failures for the end user's perspective[**19**]. It is obvious that for every kind of testing, GUI testing has some challenges, especially from the perspective of the mobile application. According to P. Aho and T. Vos, GUI testing is slower than traditional unit testing even if it is automated. GUI testing probably limited to the smoke testing set. Especially state space explosion is another challenge for modeling or extracting the GUI models for test cases. Lastly, the complexity of the state of GUI causes difficulties to deduce the state of GUI is correct or not[**19**]. These are some challenges about GUI testing, and for that reason turning GUI testing from manual to automated is hard. Furthermore, manual GUI testing can give a good overview of the system for end-users perspective because manual testing is also executed by a human, not a computer.

Make a connection to what thesis has been continued started with this chapter introduction part. In Second chapter Systematic Literature Review and Mapping will be described in details with answering my research questions. Third Chapter focuses on Source Repository Analysis and Exploratory Study. Last chapter includes results of the work and conclusion.

Chapter 2

Systematic Literature Review(SLR)

This chapter starts with Research Questions which will establish the connection between existing knowledge and problem of this thesis work. Research questions are helpful to identify classification of the papers.

Before starting analysing the papers, research strategy will be explained with identifying used keywords for search, used sources and search strings which will be described in the Table 2.1.

It will be continued with inclusion and exclusion criteria will be explained in order to understand which papers are related or not and then analysis of papers will be discussed according to division of research questions.

Next chapter,analysis of the papers will be discussed by mapping papers with the related research questions. Additionally, statistics about papers distribution will be shown by figures.

Lastly this chapter will be closed by answering all research questions and some statistics about collected paper in conclusion part.

2.1 Research Questions

The work described in this thesis aims at providing answers to following research questions;

RQ1: Which frameworks are available for developing hybrid applications?
RQ2: Which testing tools are available for automatically testing hybrid applications?
RQ3: What are the main challenges associated with hybrid applications?

The questions assemble identifying and characterizing cross-platform or hybrid applications development frameworks, automated testing tools in GUI perspective and challenges of them. RQ1 aims at assessing the development frameworks of hybrid applications. RQ2 aims at gathering information about testing tools for hybrid applications from real world and academic papers. RQ3 is more related about challenges and also advantages and disadvantages on the hybrid applications.

2.2 Research Strategy

2.2.1 Keywords

Mobile, hybrid, GUI, development, testing, challenges

2.2.2 Sources

IEEE
ACM-DL
Springer
Elsevier
ScienceDirect
Google Scholar

2.2.3 Search Strings

Search Engine	Search Strings	# included	# retrieved
IEEE	(mobile OR Android OR IOS) AND (hybrid OR cross-platform) AND (GUI OR graphicaluser-interface) AND (development OR framework OR testing OR automated OR automatedtesting OR challenges)	31	37
ACM	(mobile OR Android OR IOS) AND (hybrid OR cross-platform) AND (GUI OR GraphicalUser-Interface) AND (testing OR automatedtesting OR challenges OR development)	14	26
Springer	(mobile OR Android OR IOS) AND (hybrid OR cross-platform) AND (GUI OR Graphical User Interface) AND (testing OR automated testing OR challenges OR development)	5	15
Elsevier	(mobile OR Android OR IOS) AND (hybrid OR cross-platform) AND (GUI OR Graphical User Interface) AND (testing OR automated testing OR challenges OR development)	3	5
Science Direct	((((mobile OR Android OR IOS) AND (hybrid OR cross-platform)) AND (GUI OR GraphicalUserInterface))) AND (testing OR automatedtesting OR challenges))))	1	1
Others (Google Scholar[1])	(mobile OR Android OR IOS) AND (hybrid OR cross-platform) AND (GUI OR graphicaluser-interface) AND (development OR framework) OR (testing OR automated OR automatedtesting)AND (technique OR approach OR challenge OR limitation)	5	7
Total		59	91

Table 2.1: Search Engine and Search strings

[1]Retrieved papers in Google Scholar Search engine extracted from other search engines

2.2.4 Inclusion and Exclusion Criteria

One of the main key elements of systematic literature review, it should be decided on some selection criteria like inclusion and exclusion criteria to make the analysis more adequate. The purpose of the inclusion and exclusion criteria is to limit the study selection to papers that fit in the proposed topic, i.e., techniques and tools for the automation of GUI testing of hybrid mobile applications and that are available in the scientific literature. To this aim, a set of inclusion criteria useful to identify studies that could be considered in this mapping has been designed.
In details, the following list of *inclusion criteria* has been considered:

1. Studies must be directly related to mobile application development frameworks techniques for hybrid mobile applications.
2. Studies must be focused on testing tools of hybrid mobile applications, including, too, system testing, black-box org grey-box testing, especially GUI testing or any other testing activity aiming at the verification of the functional correctness of the application.
3. Studies must provide a qualitative or a quantitative evaluation or challenges of hybrid mobile applications.
4. Studies must have been published between 2010 and 2019 and the language should be English.
5. Studies have been placed in publications, conference papers, official, and reachable.

In order to filter the set of papers from off-topic ones, the following list of *e*xclusion criteria have been defined:

1. Studies focused on development in general or not directly related to hybrid mobile applications
2. Studies focused on testing in general, and not specifically GUI testing on hybrid mobile application

3. Studies focused on Web-services, cross-platform with related to web-service to native mobile transfers
4. Studies focused on testing in different systems rather than mobile applications
5. Studies focused on other mobile topics not directly related to application especially hybrid
6. Studies focused on surveys, reviews, mapping studies, thesis, and any other kinds of papers
7. Studies focused on private testing tools i.e., developed by a university or company, not found on the internet
8. Studies focused on too high-level definitions about development frameworks, testing techniques
9. Studies focused on another non-functional testing perspective i.e., usability, performance
10. Studies focused on other disciplines or more specific applications, i.e., mechanical engineering mobile applications.

2.3 Data Extraction and Mapping

The purpose of the data extraction is to limit the study of selected papers related to the topic. After applying inclusion and exclusion criteria, 59 papers are selected as related to the topic. However, it is not enough to point papers, it needs more detailed analysis and categorization. Therefore, papers have been categorized under the three categories which are *development, testing* and *challenges.*

Development papers provide details about hybrid application development frameworks.
Testing papers provide details about hybrid applications testing tools.
Challenges papers provide details about hybrid application challenges.

Also there are combinations of two categories in order to emphasize more details about more topics. All included papers have been filed under those categories and mapped with a related category according to their topics. To analyze the development and testing sets, the used frameworks will be listed. The papers categorized as *Challenges*, were used as input to a *grounded theory* procedure in order to list challenges which are collected from the literature.

Grounded Theory focuses on unstructured text to generate theory from data and tries to turn structured text, diagrams, or even quantitative data[**22**]. According to Stol K., Ralph P., and Fitzgerald B., there are 11 basic core features of grounded theory[**23**], however, for this thesis, only three of them is selected which are theoretical sensitivity, coding and memoing.
Theoretical sensitivity which refers to the researcher's ability to conceptualize, and to establish relationships between concepts, lies at the heart of developing grounded theory.
Coding which is the researcher uses inductive and logic to construct analytically codes and infer theoretical categories from the data by labeling 'incidents' and their properties.
Memoing which is The researcher writes memos (e.g. notes, diagrams, sketches) to elaborate categories as they emerge, describe preliminary properties and relationships between categories, and identify gaps, key features are selected.

2.4 Analysis of SLR Result

2.4.1 Context

Analysis of the SLR is based on the research questions, however, before starting to answer those questions; it can be crucial to share some brief information and statistics about related paper analysis. The reason for giving those statistics, they will clarify that some results have unexpectedly bias, even if there is no research bias in the work. Statistics will be given later that are related with which operating system, publication years, their region, and number of articles labeled for each category in order to give brief information about thesis strategy.

The details given in papers are not always enough to consider operating systems of the mobile applications. In some cases, the considered operating system has been mentioned clearly, as it seems from the Figure 2.1 below. In the first place, the mentioned mobile operating systems in the included papers are mostly about Android. Secondly, the most mentioned operating systems are Android and IOS with together, and lastly, there are very few numbers of papers that are related to both Android, IOS, and Windows phones. As a matter of fact, there has not been found any paper which is related to only IOS or Windows phone. Thus, the analysis seems to have a bias about Android phones according to the majority of the papers related to Android.

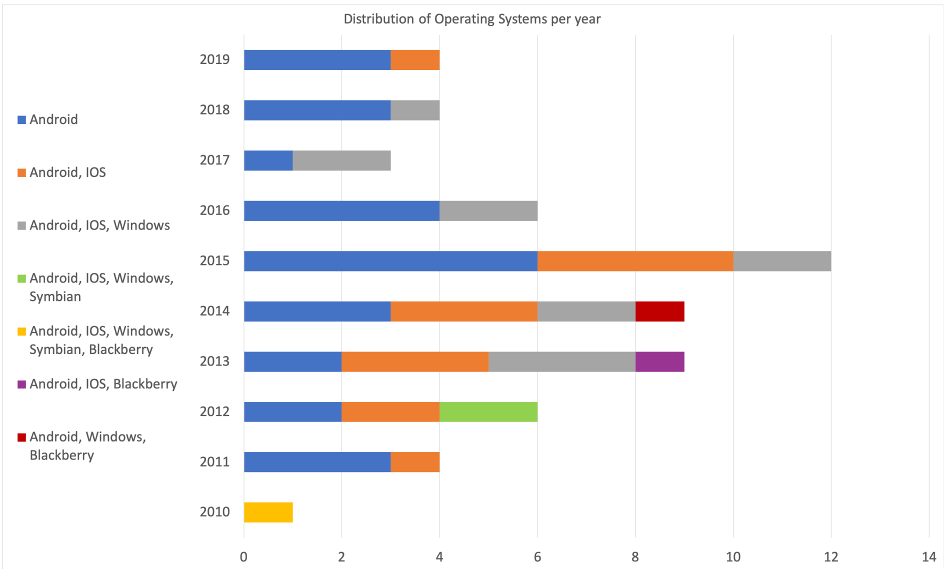


Figure 2.1: Number of papers citing (combinations of) OSs

In the high level of the analysis, we divided the papers into three categories according to research questions. Therefore it is useful to make a graph of the number of citations among the categories and combinations of categories with each other which is mentioned before and it can be seen in Figure 2.2.

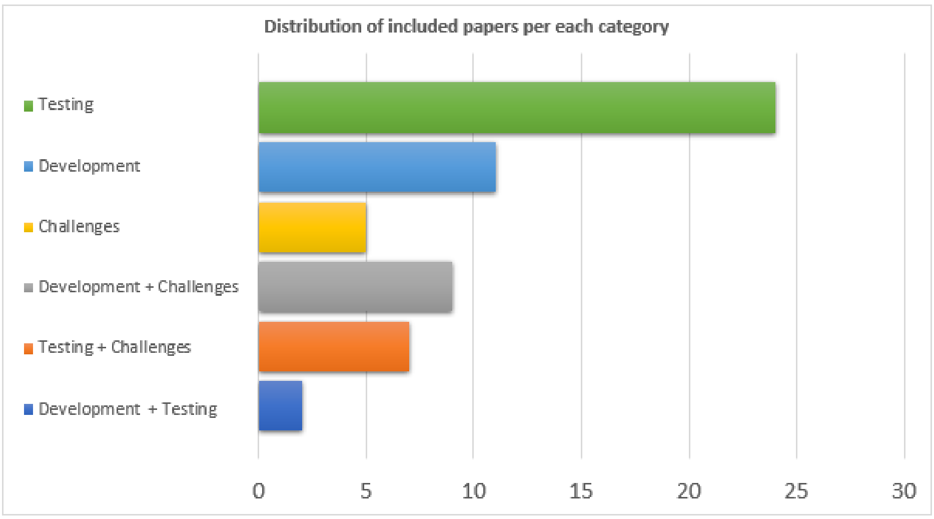


Figure 2.2: Number of selected papers per category

In light of the mapping study made by Tramontana P.,Amalfitano D.,Amatucci N.,Fasolino A. R., included papers appeared mostly after 2010[**15**]. Therefore deciding a year boundary which is 2010 and 2019 on the work could have a valuable effect like selecting more relevant papers. Number of citations by per year is handled in the Figure 2.3 below.

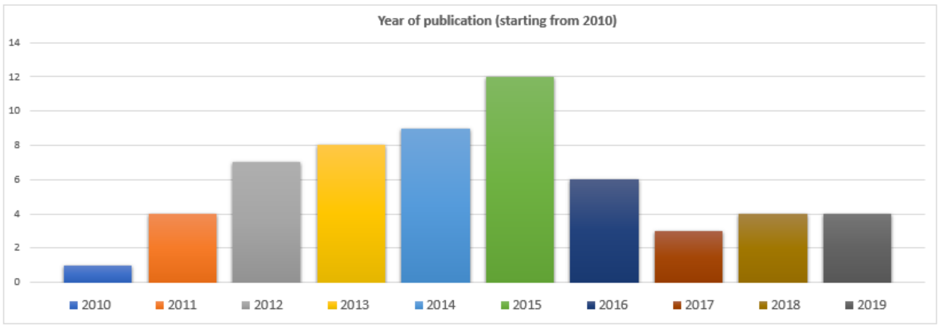


Figure 2.3: Number of papers per year

Moreover, the last statistic is the number of citations by publication countries. Hence, there is no limitation or consideration of the country in the work. However, as a result of this statistic which is seen in Figure 2.4, most of the papers are published in the USA around 11 papers, secondly many published papers from Italy around 7 papers, thirdly many papers were published in China and India around 5 papers for each. The rest of the papers are similarly taken into count in other countries as well.

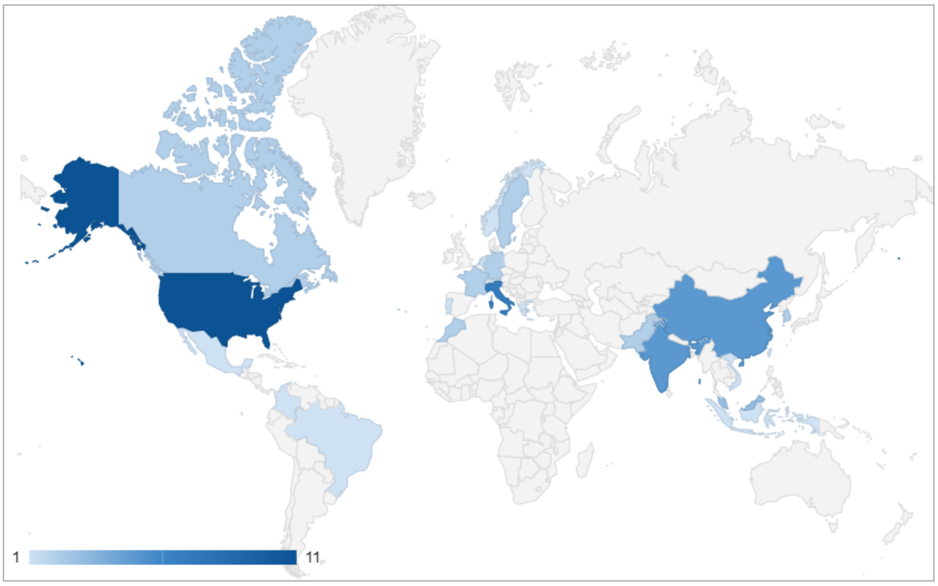


Figure 2.4: World map for the country of publications

2.4.2 Tools for Development(RQ1)

Only 11 out of 59 papers provided hybrid mobile application development techniques and tools. On the other hand, including the combination of test categories, there are 2 papers more and a combination of challenges category, there are 9 papers more. In total, 22 papers mentioned the development tools for hybrid applications, as it seems in Figure 2.2.

According to [**24**], **EasyApp** which is a tool suite that allows ending users to create mobile applications without code barriers. This tool suite involves three sub-systems, namely ServiceAccess, EasyApp, and LSCE. ServiceAccess takes charge of the registration and management of heterogeneous services and can export different forms of services according to the requirements of the other sub-systems. EasyApp is responsible for developing a GUI in the form of a mobile app. LSCE takes charge of creating the application logic that can be invoked by the mobile app directly. EasyApp is a drag-and-drop development environment that enables end-users to develop the GUI of applications. These three modules are all implemented by adopting into **PhoneGap** with mobile web techniques, such as HTML5 and JQuery UI.

According to [**25**], **MVC UI Components Modification Process** is an adaptation technique based on the reuse of manufacturers' SDKs (Software Development Kits) to create multi-user prototype applications. The MVC UI Components Modification Process LOC figures consider only the LOC needed to modify the components based on the Android (Java), IOS (Objective-C), and Windows 8 (C#) prototypes described in the previous subsection.

According to [**20**], there is the history of development platforms of hybrid mobile applications. Especially, **MoSync, RhoMobile, Titanium, PhoneGap, DragonRad, Xamarin 2.0** are mentioned development platforms according to some criteria which are;

Development Criteria	Development Platforms or Tools
Code interpretation	Titanium, RhoMobile
Re-targeting	MoSync, Marmalade
Wrapping	Phonegap, Marmalade

According to [**26**], technology Neutral DSL (Domain-specific language) intended to be cross-compiled to generate native code for a diversity of platforms. The Model Driven Architecture(MDA) approach to provide a platform-independent model under textual format and the M2M M2T transformations are applied to generate the GUI for a specific platform. Their first approach proposes a meta-model Android GUI to design a model of the graphical user interface of an Android application. Then, transformations M2M (Model To Model) and M2T (Model to Text) are applied to generate the code of the graphic user interface targeting a specific platform. In order to do this, they use **Xtext** to define a DSL and Xtend to perform different transformations.

According to [**5**], The most used hybrid development frameworks are **Apache Cordova** and **Appcelerator Titanium** counting to 258 and 116 apps, respectively, whereas all the other frameworks are very less used across all categories.

According to [**27**], using plasticity as a GUI adaptability technique, they develop a mobile cross-platform mechanism to solve cross-platform application development problems. The proposed mechanism considers the dimensions that involve both plasticity and mobile computing. While programming they used Web technologies and native libraries conforming to a hybrid approach. Using a multi-platform development, they employed a framework called **PhoneGap**, where a web page was used as the user interface and compiled in the platform native libraries.

According to [**28**], in order to develop mobile and desktop applications at the same conceptual level, they propose a tool which is **MDA SMARTAPP**, a tool that allows the user of models and provides a way to support transformations for different target device families.

According to [**29**], **Phonegap (a.k.a. Apache Cordova)** from Adobe is a frameworks to build mobile applications using CSS3, HTML5, and JavaScript source code. The code is embedded on a skeleton native app, which interprets and executes the app code on the specific device. On the other hand, **Xamarin or React-Native** is another cross-platform mobile app development framework. It makes it possible to do native Android, IOS and Windows development in C#. Developers re-use their existing C# code and share significant code across device platforms.

According to [**30**], the explosively increasing demand for the development of mobile apps creates the necessity of hybrid apps operates based on web browsers already installed on mobile devices. Therefore web-based application platform emerging as a solution for cross platforms with Web-Based OSMU(One Source Multi-Use) technique.

Ultimately, **PhoneGap** and **Titanium** are the most used hybrid mobile application development tools with OSMU technique.

According to [**31**], **Qt SDK** framework could be a solution for how to develop cross-platform mobile applications. Hence, Qt is a cross-platform application development framework widely used for the development of GUI programs (in which case it is known as a widget toolkit), and also used for developing non-GUI programs such as console tools and servers.

According to [**11**], frameworks like **Titanium** and **PhoneGap** are using widely used web development technologies (especially JavaScript), do not require detailed knowledge of the target platform and are certainly worth considering for building cross-platform applications. Titanium supports the operating systems IOS, Android and Blackberry (is in beta), while supported operating systems by PhoneGap include IOS, Android, Blackberry, Windows Mobile and more.

According to [**32**], **PhoneGap, Rhomobile, JQuery Mobile, and Xamarin** are some of the cross-platform mobile application development tools available.

The comparison of four different development tools is mentioned in [**33**], this paper provides proposes a survey on four major available cross-platform development tools on the market which are Rhodes, PhoneGap, DragonRad, and MoSync.

According to [**34**], **ReactNative, Cordova and PhoneGap**, are increasingly gaining traction, as supporting cross-platform applications is rapidly becoming required for any non-trivial mobile venture. Cordova and PhoneGap are designed to produce web view based applications for a mobile platform by allowing the developer to write the base application in HTML, Javascript, CSS and then to display the application UI components using a provided special framework. ReactNative on the other hand renders this base application code into native platform widgets that call native APIs directly. However, such frameworks are specifically designed to aid in the development of UIs and views.

According to [**35**], there are many different vendors in order to create hybrid mobile applications, such as Application Craft, Appcelerator, Rhodes, dragonRAD, RHOMobile, IBM's Worklight and Apache Cordova/Phone Gap.

According to [**36**], several frameworks are supporting cross-platform mobile application development, such as Sencha Touch, PhoneGap, DOJO, Kendo UI, and jQuery Mobile.

Additionally, in the paper[**37**], there is a comparison and classification about some hybrid appellation development tools such as; **PhoneGap, Titanium, Sencha Touch**. As a result of that paper, The app written in PhoneGap is found to use minimum memory, CPU, and power but provides a very simple user experience. It is also reported that PhoneGap with Sencha Touch 2.0 work significantly well when available memory is not an issue and better UI is desired.

Lastly according to [**71**], many leading industrial companies (e.g., Google) attempt to leverage the native development scheme for cross-platform code development (e.g., **Flutter and Reactnative**). Their solutions are to develop the code only once and make it run on the two different mobile systems.

From all analyzed papers, the Number of citations about hybrid mobile application development tools graph has been created and shown in the Figure 2.5. According to this statistic, the most used tool is Phonegap, and then second used tool is Titanium and the third one is Xamarin.
As a result of the graph in above, there are basic definitions of each hybrid development tools from the most used to least used.

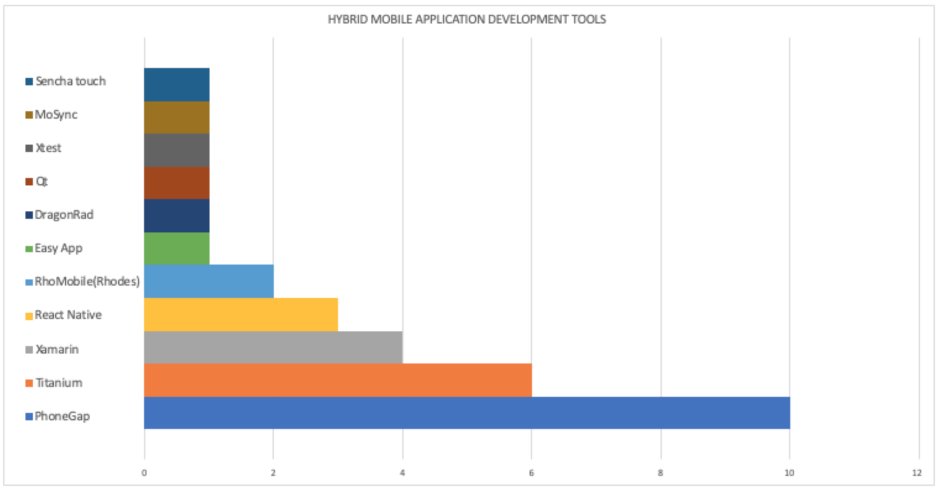


Figure 2.5: Number of citations about Hybrid Development Tools in the considered papers

- **PhoneGap:** is an open-source cross-platform smartphone application development tool developed by Adobe System Inc under Apache license. It provides a decent toolbox for building native mobile applications using only HTML5, JavaScript and CSS. PhoneGap is a "wrapper" that allows developers to enclose applications written in known programming languages into native applications [**20**][**32**][**33**][**34**].

- **Titanium:** is written in JavaScript and complies with a native application and utilizes native controls. A Titanium application provides a set of security features that are required by mobile app developers. It provides a rich API and low-level objects like TCP Sockets. UI objects are customizable through a JavaScript API. There is no HTML and CSS coding here. [**5**][**11**][**37**].

- **Xamarin:** allows developers to automatically build Android and IOS apps using this codebase. The Xamarin framework makes it possible to do native Android, IOS and Windows development in C#. Developers reuse their existing C# code and share significant code across device platforms[**29**][**38**].

- **React Native:** allows developers to write components in JavaScript language, which are converted to native components for IOS and Android platforms. React-Native renders the base application code into native platform widgets that call native APIs directly. This Framework receives as input an application written in a particular not-native programming language and transforms it into native code for a particular mobile platform [**29**][**34**][**71**].

- **RhoMobile or Rhodes:** is another cross-platform mobile application development tool developed by Motorola solutions Inc that is used to build an application for IOS, Android, BlackBerry, Windows Phone, and Symbian. It is an open-source Ruby-based mobile development environment used to develop enterprise applications and data on a single source code across the different operating systems listed above [**32**][**33**].

- **DragonRad:** is a cross-platform mobile application development platform by Seregon Solutions Inc. and distributed under a commercial license. It allows developers to design, manage and deploy mobile applications once and use it across IOS, Android, BlackBerry, and Windows Mobile[**33**].

- **MoSync 4.0:** is an open-source solution developed by a Swedish company targeted to the mobile market. MoSync has fully-fledged SDK which helps developers to build and package all types of mobile applications, such as simple, advanced and complex applications that share the same code base[**33**].

- **Sencha Touch 2.0:** is another powerful yet complex cross-platform mobile application development framework. Its SDK tools provide access to a subset of phone native API such as camera, notification, connection, and orientation[**32**].

- **jQuery Mobile:** is a mobile application development framework that enables and supports touch events and design elements for a wide variety of tablets and smartphones in order to make them look and function like native applications[**32**].

2.4.3 Tools for Testing (RQ2)

There are 24 papers are related to the testing category of the hybrid mobile application testing techniques and tools out of 59 total papers. Additionally, there are 9 papers more which are the combination with other selected papers. In total, 33 papers mentioned the testing tools for hybrid applications, as it seems in Figure 2.2.

According to [**38**], **Android Monkey** tool and analyzes the log files of this execution for certain kinds of faults. The **AndroidRipper** tool also performs stress testing of an Android app but by systematically crawling its GUI. The **M[agi]C** tool is used to generate test cases for apps using a combination of model-based testing and combinatorial testing. **Android GUITAR** can fire only click actions, it seems unfair to use our results with action inference for comparison. **ORBIT** is designed specifically for mobile apps and uses a more precise state-based model, which would also integrate well with other state-based testing techniques.

According to [**39**], GUI-based testing needs some user interaction to move an application from one state to another, therefore **Monkey** testing tool is selected for an automatic event generation tool, to produce events in both random and deterministic ways.

According to [**40**], the tool, named **A2T2(Android Automatic Testing Tool)**, has been developed in Java and is composed of three main components: a Java code instrumentation component, the GUI Crawler, and the Test Case Generator.

According to [**41**], **GUI ripper**, the events of that iterative fire on the GUI and observes the resulting GUI state changes. A GUI ripper can be used to execute automated software testing processes too: in this case, the ripper is used as an engine that fires sequences of events on the GUI of the application under test (AUT) intending to search for application failures.

According to [**42**], **Automated Model Generator for Android apps (AMOGA)**, a tool for automated UI model generation from mobile applications. It is a strategy that uses a hybrid, static-dynamic approach for generating a user interface model from mobile apps for model-based testing. AMOGA implements a novel crawling technique that uses the event list of UI elements associated with each event to dynamically exercise the events ordering at the run time to explore the applications' behavior. On the other hand, the tools selected for the comparison with the AMOGA testing tool are **Monkey, AndroidGUITAR, SwiftHand, ORBIT, MCrawlT, and MobiGUITAR**. The AndroidGUITAR and ORBIT are not available freely to download.

According to [**43**], a concolic-testing approach for generating single events can be extended naturally to iteratively compute sets of increasingly longer sequences of events. An algorithm, hereafter called **AllSeqs** can generate all event sequences of length up to k such that each event sequence executes a unique program path. AllSeqs for several Android apps and found a significant source of redundancy, namely, that a large fraction of events do not have any effect on the program state.

According to [**44**], GUI ripping tools simulate real user events on an Android device to explore an application GUI. The majority of these tools detect and report crashes generated during the exploration. **Coupled** with detection of crashes, others of these tools which are **MonkeyLab, AimDroid, Android Ripper** also reconstruct GUI models resulting from the exploration. A previous effort has been also focused on tools which are **Sapienz, MobiGUITAR** that build testing suites based on their exploration strategy. **CrashScope** enables context-aware input data and sensors and connectivity analysis. CrashScope makes contextual changes in the application based on API calls found statically in the code. **Contextual fuzzing** refers to exercising apps with contexts observed in the wild. In this case, GUI ripping is performed along the contextual fuzzing. **Thor** is another tool that injects unexpected events (i.e., device rotation or incoming calls) in existing test suites.

According to [**45**], **Multi-Level GUI Comparison Criteria (Multi-Level GUICC)** is a set of abstraction levels for automated model-based Android GUI testing. Also developed an automated model-based GUI testing framework to perform automated GUI testing for real-world Android apps, and we evaluate the influence of GUICC on the testing results.

According to [**46**], today, the most popular mobile applications are based on Android and IOS platforms. The functionality of the IOS and Android apps can be successfully tested using Appium automation tool. **Appium** uses the Selenium WebDrivers to execute scripts and being an open-source cross-platform tool, scripts can be written in Java, Ruby, C, Python, Perl which gives the programmers the liberty to use any language.

According to [**47**], **Monkey,** is an automatic event generation tool, to produce events in both random and deterministic ways and feed these events to the application. To discover a wide range of issues, random sequences events are used by the Monkey tool and the sequences are fed to the application under test.

According to [**48**], **Calabash** allowing developers to formally describe touch gestures within a specification for UI tests. Running the test mimics human users by translating the specified gesture into corresponding touch events injected into the AUT upon test case execution. Developers and testers can write test scenarios in Calabash's Gherkin language and directly include touch gesture expressions without the need for manual recording.

According to [**49**], **juGULAR**, a Hybrid GUI Exploration Technique combining Automated GUI Exploration with Capture and Replay. The approach can automatically detect Gate GUIs during the app exploration by exploiting a Machine Learning approach and to unlock them by leveraging input event sequences provided by the user. juGULAR is implemented in a modular software architecture that targets the Android mobile platform. juGULAR covered more source code and Activities and generated more network traffic than the purely automated exploration, thanks to the automatic detection of two classes of Gate GUIs, i.e., Login and Network Settings.

According to [**50**], the top 4 most used Android testing frameworks are selected according to bitbar.com, which is we will evaluate, there are: **Espresso, UI Automator, Appium, and Calabash**. This paper also gives a slight comparison between Appium and other Automated Testing tools.

According to [**51**], **CHECKCAMP (Checking Compatibility Across Mobile Platforms)** can help mobile developers in testing their apps across multiple platforms. An evaluation of the approach with a set of 14 industrial and open-source multi-platform native mobile app-pairs indicates that CHECKCAMP can correctly extract and abstract the models of mobile apps from multiple platforms, infer likely mappings between the generated models based on different comparison criteria, and detect inconsistencies at multiple levels of granularity. CHECKCAMP, which for the same mobile app implemented for IOS and Android platforms (1) instruments and generates traces of the app on each platform for a set of user scenarios, (2) infers abstract models from the captured traces that contain code-based and GUI-based information for each pair, (3) formally compares the app-pair using different comparison criteria to expose any discrepancies, and

(4) produces a visualization of the models, depicting any detected inconsistencies.

According to [**52**], there are some limitations of existing testing tools for GUI-based testing of Android apps in a novel hybrid approach, therefore a new framework approach which is **T+** is created. The approach of T+ is based on a novel framework, which is aimed at generating actionable test cases for different testing goals. The framework also enables GUI-based testing without expensive test scripts collection for the stakeholders.

According to [**54**], The proposed model **TriTest** is an MBT approach, based on **SlumDroid**, extending its functionality by integrating performance and compatibility tests, which are not performed in SlumDroid, and automatically tests mobile apps' functionality, performance, and compatibility, which are important characteristics of quality. TriTest system initiates its process using SlumDroid, the functional test results then generated are used as input to performance and compatibility test modules.

According to [**55**], the only two tools that are widely used in practice for testing Android apps are **Monkeyrunner**, a framework for manually scripting sequences of user inputs in Python, and **Monkey**, a random automatic user input generation tool. There is a proposed testing algorithm, called **SwiftHand**, that uses execution traces generated during the testing process to learn an approximate model of the GUI. SwiftHand then uses the learned model to choose user inputs that would take the app to previously unexplored states. SwiftHand implementation is made in Monkeyrunner and Monkey.

According to [**56**], **MONKEYLAB** provides stakeholders with an automated approach for scenario generation that can be as powerful as manual testing. MONKEYLAB mines event traces and generates execution scenarios using statistical language modeling, static analysis, and dynamic analysis. Moreover, MONKEYLAB can generate scenarios that differ from observed executions enabling it to explore other paths that could trigger unexpected app crashes.

According to [**58**], **MobiGUITAR(Mobile GUI Testing Framework)** conceptual framework, which we implemented in a toolchain that executes on Android. It employs three primary steps: ripping, generation, and execution. MobiGUITAR models the state of the app's GUI, which helps us more accurately model mobile apps' state-sensitive behavior.

According to [**13**], there are many testing tools for both native and mobile Web app testing as well. Especially, **MITE, MonkeyTalk, seeTest Mobile, MobileCloud, Sikuli, Calabash, eggPlant, MonkeyRunner, Robotium, Dollop, Selenium Android, Appium, Selendroid** are the list of testing tools for GUI-based functional testing. Keynote's MITE and Selenium's Android webDriver, for example, only support testing for mobile Web apps, while Selendroid, Appium, and Calabash are designed to support just native app testing.

According to [**12**], a basic test infrastructure consists of a PC running the test script, one local device, and an automation tool. **Selenium Web Driver** and **Robotium** are sample tools for mobile web apps and Android apps testing.

According to [**59**], the **MobiTest** tool is designed to address some of the difficulties associated with the automated testing of mobile applications, by using a single set of unit tests to test the application on multiple platforms. The initial version focuses on testing through the interface as this should be similar on all platforms. In this way, the need for platform-specific code can be minimized.

According to [**60**], **Paladin** to achieve the goals of automated test generation, reproducible executions, and better behavior exposures. The key design of Paladin is to leverage the complete structure of the view tree to identify equivalent app states and locate UI widgets. Paladin was compared with different Automated testing tools which are; **Collider** defines a state as a combination of registered event handlers and transitions as execution of event handlers, **JPF-droid** examines method invocations to verify states, **PUMA** constructs a GUI feature vector and uses the cosine-similarity metric with a user-specified similarity threshold to identify equivalent states, **Stoat** encodes the string of the GUI view tree as a hash value to distinguish states, **VELERA** uses human visual perception to judge the test results, which is impractical for large-scale analysis, **Mosaic** uses a series of scalings and normalization to map coordinates between platforms, **Barista** uses XPath selector to locate UI widgets since the GUI view tree can be mapped to an XML document, **SPAG-C** uses image comparison to identify equivalent states. However, image comparison is very susceptible to slight GUI changes such as different font styles and color settings, **Monkeyrunner** which uses coordinate to locate widgets, **Culebra** uses the GUI content to identify equivalent states, suffering from state explosion as discussed before.

According to [**61**], **PATDroid(Permission-Aware GUI Testing of AnDroid)** performs a hybrid program analysis on both an app under test and its test suite to determine which tests should be executed on what permission combinations. PATDroid performs a hybrid program analysis on both an app under test and its test suite to determine which tests should be executed on what permission combinations. PATDroid currently supports the major Android's GUI test frameworks, namely **Espresso**, **Robotium**, and **Monkey**.

According to [**62**], **Sapienz**, an approach to Android testing that uses multi-objective search-based testing to automatically explore and optimize test sequences, minimizing length, while simultaneously maximizing coverage and fault revelation. It combines random fuzzing, systematic and search-based exploration, exploiting seeding and multi-level instrumentation. Of the publicly available tools, **Dynodroid** and **Monkey,** were found to perform best in the recent comprehensive study. Also, Sapienz always outperforms both Dynodroid and Monkey, statistically significant and with large effect size.

According to [**63**], **Appium** can be termed as a revolutionary tool that can completely change the testing process in a much efficient and swift way. Appium has improved significantly since its inception and is constantly being added up with new features. Since it supports multiple scripting languages which means developers having diverse expertise can use the same tool.

According to [**64**], Android **Robotium** is an open-source tool that enables the automated and black box test execution of third parties applications. The **MonkeyRunner** tool can run an automated functional test for Android applications. **MobileTest** adopts a sensitive-event based approach for the automatic black-box testing of software running on mobile devices. **Android Monkey** tool provides features for stress testing of mobile applications user interfaces.

According to [**65**], **Robotium,** supports test case developers to write function, system, and acceptance test scenarios and simulate Black Box testing for android application. **Ranorex** is mainly used for GUI testing in windows which also supports mobile and web based applications. **Appium** is cross-platform, which allows writing a test against multiple platforms using the same API. **MonkeyTalk** test from simple "smoke test" to sophisticated data-driven test suites. **UIAutomator** is a testing framework, which ensures an app to meet its functional requirements and achieve a high standard quality.

According to [**66**], the systematic exploration tool called **AMOGA** is created in order to comprise an event mapping section that is responsible for extracting all app's supported events statically. AMOGA is compared against the code coverage with other existing Android exploration tools **Android Monkey** which is a testing tool available in the Android SDK that can produce and send events in both random and deterministic ways to an app, **Android GUI Ripper** which dynamically analyses app's GUI to obtain event sequences that can be fired on the GUI, with each sequence representing an executable test case that can be used for regression test and crash testing, and **Orbit** which performs static analysis on the source code of an app to generate set of user actions supported by an app.

According to [**38**], **X-Checker** aims to find bugs in Xamarin by generating apps, executing these apps on Windows Phone and Android, and looking for inconsistencies in them. Thus, X-Checker's design consists of two parts, the test case generator, and the inconsistency checker. X-Checker generates test cases that exercise the programming API used by Windows Phone developers and X-Checker produces a pair of apps for Windows Phone and Android. It executes them atop these platforms to observe inconsistent behavior.

Lastly, according to [**67**], it has been found that Android app developers mostly use automated testing tools such as **JUnit, MonkeyTalk, Robotium, Appium**, and **Robolectric**. The perceived usability of JUnit was the highest (45%). The perceived usability of MonkeyTalk was next (20%), followed by Robotium (17.50%), and then Robolectric (15%). Then automated testing tool with the least perceived usability was Expresso (2.50%).

From all analyzed papers in the above, all mentioned testing tools are collected and counted according to an appearance in the papers, then the number of citations about the hybrid mobile application testing tools graph has been created and shown in the Figure 2.6. According to the graph, the most used tool is Android Monkey, and then second used tools are Appium and Robotium and third tools are MonkeyRunner and Android GUIRipper.

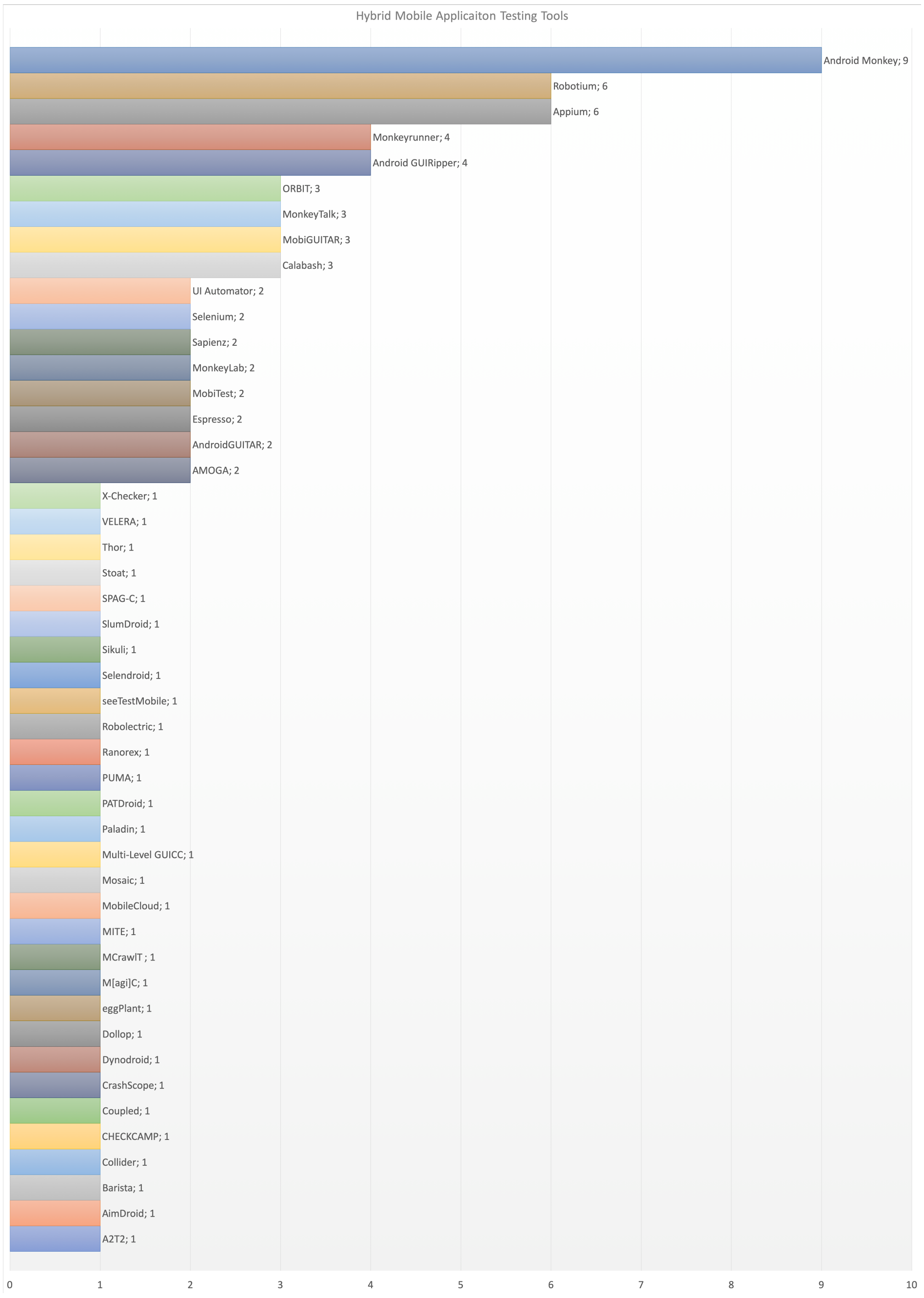


Figure 2.6: Number of citations of Testing Tools in the considered papers

As a result of the line graph above, the most used and desired testing tools are selected to give more details of them. Since some tools are just created for making research, some testing tools have a price, therefore, many different testing tools are not entered to the will be described list below.

- **Android Monkey:** is a testing tool available in the Android SDK emulator, therefore, no need to install explicitly. That tool can produce and send events in both random and deterministic ways to an app and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as several system-level events. You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner. It generates automatic events and analyzes the log files of this execution for certain kinds of faults. To discover a wide range of issues, random sequences events are used by the tool and the sequences are fed to the application under test. This tool provides features for stress testing of mobile applications user interfaces. Therefore it simply gives a very quick idea if any exception can be got by back and forth. [**38**][**47**][**55**][**64**][**66**]

- **Appium:** can be termed as a revolutionary open-source cross-platform testing tool that can completely change the testing process in a much efficient and swift way by using Selenium to execute scripts. Those scripts can be written in Java, Ruby, C, Python, Perl which gives the programmers the liberty to use any language. Appium has improved significantly since its inception and is constantly being added up with new features. Since it supports multiple scripting languages which means developers having diverse expertise can use the same tool. Appium allows writing tests against multiple platforms using the same API that the functionality of the IOS and Android apps can be successfully tested using this automation tool. [**46**][**63**][**65**]

- **Robotium:** Robotium is an open-source test framework that has full support for native and hybrid applications for writing automatic black-box test execution of third parties applications. Robotium supports test case developers to write function, system, and acceptance test scenarios and simulates Black Box testing for android applications. Robotium can be used both for testing applications where the source code is available and applications where only the APK file is available and the implementation details are not known.[**53**][**61**][**64**][**65**][**67**]

- **MonkeyRunner:** provides an API for writing programs that control an Android device or emulator from outside of Android code also it can be accessed to device and application component. With monkeyrunner, you can write a Python program that installs an Android application or test package, runs it, sends keystrokes to it, takes screenshots of its user interface, and stores screenshots on the workstation. This tool is primarily designed to test applications and devices at the functional/framework level and for running unit test suites, but you are free to use it for other purposes. monkeyrunner which uses coordinate to locate widgets and also runs an automated functional test for Android applications. The monkeyrunner tool is not related to the UI/Application Exerciser Monkey, also known as the monkey tool. [**55**][**57**][**60**][**64**]
- **Android GUIRipper:** can be used to execute automated software testing processes that are used as an engine that fires sequences of events on the GUI of the application under test (AUT) intending to search for application failures. It is a tool also performs stress testing of an Android app but by systematically crawling its GUI. Android GUI Ripper which dynamically analyses app's GUI to obtain event sequences that can be fired on the GUI, with each sequence representing an executable test case that can be used for regression test and crash and observes the resulting GUI state changes testing.[**38**][**41**][**66**]

2.4.4 Challenges for Development/Testing(RQ3)

There are 5 papers are related to the category of the challenge of the hybrid mobile applications. Additionally, there are 9 more papers which are the combination with challenges and development of hybrid mobile applications. Lastly, 7 more papers are related to the combination of challenges and testing. In total there are 21 papers are under the category of challenges out of 59 total papers as it seems in Figure 2.2. In this part of the paper, papers will be analyzed based on *Grounded Theory* that technique is explained under the title 2.3. While analyzing papers, coding, and memoing techniques were used to differentiate and make high-level analyses about the challenges. During the categorization of the challenges, it realized that collecting challenges under 3 high-level categories which are *general challenges* about hybrid mobile applications, *development challenges* and *testing challenges* of hybrid mobile applications and analyzing under these subtitles.

It is better to start with [**3**], because this paper has a very good categorized collection of challenges in Hybrid Applications which have already done a similar study like this paper. Hybrid mobile applications challenges are;

- **Testing** challenge is the most severe challenge reported in 47% of studies. Hybrid applications are targeted to run across platforms, thus, testing each aspect in such applications quite challenging for developers.
- **User Experience** challenge is the second frequent mentioned challenge in the literature (44%). Hybrid applications are not executed like native applications. Therefore, developing these applications have several issues due to different user interfaces, lack of access to device features, and contextual factors, which ultimately affects the performance and user experience (usability, response time, and reliability) of hybrid applications.
- **Compatibility** issue in hybrid application development is the

third frequent challenge as reported in 21% of the identified literature. Hybrid applications also have compatibility issues due to platform-specific constraints or access to API when translating from one platform to another.

- **Lack of Tool Support** was reported in 12% of research studies. Developing hybrid applications are quite challenging since it has both native and web applications characteristics.

- **Lack of Expertise** was reported approximately 12% of studies as a significant challenge in hybrid application development. Developers are always faced with a lack of expertise in development tools (translating code efficiently from one platform to another) and knowledge of each platform's API or features.

- **Change Management** challenge is reported in 11% of the identified literature studies. The frequency of the change management challenge is less reported in the literature. Nevertheless, managing change requests is quite difficult to tackle for developers.

- **Fragmentation** challenge is mentioned in 4% of the articles. Hybrid applications are aimed to overcome the shortcomings of both native and web applications.

- **Security** is one of the least reported challenges in the literature (4%). The web part of the hybrid application is prone to security threats due to its open and dynamic interaction.

- **Reuse of Code** is the least mentioned challenge in the literature (2%). However, we consider this as one of the significant challenges that need to be tackled by academia and industry.

According to [**10**], Hybrid applications are inferior in performance compared to the native applications since the execution happens in the browser engine.

- Hybrid applications are inferior in performance compared to the native applications since the execution happens in the browser engine.

- Since a hybrid application uses JavaScript Hardware abstraction layers, it is subjected to cross space communication vulnerabilities. Hybrid applications also suffer from platform-specific behavior of JavaScript and threading model incompatibilities with JavaScript.

- Even though the user interface can be reused across different platforms the user interface will lack the look and feel of a native application. To achieve the native look and feel the platform-specific styling might be required.

According to [**68**], challenges in Hybrid Applications are;

- **Performance** is a slight one issue that can be considered as a challenge of the hybrid mobile applications because according to a survey they perform poorly when dealing with low-level, platform-specific features.

- **Testing** hybrid development frameworks are perceived as better suited for finding the presence of bugs

- **User experience** is another challenge that should be considered because if a hybrid application seems more like a similar to the native version it is more preferable.

- **Reuse Code** is the reason for the creation of hybrid development but it needs existing knowledge of web developers to be able to reuse.

According to [**69**], potential challenges on the use of multi-platform are mentioned that;

- **Development Tool** needs to overcome the constraint of utilizing different languages and frameworks for each platform.

- **Development Practices** needs to take advantage of the knowledge and expertise already attained by programmers.
- **Development Cycles** needs to be developed once, deploy anywhere.
- **User's Experience** needs to do not allow access (or provide limited access) to some features of the mobile device.
- **Application Marketing** needs that applications can be distributed through a variety of marketplaces.

According to [**70**], developing for multiple platforms is a recurring problem that is not unique to the mobile world;

- **the lack of proper development and analysis support** in the mobile environment exacerbates the challenges.
- **Correctness and consistency** of the app across different platforms. One way to tackle this problem is by constructing tools and techniques that can automatically infer interaction models from the app on different platforms.
- **Testing** challenges, follow-up studies could focus on generating test cases for mobile apps. A centralized automatic testing system that generates a (different) test case for each target platform could be a huge benefit. While platform-specific features can be customized, core features could share the same tests.
- The **existing testing frameworks** have serious limitations for testing mobile-specific features and scenarios such as sensors, rotation, navigation, and mobility (changing network connectivity). As consequence developers either need to write much test fixture code to assert mobile-specific scenarios or opt for manual testing.
- **Rooted emulators** that can mimic the hardware and software environments realistically limitation is another challenge.

- **Better analysis tools**, in order to measure and monitor different metrics of the app under development, is a necessity, therefore, having an analysis tool could be a challenge of cross-platform mobile applications.

According to [**11**], cross-platform application challenges are in the following:

- **Market place deployment** is one of the challenge developers need to consider that evaluate whether and how easy it is to deploy apps to the app stores of mobile platforms.
- **Widespread technologies** should be evaluated whether apps can be created using widespread technologies, such as JavaScript.
- **Hardware and data access** is another challenge that evaluates whether apps have no access, limited or full access to the underlying device hardware and data.
- **User interface and look and feel** evaluates whether apps inherently support native user interface components or native user interface and look and feel is simulated through libraries, such as JQuery Mobile.
- **User-perceived performance** evaluates whether apps have low, medium or high performance as perceived by the end-users (like loading time and execution speed).

According to [**32**], cross-platform development environments are challenged by the **different implementations, immature platform support, variety of devices, and variety of browsers** while the platform-specific ones. Cross-platform development tools are flourishing aiming at addressing **user experience, the stability of framework, ease of updating, cost of development for multiple platforms, and the time to market of an application**.

According to [**33**], it is a big challenge to develop high-performance mobile applications in this competitive market that would meet the expectation of customers. Cross-platform application development challenges are categorized below;

- **Mobile Operating Systems** supported to understand possible effects on respective business models.
- **Tool licenses** offered to evaluate the terms and conditions of use.
- **Programming languages** offered to developers for building applications.
- **Availability of API's** provided to get an idea of different hardware parts accessible in the OS.
- **Accessibility to native API's** to compare how it is possible to access them from each tool.
- **Architecture** provided for the development process of the application.
- **Integrated Development Environments** available for developing applications.

According to [**9**], hybrid mobile application have some challenges which are;

- **Potential customer base** of applications could be the first he challenges.
- Providing a better **user experience** is with the UI builders is another challenge.
- **Fragmentation and the lack of updates**. Fragmentation is simply the problem that there are so many different devices supporting Android, and it is difficult to create an App that works across all various sorts. The lack of updates is the case that certain devices, even quite new, will not receive updates of the OS.

- **Manufacturer of device** difference is also another challenge for hybrid applications because making consistent for all devices hard such that Android has many different manufacturers and to be consistent with all of them could be a trade-off when we compare with a market share of Android vs IOS.

According to [**35**], there are three main challenges of a hybrid mobile application is that ;

- **User Interface(UI)** layer has to be developed separately for each hand-held mobile device such that it can provide device-specific gestures to increase user satisfaction. Hence the native application development offers the ultimate user experience and performance for mobile applications.

- **Fragmented set of development tools** and multiple versions of an application to serve the same user need. Using the above mentioned frameworks and products is a sensible option when developing apps for new initiatives where no functionality has been deployed yet.

- **Translator** is necessary that the non-UI components be translated into the target platform's language(s) Objective-C, C# for deploying a mobile app on Apple and Microsoft mobiles respectively. since the core app engine is translated to the target platform using open source or commercial translators.

According to [**36**], cross-platform mobile applications have some challenges about **UI, reusability, a variety of mobile devices, and development tool support**. The wide variety of hardware and software of smartphones is one of the difficulties for reusability in software development for mobile devices. This challenge initiates many types of research towards adapting UI across many devices. Most of the common development tools for visual design user interfaces are local applications, which require developers to install locally in their computers not like downloading once and deploying on many supported platforms.

According to [**37**], cross-platform frameworks have to consider some requirements which are listed below, therefore those requirements seem as challenges of cross-platform frameworks most of them are related to development challenges.

- **Multiple mobile platform support** for Android and IOS are very essential since they have the largest share in the application markets.
- **Rich user interface** should support for sophisticated graphics (2D, 3D), animation, multimedia are necessary. Since the success of an application highly depends on the user experience of the interface, rich UI development should be incorporated.
- **Back-end communication** protocols and data formats are absolutely mandatory. Hence mobile devices promote an "always-connected" model in which the users are sharing material in social networking sites, watching videos, communicating via live chat, gathering information 24X7.
- **Security** needs to be considered of applications developed by cross-platform tools because in general, they are not highly secure.
- **Support for app-extensions** is required to install app extensions on top of existing applications like in app purchase/billing capability.
- **Power consumption** is an important issue nowadays with thousands of smartphones and tablets are being activated daily. The generated applications must be optimized for power.

According to [**71**], the cross-platform framework faces the following difficulties:

- It is difficult to extract the components in the GUI pages accurately.

- The types of the extracted components cannot be identified only relying on the image processing techniques.
- There's no such a technique that maps and transfers the GUI implementation code of the two different platforms (i.e., Android and IOS).

It can be understood that these difficulties mentioned above are mainly merged under **User Interface** challenge of mobile application development.
According to [**44**], testing is an underlying necessity in hybrid mobile applications to deliver high-quality apps.

- **defining tests suites** and **large combination of mobile devices** and **operating systems** for app testing is a difficult task that requires a lot of effort, because it must consider all the possible states of an app, its context (e.g., device in which is running, sensors, touch gestures, screen proportions, connectivity) .
- **GUI extraction** is another difficult task for testing due to the nature of model extraction based on GUI events. The majority of automated testing tools detect and report crashes generated during the exploration. Coupled with the detection of crashes, others of these tools also reconstruct GUI models resulting from the exploration.

According to [**52**], some challenges impact the automatic generation of test cases for Android apps during the preliminary experiment.

- **encapsulated components** (e.g., KeyboardView, AutoCompleteTextView, CalendarView, DatePi cker) can not be analyzed during systematic exploration, because the subcomponents (e.g., each key of the keyboard) are not available for ripping at execution time.
- **testing approaches** should be able to deal with clean (i.e., no previous data or cache in the app) and dirty launches (i.e., the app keeps data from previous executions).

- **the arrival time between events, during real execution of mobile apps**, should also be modeled to reduce idle time during testing and to avoid the situation where feasible events become infeasible (one option for dealing with this is by generating test cases written with the Android UI Automation API).
- **statistical models** require appropriate mechanisms for identifying the required size of the training corpus.

According to [**56**], there are some challenging tasks in GUI-based hybrid Android app testing.

- **Generation of test scenarios or test cases** is a difficult task because of the extraction of GUI models from Android app execution traces, events, or source code.
- Mobile developers and testers face other emerging challenges such as;
  - **rapid platform/library evolution and API instability**
  - **platform fragmentation**
  - **continuous pressure from the market for frequent releases**
  - **limited availability/adequacy of testing tools for mobile apps**.

According to [**13**], cross-platform applicaiton testing has some difficulities and challenges.

- **construction of mobile test environments** still involves high costs and levels of complexity. Setting up a mobile test environment for multiple apps on each mobile platform for a range of devices is tedious, time-consuming, and expensive, and frequent upgrades in both device and platform space only exacerbate this challenge. Additionally, there are some sub-challenges with related the main challenge;

  - **The lack of standardization in mobile test infrastructure, scripting languages**
  - **Connectivity protocols between mobile test tools and platforms**
  - **The lack of a unified test automation infrastructure**

- **the market for cross-platform mobile apps** will grow 88 percent from 2009 to 2014, bringing an even stronger demand for mobile test automation solutions that can cope with the issues and challenges we described earlier.

According to [**59**], the creation of cross-platform testing tool presents a number of challenges:

- **Use of XML to specify components** although IOS and Android can specify their layouts with XML, it may be that some platforms do not.
- **Incorrect assumptions about layout** suppose there are IOS and Android layout XML files that both specify two buttons. Based on the order in which components are specified in the file, it will be assumed that the first button from one file corresponds to the first button from the other.
- **Conflicting guidelines** both IOS and Android provide a set of guidelines for user interfaces. These guidelines are not always compatible.
- **Platform-specific features** each mobile platform has several components unique to that platform.
- **Inconsistent number of screens** a single screen on one platform may correspond to multiple screens on another.

According to [**64**], a systematic software engineering approach to software testing is aimed at maximizing fault detection, making results reproducible, and reducing the influence of external factors, therefore there are some challenges;

- **Test Selection** leads to the unpredictability and high variability of the inputs the application is potentially receiving.
- **Test Execution** leads to the challenge on how to execute test cases including rich contextual inputs.
- **Structural** mobile application languages add some specific constructs for managing mobility, sensing, and energy consumption.
- **Functional** mobile applications functional testing requires to specify both the application and the environment it may operate in (especially in MobileApps).
- **Performance and reliability testing** performance and reliability of mobile applications strongly depend on the mobile device resources, on the device operational mode, on the connectivity quality and variability, and other contextual information.
- **Memory and Energy testing** memory leaks may preempt the (limited) memory resource, and active processes (of killed applications) may reduce the device (battery) autonomy.
- **Security testing** security is particularly relevant due to the mobility of the device into networks with different security levels. A Trojan might have access to personal data, private networks, and private contextual information (e.g., where the application user is located).
- **GUI testing** test whether different devices provide an adequate rendering of data, and to test whether native applications are correctly displayed on different devices.
- **Product line testing** is an application on a multitude of mobile devices is certainly an important challenge, especially in the Android O.S., where different mobile phones provide different features and hardware components, and phone manufacturers may deeply customize the O.S.

Out of 21 papers were selected as Hybrid Mobile Application challenges were manually examined, and creating categories for challenges through the application of Theoretical sensitivity, Coding, and memoing according to Corbin's Grounded Theory approach[**23**]. Theoretical sensitivity helps to conceptualize different words with a similar meaning and make a relationship between them. Additionally, coding helps to put the same category name if they are serving the same purpose. Lastly, memoing helps to put those words in the taxonomy diagram to visualize for the reader in the high-level perception. At the end of categorized challenges, it is found that challenges are divided into three high-level categories which are related to Development, Testing and General Challenges about hybrid mobile applications from all related papers.

Development Challenges of hybrid mobile applications

Development Challenges were examined and categorized thanks to theoretical sensitivity that helps understand challenges related to the development phase of the application. Then, challenges were coded with meaningful and related name and then thanks to the memoing taxonomy diagram are created and shown in Figure 2.7. Additionally, the occurrence of these categories was counted and made a graph about them. Out of all related papers, there is 41 occurrence of challenging keywords that are mostly related to development. According to that occurrence of keywords, User Interface and User Experience is the most appeared challenge(31.7%) in papers. The reason for it could be that hybrid mobile application tried to more seem like native versions. The second most appeared challenge is Development Tool support(24.3%) and the reason behind it could be a difficulty of finding the best compatible development tool both cover all native and web application characteristics. And the third challenge is Performance(12.2%) and the reason behind it could have problems like working slow while trying to reach platform-specific features. More details of the hybrid development challenges shown in Figure 2.8.

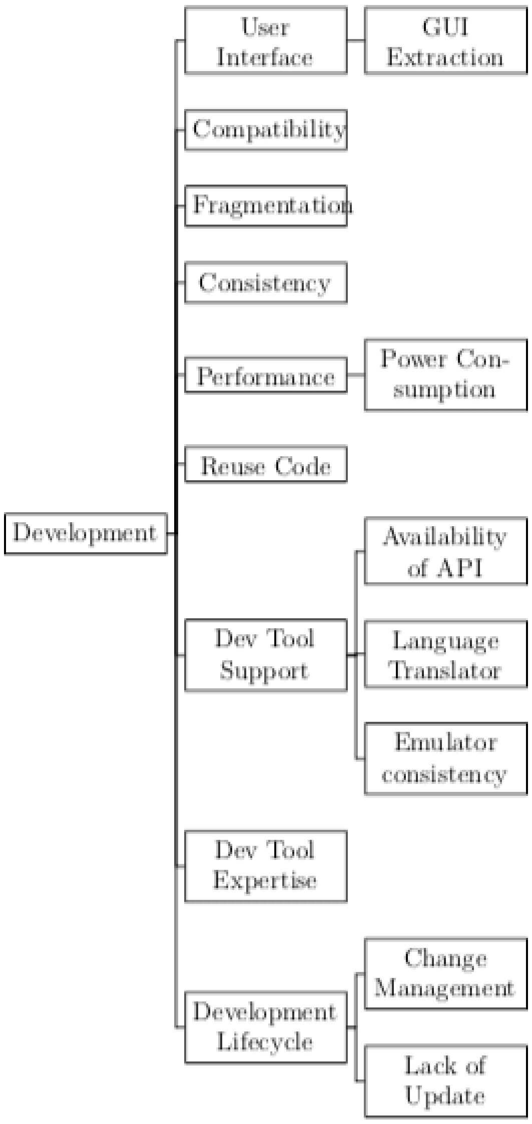


Figure 2.7: Taxonomy of Development Challenges

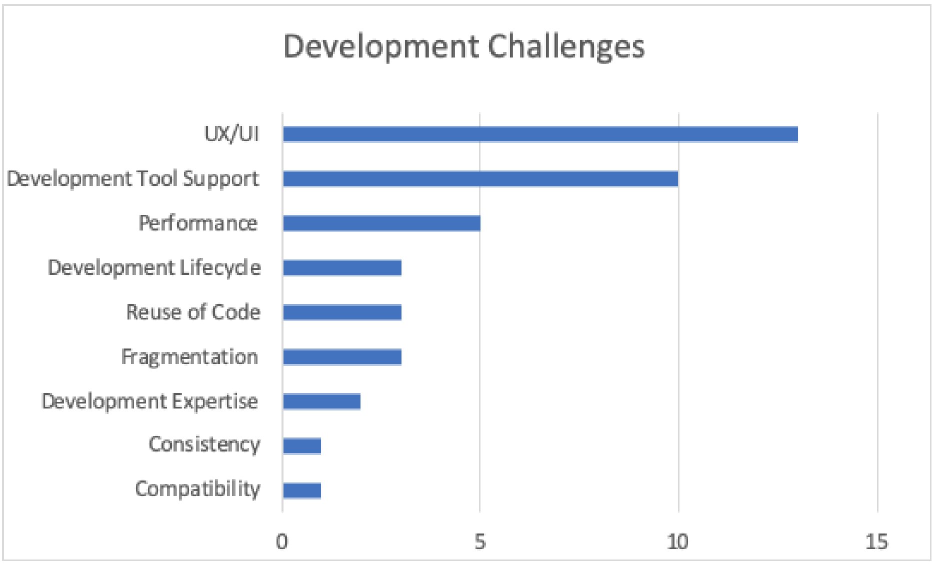


Figure 2.8: Hybrid Mobile Application Development Challenges

Testing Challenges of hybrid mobile applications

Testing Challenges were examined and categorized the same methodology with the development challenges approach. Categories were scratched to make more visualize as taxonomy and shown in Figure 2.9. There is a statistical graph about high-level categories of testing challenges shown in Figure 2.10. Furthermore, the same occurrence statistic was made for this kind of challenge subdivision. According to statistics collected from all related papers, there is 23 occurrence of different keywords related to challenges about testing. Testing Approach, Testing Tool/Framework, and Platform Specific Features challenges have the same ratio (26.1%) and the most challenging topics.

Testing Tool and Framework have many sub-challenges such as Tool Standardization or API Availability because there are many different testing tools and those criteria need to be considered while hybrid mobile application testing. Also, the number of different kinds of testing tools is quite big when compared to development tools, therefore, tool standardization has a very significant role.

Other problems are that even if the application would be developed in any framework with whatever you won't like the same layout for each of different operating systems, etc. However, at the end of testing it, it is necessary to take in the count for every Platform-specific features aspects of different operating systems, because they could have different structural and functional features.

Test Tool Expertise is also necessary because of time and cost consideration since the well prepared conflicting guidelines would be helpful for testers to deal with problems during the testing at the end it helps saving more time and time money. Fragmentation is a general problem for both development and testing perspectives.

Lastly, Statistical model support could be beneficial at the end of the test by creating statistics about bugs, etc. Hence, it is very important feedback for the application quality.

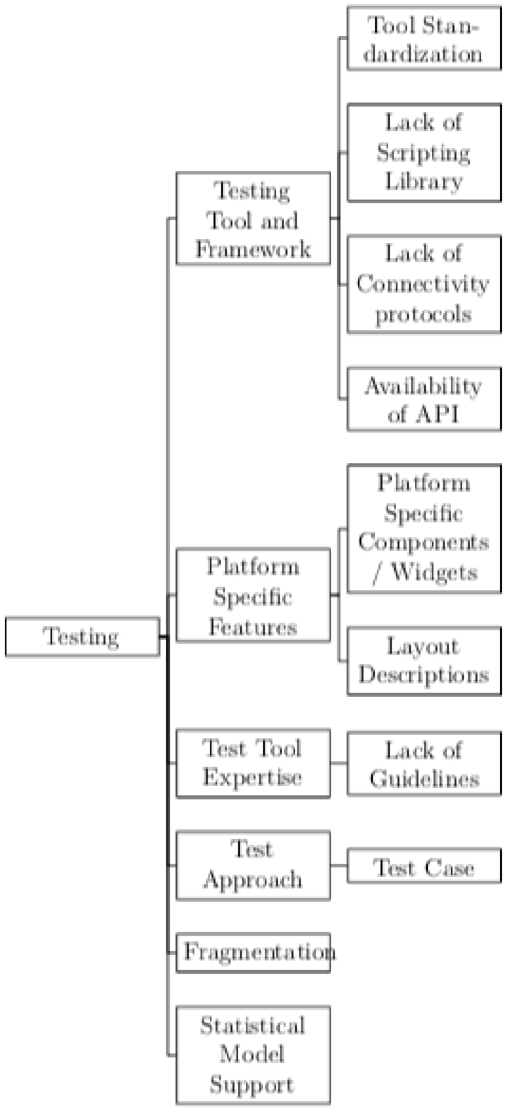

Testing
Testing Tool and Framework
Tool Standardization
Lack of Scripting Library
Lack of Connectivity protocols
Availability of API
Platform Specific Features
Platform Specific Components / Widgets
Layout Descriptions
Test Tool Expertise
Lack of Guidelines
Test Approach
Test Case
Fragmentation
Statistical Model Support

Figure 2.9: Taxonomy of Testing Challenges

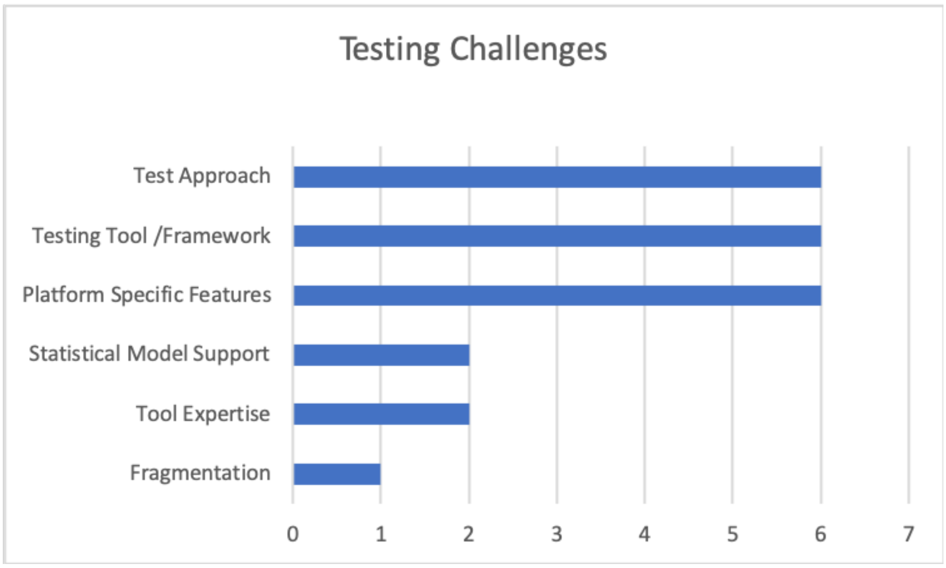


Figure 2.10: Hybrid Mobile Application Testing Challenges

General Challenges of hybrid mobile applications

General Challenges are more related to common hybrid mobile application features especially not technical detailed means that not related to both development and testing. All 21 papers, some challenges were not grouped under the development or testing categories. Thus, it was better the called them as general challenges and the taxonomy of it shown in Figure 2.11. Since these challenges are more related to high-level features or inspections of hybrid applications.

From all analyzed papers, only 18 times these kinds of challenges were faced, and the most challenging one is a variety of devices and browsers(33.3%) should be considered the first challenge of this category. Additionally, this challenge could be supported by other statistics that it mentioned in the introduction of this paper which is "nowadays, the number of mobile phones is forecast to reach 4.68 billion[**2**]". It is an inevitable fact that today, in the mobile phone market there are a lot of different devices and different browsers specific to that device.

Second most challenge is about Application Market (27.7%) such as Google Play Store, AppStore, etc and also it was mentioned in introduction again "today the most used operating system is Android[**9**]", and it means that the winner of application market is Google Play Store and the balance should be stabilized thanks to hybrid applications. Therefore hybrid mobile applications have a very huge and strong impact on the application market topic.

Details of the number of citations about general challenges can be seen from Figure 2.12

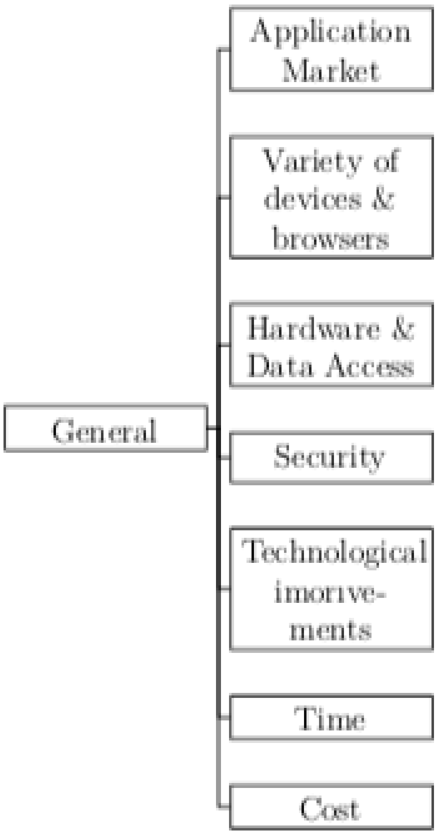


Figure 2.11: taxonomy of General Challenges

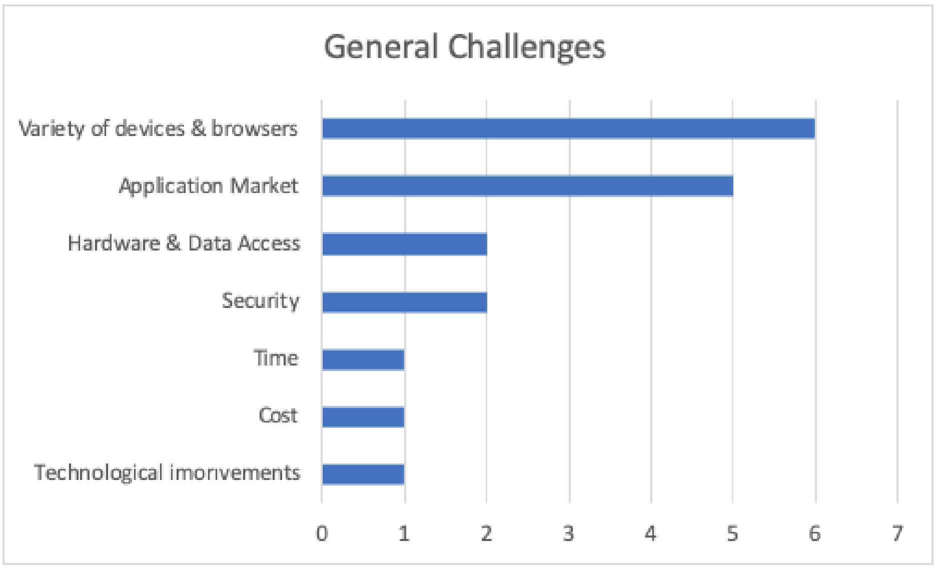


Figure 2.12: Hybrid Mobile Application General Challenges

2.5 Discussion and Conclusion

The purpose of this study was to collect, interpret and analyze all shreds of evidence related to hybrid mobile application development, testing techniques, approaches, and challenges. This study indicates several research gaps that acquire further research and investigation and find out the answers to three research questions.

The answer of RQ1 is reported in the section under 2.4.2 and Figure 2.5. Among collected related papers, the three used hybrid mobile application development tools are Phonegap, Titanium, and Xamarin, respectively, whereas all the other frameworks are very less used across all categories.

The answer of RQ2 is reported in the section 2.4.3 and Figure 2.6. The three most used hybrid mobile application testing tools are; Android Monkey, Appium, and Robotium. All three testing tools have almost the same qualifications as the diversity of operating systems, free to use, and have good guidelines. For future research, these tools can be selected for making deeper comparisons.

Lastly, to draw high level perspective of RQ3, made detailed explanations are provided in section 2.4.4 with general results shown in the Figures 2.8, 2.10, and 2.12. For answering this question, three major challenge categories were selected to mainly focused on the topic which are development, testing, and general challenges. Development challenges mainly focused on User Interface which is more related to differences among operating systems and trying to create similar visualization for all operating systems, Development tool Support, and Performance of the applications.

Chapter 3

Source Repository Analysis and Exploratory Study

In this chapter, open-source hybrid mobile apps mining and an exploratory work about automated GUI testing for hybrid apps were performed. The process of conducting the mining part and making an analysis of it will be explained in the next section. Then, in the second section, a real-life example of automated GUI testing on a Hybrid Android application will be experienced based on some use cases.

The purpose of these works is getting results of these automated testing tools for hybrid mobile applications. In order to make good assumptions, literature reviews are not enough. Due to this reason, the top five open-source automated testing tools, which are explained in the previous chapter "Testing Tools" section, are selected for searches in the GitHub project repository. The mining process will be explained in the next section. In the second part of the chapter, the tentative to test a randomly-selected hybrid mobile app with those mostly cited mobile frameworks.

3.1 Source Repository Analysis

According to the results of the literature review related to the operating system, Android is the most used operating system among mobile applications. Thanks to new cross-platform development tools, developers can develop not only Android but also other operating systems. In the literature review results, Phonegap, Titanium, Xamarin, React-Native are the most used open-source development tools. Additionally, Monkey, Appium, Robotium, Monkeyruner, and Calabash are the most used open-source testing tools. Thus first filtering of the mining made around the operating system of the application, later development and testing tools will be filtered as well. The thesis needs some real data for this research, therefore, R. Coppola's[**73**] paper was analyzed and used their data in Github to filter some hybrid projects. This analysis made by gathering open-source projects from GitHub and applying some filtering thanks to Bash scripts that were created. The purpose of this work is by answering the Research questions below:

- RQ4.1: How many projects use cross-platform development tools?
- RQ4.2: How many projects use automated testing tools?
- RQ4.3: How many projects developed by cross-platform tools and tested with automated testing tools?

The starting point of the analysis was the set of 9827 projects provided in the mentioned paper. Later a filter was applied to identify cross-platform development tools keywords such a PhoneGap, Cordova, react-native, etc. Then, testing tool keywords such as Appium, Robotium will be used for filtering from the specific projects. Later on this automated search, there was an output file that has project names and files that include searched keyword.
Furthermore, those projects were analyzed as being good candidates or not manually. If the searched keyword is found in a comment or variable name, or just a specific purpose will cause the project will

be excluded. According to this last manual filtering criteria, the collected results result that are detailed later, were not sufficient to make statistics about Hybrid app development and testing tools.

How many projects use cross-platform development tools?

It is necessary to choose cross-platform development tools that are mentioned at the beginning of this section as Phonegap, Titanium, Xamarin, and React-Native. It seems from the Figure 2.5, top four development framework was selected because other development tool occurrence is very few and not a good candidate for making this analysis. There are exclusion criteria as well for the selection of the candidate projects. If the search keyword appears in the comment blocks, or to be a dictionary word, not related to hybrid development just a specific variable naming or library usage, exclude these projects from the output list. After applying this exclusion criterion, the real applicable projects will remain.

It reported in Figure 3.1, among all projects only 26(0.0026%)projects have been filtered as Adobe Cordova - Phonegap projects, out of these 26 projects only 15 of them developed as hybrid android applications and most of them are tutorials, not real projects. For Titanium 26(0.0026%)projects found but none of them is development, all projects excluded from the output list. Additionally, only 8(0.0008%) projects have been filtered with the react-native keyword, and only 4 of them are developed by React-Native. Then, only 15(0.0015%) of them have been filtered with Xamarin keyword, and only 1 of them is really developed with hybrid development.

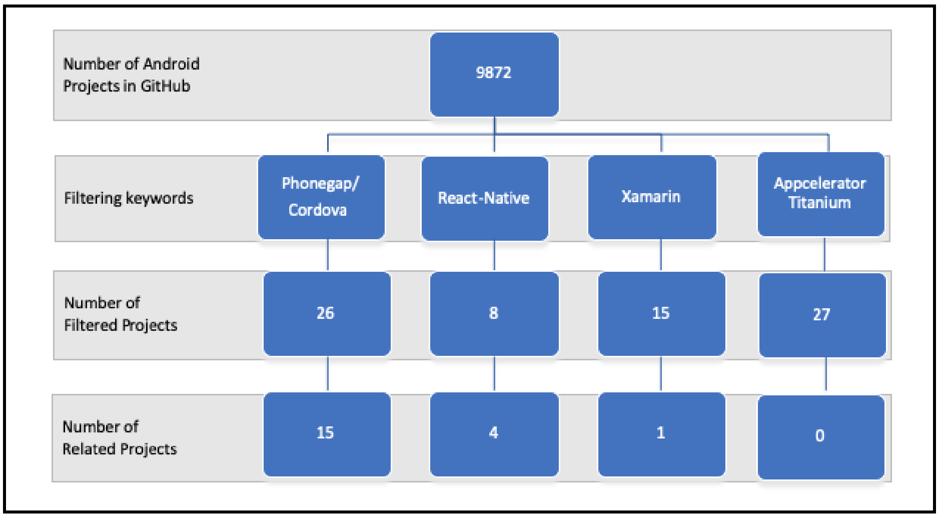


Figure 3.1: Number of projects associated to the selected development frameworks

How many projects use automated testing tools?

In order to decide which testing tools will be used in this mining, Figure 2.6 result is used. Among the top five open-sources, testing tool keywords searched in GitHub. In order to make another mining with the same data set 9827 projects wherein GitHub, the same methodology of the deciding development tool is being applied as well. This means that specific keywords will be searched inside GitHub automated by script language. If the search keyword appears in the comment blocks, or to be a dictionary word, not related to hybrid mobile application development just a specific variable naming or library usage, exclude these projects from the output list. At the end of this exclusion criteria application, the real applicable projects will be remain.

It reported in Figure 3.2, among all projects, 441 projects filtered with Monkey keyword-only 316 of them tested with Android Monkey tool. However, this tool is inside of the Android studio and no possibility to write code because that tool is working as pressing random buttons, entering some keywords something like that. For that reason, it is excluded from this analysis. Among all projects, 49 projects have been filtered with appium keyword and 41 projects have been tested by it. Out of 86 filtered project with Robotium keyword, 74 projects have been tested with Robotium. Among all projects, only 6 of them tested with Monkeyrunner and there is no project tested by Calabash.

Testing tools analysis gave more results than cross-platform development tools but still not enough to make statistical analysis. It is obvious that around out of 10K projects, the maximum valuable project number is very low to make some predictions and assumptions. For that reason, it is better to collect some precise data manually. As it is better to select a hybrid mobile application and try to test automatically it then compares with the results of testing tools.

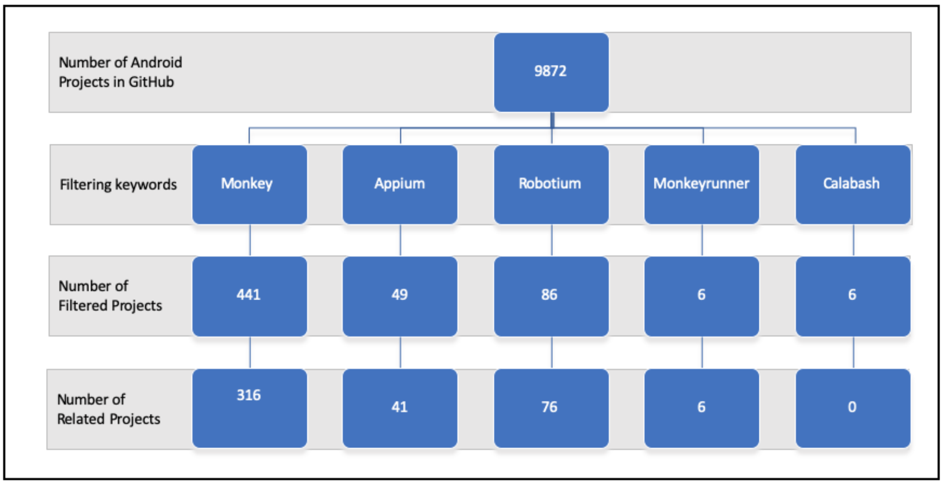

Number of Android Projects in GitHub
9872
Filtering keywords
Monkey
Appium
Robotium
Monkeyrunner
Calabash
Number of Filtered Projects
441
49
86
6
6
Number of Related Projects
316
41
76
6
0

Figure 3.2: Number of projects with the selected testing frameworks

How many projects developed by cross-platform tools and tested with automated testing tools?

After collecting data for development and testing tools in the previous part, it is better to move on to check is there any intersection between development and testing tools. For example, trying to find out is there any project that is developed by selected cross-platform tool and then tested by selected test tools. Unfortunately, again the number of intersection sets of this question is very low as well, reported in Table 3.1. The total number of the project has a cross-platform development tool and the testing tool as well is 9(0.009%).

Number of Project	Development Tool	Testing Tool
7	Cordova-PhoneGap	Robotium
1	Cordova-Phonegap	UIAutomator
1	React-Native	MonkeyRunner

Table 3.1: Distribution of projects among development and testing tools

According to the results of this mining shows that Robotium is the most used tool which is developed by Apache Cordova Phonegap Development Framework. Unfortunately, this is not a good number to perform a thorough static code analysis. For that reason, doing a real-life experiment to observe automated testing tools behavior is a good choice.

3.2 Exploratory Study

This last empirical study was carried out by selecting a Hybrid Android application which is Fresh-Food-Finder application, then this application will be evaluated by the testing tools which are Appium, Robotium, UIAutomator, Selendroid and they will be compared against each other in usability of the test tool, test execution speed, maintainability of the test code, reliability of the test tools and in general issues. The test suites were then run and the execution speed and reliability were analyzed based on these results. The test code is written is also analyzed for maintainability by calculating the lines of code and the number of method calls. The issues faced by the test developer with the different tools are also analyzed. Shortly in this section, it will be covered the brief evaluation of testing tools on a single hybrid application. In the first part, will be used the project selection process will be explained concerning selected software objectives. In the second part, used testing tool selection process and basic information about selected tools will be explained. In the third part, it will be present information about test cases for testing automatically. In the fourth part, test case implementation will be explained.

3.2.1 Application Selection and Software Objectives

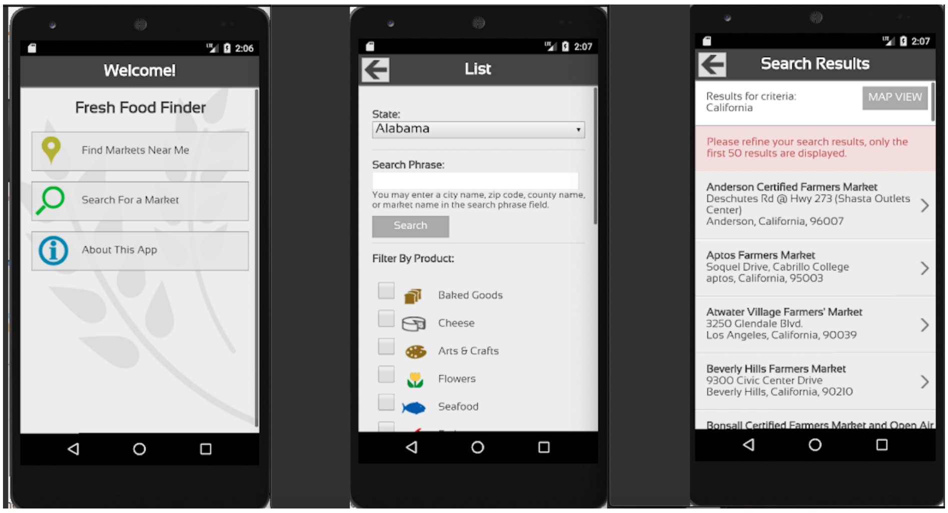
In order to select a hybrid mobile application, Google search has made, some technological blogs have found in order to identify selecting criteria. Hybrid mobile applications should be open source to analyze it better, thus GitHub is a very useful platform to find a good project for this purpose. The candidate projects are evaluated carefully with usability, the functionality of the application, the number of screens, number of classes are the criteria to select the best candidate for this evaluation.

In the light of these criteria, the Fresh-Food-Finder application seems enough to satisfy them. The project can be directly downloaded or viewed from Github. [1] Fresh Food Finder is an open-source mobile application developed with Phonegap which is the most used cross-platform development tool as a hybrid mobile application. It is an application to find fresh markets under some criteria such as category, city, payment type for users. The application is very easy to use because only 3 buttons appear on the main page of the application

[1]https://github.com/triceam/Fresh-Food-Finder

like in the Figure 3.3. Users can directly see fresh food markets by pressing the first button whose name is "Find Markets Near Me". It directs the next page to the maps based on customer current location. Also, user can decide some filtering options like state, product type or payment type for his specific search by pressing the "Search For Market" button like in the second image in the same figure. When a user wants to see the details market, the address of the market and location on the map can seem as well.

After introducing the application, it is better to touch the technological perspective of the application. It is written entirely using HTML, CSS, and JavaScript, and runs on different platforms such as Android and IOS. The code is organized into the following structure:
assets – This folder contains fonts, images, and CSS styles used within the application.
js – This folder contains JavaScript resources and libraries used within the application.
views – This folder contains UI/Mustache templates. Each template is within a separate HTML file.
The majority of the application logic is inside **application.js**, all views are rendered from the Mustache templates inside of **viewAssembler.js**, and all UI styling is applied via CSS within **styles.css**.



State:
Alabama

Search Phrase:

You may enter a city name, zip code, county name, or market name in the search phrase field.

Results for criteria:
California

Please refine your search results, only the first 50 results are displayed.

Anderson Certified Farmers Market
Deschutes Rd @ Hwy 273 (Shasta Outlets Center)
Anderson, California, 96007

Aptos Farmers Market
Soquel Drive, Cabrillo College
aptos, California, 95003

Atwater Village Farmers' Market
3250 Glendale Blvd.
Los Angeles, California, 90039

Beverly Hills Farmers Market
9300 Civic Center Drive
Beverly Hills, California, 90210

Bonsall Certified Farmers Market and Open Air

Figure 3.3: Fresh Food Finder application views

3.2.2 Testing Tool Selection and Description

According to the Systematic Literature Review part of the paper, the top 5 mentioned testing tools among all papers are:

- Android Monkey
- Robotium
- Appium
- MonkeyRunner
- Android GUIRipper

Android GUIRipper is a toolset that is developed by a university and is not freely available, therefore, it can be excluded from the list. Android Monkey performs random events with SDK, therefore, we cannot write any code for automation test because of that it can be excluded from the list as well. According to mining results, Robotium is the used testing tool, thus, it can be a good candidate for the first testing tool. As stated in the literature review results, after Robotium testing tool Appium seems the second used testing tool, therefore, it can be the second candidate for this exploratory work. Additionally, to web search there are other candidates for hybrid testing tools are UIAutomator and Selendroid as well.

In this study, testing tools will be compared against each other under these requirements and results will be explained in the next chapter;

- **Usability**; setup of tools, easy to use, etc.
- **Compatibility;** run among all API levels, configurations, etc.
- **Performance**; test execution time and speed for each test cases
- **Maintainability**; for test cases, written lines of code(LOC)
- **Reliability**; results of the test cases, how many test cases passed or not

Appium

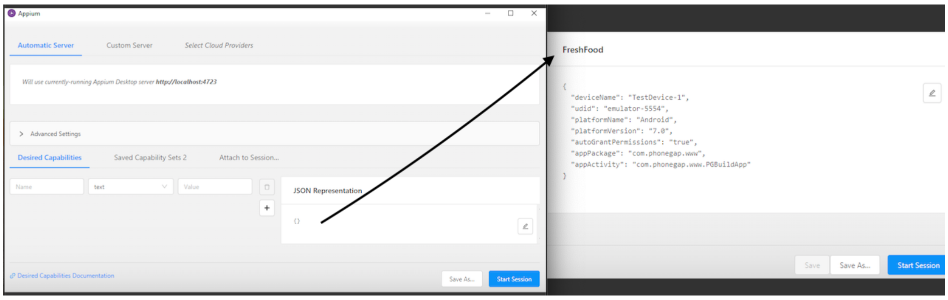
Appium is an open-source automated testing tool for native, mobile web, and hybrid mobile applications both on Android and IOS. For Hybrid Applications, a wrapper around "web-view" is necessary which enables the interaction with web content to provide cross platform features. Importantly, Appium is "cross-platform" it allows you to write tests against multiple platforms (iOS, Android, Windows), using the same API. This enables code reuse between iOS, Android, and Windows test suites. Appium works as client-server architecture which receives connections from a client, listens for commands, executes those commands on mobile device, and responds with an HTTP response with results. In the web page of Appium, there is a very good documentation about usage like actions. Thanks to this documentation, test cases can be created with any desired and supported language such as Java, Python, etc with only using Appium client libraries that provides easy adaptation for everyone[**73**].

In order to setup and then use Appium as a testing tool, there are some prerequisites for the system;

- Java JDK
- Android Studio Android SDK
- ANDROID_HOME and JAVA_HOME environment path configurations
- IntelliJ IDEA or Eclipse or any other Java IDE
- Appium Desktop
- Some other necessary configurations, it needs to be checked from the original Appium web page for the latest updates.

After providing those prerequisites, it is necessary to create a pom.xml with some dependencies such as Selenium, Appium, TestNG, these dependency versions and maven can be found directly on the web.

The last configurations such as device number, platformVersion, appPackage and appActivity names about test device should be done from Appium Desktop application >Desired Capabilities Page as in Figure 3.4. These configurations help to open inspector on Appium in order to test GUI. After all these jobs, test cases can be written and then with Appium can be tested. At the end of each test case, Appium provides a test result report to check the test passed or not.



Will use currently-running Appium Desktop server **http://localhost:4723**

```
{}
```

```
{
  "deviceName": "TestDevice-1",
  "udid": "emulator-5554",
  "platformName": "Android",
  "platformVersion": "7.0",
  "autoGrantPermissions": "true",
  "appPackage": "com.phonegap.www",
  "appActivity": "com.phonegap.www.PGBuildApp"
}
```

Figure 3.4: Desired Capability Configuration in Appium

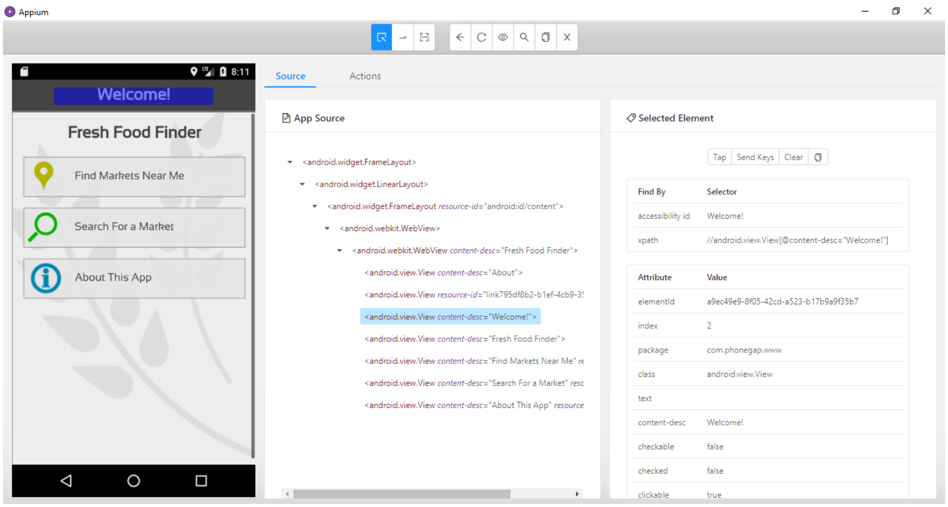
While developing the tests it is necessary to import specific libraries and then create a *driver* object in order to interact with the test device. Thanks to Appium Desktop Tool, all source code can be followed, and the desired element can be traceable in Figure 3.5. Selected element has a specific id, it would be used to reach that element from test case with *findElement* method.

driver.findelementByXPath("//android.view.View[@content-desc="Welcome!"]");

Additionally, it can be possible to prepare test suite under *@BeforeSuite* or *@AfterSuite* tags. In order to start to write test case code, *@Test* tag should be there and the methods should be public. After initializing driver object, this driver object should be initialized as *WEBVIEW* tag in order to test hybrid applications.

driver.context("WEBVIEW_com.phonegap.www.PGBUildApp");

After that, defined methods can be used for specific purpose of test case. Lastly, the developer can run that created a test case with result output which includes test execution time as well.



App Source

```
<android.widget.FrameLayout>
  <android.widget.LinearLayout>
    <android.widget.FrameLayout resource-id="android:id/content">
      <android.webkit.WebView>
        <android.webkit.WebView content-desc="Fresh Food Finder">
            <android.view.View content-desc="About">
            <android.view.View resource-id="link795df8b2-b1ef-4cb9-3
            <android.view.View content-desc="Welcome!">
            <android.view.View content-desc="Fresh Food Finder">
            <android.view.View content-desc="Find Markets Near Me" re
            <android.view.View content-desc="Search For a Market" resc
            <android.view.View content-desc="About This App" resource
```

Selected Element

Find By	Selector
accessibility id	Welcome!
xpath	//android.view.View[@content-desc="Welcome!"]

Attribute	Value
elementId	a9ec49e9-8f05-42cd-a523-b17b9a9f35b7
index	2
package	com.phonegap.www
class	android.view.View
text	
content-desc	Welcome!
checkable	false
checked	false
clickable	true

Figure 3.5: UI Element Detection feature of Appium Desktop Tool

Robotium

Robotium is an open-source Android test automation framework that has full support for native and hybrid applications. Robotium makes it easy to write powerful and robust automatic black-box GUI tests for native and hybrid Android applications. Do not necessary to have a huge knowledge about the application while testing, hence, it captures the features automatically. With the support of Robotium, test case developers can write function, system, and user acceptance test scenarios, spanning multiple Android activities.

Robotium tests can be written where a UI widget is searched for, it is sent an input and assertions can be done afterward. It does not have any desktop tool version or like that. In order to use it downloading a jar file is enough from it's official GitHub account [74]. It is compatible with both Android Studio, IntelliJ, and Eclipse IDE in order to run tests automatically. It officially supports Android API level 8 and up for any Android devices as well. Robotium tests extend an outdated(not working with androidx) *ActivityInstrumentationTestCase2* and use JUnit3 instead of JUnit4 in the test cases. Since Robotium relies on the instrumentation framework, it can only handle interactions within the one application for which the tests have been written.

In order to test the APK file without source code, it is necessary to make some changes. Fresh Food Finder has project files in GitHub but they are developed in Phonegap, therefore it is better to build its APK file to use it Robotium Test. Adobe provides a website[2] for building the APK file. In order to test with it, the APK file needs to have the same certificate signature with the test project. Signature matching is a process that has some compelling steps. In order to sign the application, the following lines of command are used:

jarsigner -keystore /.android/debug.keystore -storepass android - keypass

android Application.apk androiddebugkey zipalign 4 Application.apk TempApplication.apk.
After these commands it is necessary to change the name of TempApplication.apk to Application.apk. If the application is signed, then it is necessary to delete the application sign by deleting META_INF folder of the application. Afterward, the application is designed using the commands presented above. Thus, testing with APK files is possible but not as easy as testing with source code.

When creating a test project using Robotium, it is needed to find the package and launcher activity name of the application under test. Without knowledge about source code, it needs some complicated process to reach that information. UI elements are reachable by their index and texts. However, sometimes Robotium faces problems finding elements by this information. Thus, reaching UI elements by their ID might be more effective. When making a test project utilizing

[2]https://build.phonegap.com

Robotium, it is expected to discover the package and launcher activity class name of the application under test. It was a complicated and tough task without knowing the source code. On the other hand, there was not enough source to learn details of testing Hybrid APK without the source code of the application. It was a very hard process to start testing hybrid apk when it compared with the Appium.

When developing the tests one uses *Solo* class to interact with the test device. It uses set of classes (com.jayway.android.robotium.solo) for testing. Robotium should be defined in gradle file for Android Studio or it should be downloaded and added under lib folder for Eclipse IDE development of testing. This class supports test cases that span over multiple activities. Solo is integrated with the ActivityInstrumentationTestCase2.

During this experience, there were a lot of problems like defining solo class because ActivityInstrumentationTestCase2 is deprecated in the newest version of Android library. Android Studio improved its version many times and latest updates made in September 2019, however, Robotium didn't get any update until the last 3 years, available in Github [3]. Therefore it is normal that there are many bugs and compatibility issues. Even if Robotium seems compatible with every open-source apps, unfortunately, it is not working with hybrid apps that are developed by PhoneGap(Cordova). Hence, Phonegap uses its custom web client instead of creating objects inside of the Webview, therefore, it is not compatible with Robotium. Because of the development type of Fresh Food Finder testing by Robotium study have not been concluded.

UI Automator

UI Automator is a UI testing framework suitable for cross-platform functional testing across the system and installed apps for Android

[3]https://github.com/RobotiumTech/robotium

apps. UI Automator testing framework offers a set of APIs for creating UI tests that execute user applications and system applications interactions. The UI Automator APIs enable you to conduct activities such as opening the Settings menu or the test device app launcher. UI Automator testing framework is suitable for automated testing in a black-box style, where the test code does not depend on the target app's inner execution information.
The features of the UI Automator testing framework are:

- A viewer to inspect layout hierarchy. For more information, see UI Automator Viewer.

- An API to retrieve state information and perform operations on the target device. For more information, see Accessing device state.

- APIs that support cross-app UI testing. For more information, see UI Automator APIs[**75**].

In order to run UI Automator tests, it is necessary to have Android 4.3 (API level 18) or higher. According to the new Android Studio, API level is 29 and the Fresh-Food-Finder API level is 25 and there is no problem for availability problems like in the Robotium. The UI Automator testing framework is an instrumentation-based API and works with the AndroidJUnitRunner test runner.
Before starting the black-box test it is necessary to declare app package name with *getLauncherPackageName()* function to the driver to fetch app to test. However, Cordova doesn't support to give appPackageName as in the native android apps. Furthermore, UIAutomator is able to test hybrid apps but only developed as native WebView component, not developed with CSS and js codes like in Cordova.
For that reason, UI Automator doesn't provide Platform Specific Features for every kind of Android applications, it supports only native widgets like WebView for hybrid applications, so, Fresh-Food-Finder is developed by Cordova, which does not have WebView widget, it means that cannot be tested with UIAutomator as well.

Selendroid

Selendroid is a test automation framework that drives native and hybrid Android applications (apps) and mobile web user interfaces. Selendroid can be used on emulators and real devices and for scaling and parallel testing can be integrated as a node into the Selenium Grid. Selendroid can be used on Mac, Linux, and Windows as well for Native and Hybrid apps[**76**]. To start using Selendroid there are some system requirements;

- Java SDK (minimum 1.6) must be installed and JAVA_HOME configured. IMPORTANT: If JAVA_HOME is pointing to a Java runtime environment, selendroid will produce errors because tools like the jarsigner are not available!
- Latest Android-SDK must be installed and ANDROID_HOME set. If detailed instructions are needed, have a look at this guide.
- If you run selendroid on a 64bit Linux machine, please install:
  sudo dpkg –add-architecture i386
  sudo apt-get update
  sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
- At least one Android virtual device must exist or an Android hardware device must be plugged into the computer.

After meeting these requirements, selendroid-standalone-0.17.0-with-dependencies.jar file should be downloaded to the test PC. In order to use Selendroid for testing, the apk, which should be signed with the same certificate with selendroid-server, should be downloaded as well where the selendroid-standalone server will be launched. Since custom selendroid server will be developed for the test of apk. To launch Selendroid

java -jar selendroid-standalone-0.17.0-with-dependencies.jar -app selendroid- test-app-0.17.0.apk

code should be written in the command-line-prompt. Selendroid-standalone will start a http server on port 4444 and will scan all Android virtual devices (avd) that the user has created (/.android/avd/).

The Android target version and the screen size will be identified[**77**].

After written code which is described in the above, Selendroid server should be started with the dependencies but unfortunately, it didn't start because of the error that the "android command is deprecated. For manual SDK, AVD, and project management, please use Android Studio. For command-line tools, use tools/bin/sdkmanager and tools/bin/avdmanager".

Selendroid has a bug about AVD of Android because Android Studio has a new version 3.2 and in that new version is not compatible with the Selendroid version 0.17. Furthermore the reason is Selendroid didn't upgrade as Android Studio like that the latest date for Selendroid is more than one year. Additionally, Selendroid has open issues about the hybrid test for Cordova-Phonegap crashes #1034[**78**], issue opened date February 2016.

As it is known that Fresh-Food-Finder is a hybrid app, it should be tested with a hybrid feature of Selendroid as finding "WebView" from the window. However, there are some open issues about hybrid tests especially for apps that are developed by Cordova-Phonegap. ın the light of these reasons, Selendroid is not a good testing tool for the Fresh-Food-Finder app.

3.2.3 Test Case Definition

This empirical study is focusing on GUI Testing of Hybrid mobile applications, therefore, it is necessary to draw a border to this functionality about GUI. It is obvious that the GUI concept has very huge subclasses and features, therefore it is better to create a checklist with a high-level perspective to understand the main features of these selected testing tools. Additionally, in order to decide the evaluation criteria and test cases, it is better to identify some requirements for testing frameworks. Fresh-Food-Finder has 3 main features which have to be tested, are described below;

- Find markets directly from the map(according to your current position).
- Categorize his desire about fresh-market such as; State, Food or/and Payment Type.
- Check a selected market direction to go there and map view of the market

According to these main features, detailed test cases should be designed with checking each feature one by one with correct and false inputs. All test cases which will be used to test app as shown in the Table 3.2.

Test Case	Test Case Name
TC1	Find Markets Near Me with in Map View
TC2	Search for a Market by city - Correct input " Birmingham"
TC3	Search for a Market by state - Wrong input " Torino"
TC4	Search for a Market by market name - Correct input "East Lake Farmers Market"
TC5	Search for a Market by market name - Wrong input "Torino markets"
TC6	Search for a Market by using state picker "California"
TC7	Search for a Market by filtering no products and no payments
TC8	Search for a Market by filtering products as "Baked Goods" and no payments
TC9	Search for a Market by filtering no product and "Credit Cards Accepted" in payments
TC10	Search for a Market by filtering products as "Baked Goods" and "Credit Cards Accepted" payments
TC11	Open Map View of the Selected Market
TC12	Getting directions of the Selected Market

Table 3.2: Test Cases

3.2.4 Test Cases Implementation

Appium known that supports many different languages such as Java, Objective-C, JavaScript with node.js, PHP, Ruby, Python, etc. For this thesis study, Java is selected and written test suites run inside IntelliJ IDE with Appium Desktop application. In order to evaluate the behavior of the application, Android Studio a standard Nexus 5 emulator is used. Even a real android device can be used to see the behavior of the application. At the beginning of each test case, necessary configurations dependencies are adjusted like switching from a NativeView to WebView switch in @BeforeSuite;

```
Set<String> con = driver.getContextHandles();
for(String c: con){
    if(c.contains("WEBVIEW")){
          driver.context(c);
          break;
     }
}
```

This BeforeSuite is the same for each test case, as well as AfterSuite. After each test APK turns the original version by uninstalling the apk for the next test, code is in under tag @AfterSuite;

```
public void uninstallApp() throws InterruptedException {
          driver.quit();
     }
```

All results of test cases are shown in the Table 3.3. During testing, some bugs in the application were found which are caused by the availability of the app in the Google Market. Hence, Fresh-Food-Finder is not visible in the Google Market now because of the update on API level 29. On the other hand, the app is working API level 25. It means that *geolocation ()* was working on that time but now it is not working because the API level changed and not adopted it. So, it's broken the last version of the plugin (2.4.3) for Android 8 and for Android 9.

Only TC1 and TC11 is affected on this problem because they need geolocation() function inside of the code. Also, search/filtering feature has a problem that cannot filter by name of the market like in TC4, therefore, those tests have been failed. The rest test cases are passed correctly. Additionally, test execution time very fast also because the test cases are not long, these detailed code pieces can be seen in Appendix.

Test Case	Result	Time(Second)
TC1	Failed	25s
TC2	Passed	29s
TC3	Passed	31s
TC4	Failed	26s
TC5	Passed	29s
TC6	Passed	34s
TC7	Passed	27s
TC8	Passed	32s
TC9	Passed	38s
TC10	Passed	37s
TC11	Failed	38s
TC12	Passed	40s

Table 3.3: Results of Test Cases

For Robotium, UIAutomator, and Selendroid test were not be run because of the problems will be covered in the result section of this chapter.

3.2.5 Test Tools Evaluation and Results

In this section, the results of tools will be evaluated according to testing challenges 2.9 that was mentioned in the challenges chapter. In the literature review part of this thesis, hybrid development and testing tools were described. According to these reviews, hybrid apps can be tested with both Appium, Robotium, UIAutomator, and Selendroid.

It is proven that there are some gaps in this claim because during the exploratory study only Appium performs well with Fresh-Food-Finder app which is developed by PhoneGap. The other testing tools failed because of some compatibility problems such as Availability of API, Platform Specific Components, not update guidelines. The compatibility table ,which are combination of testing challenges with respect to each testing tools, is reported in Table 3.4. These compatibility problems are also pointed in the testing challenges in the previous chapter.

It is the proof that some challenges, which were discussed in literature reviews, occurred within the real practice that was made during this study.

Challenges(L1)	(L2)	Appium	Robotium	UIAutomator	Selendroid
Testing Tool & Frameworks	Tool Standardization (Last Update Date)	August 2019 (v1.14)[**79**]	September 2016[**80**]	March 2015[**81**]	October 2015[**82**]
Testing Tool & Frameworks	Android API Level	>= 18[**83**]	>= 14[**84**]	>= 18[**85**]	10-19[**86**]
Platform Specific Components	Component and widgets	Native WebView & Web Elements	Native WebView	Native WebView	Native WebView
Test Tool Expertise	Lack of Guideliness	Updated	Outdated	Updated	Outdated

Table 3.4: Testing Challenges vs Testing Tools

It is also necessary to mention that, PhoneGap creates an app like a native application without OS-specific components. It does not create the native layout or components of Android and IOS. PhoneGap framework allows developers to use standard web technologies like HTML, CSS, and Javascript for creating cross-platform mobile applications. Therefore, components which are developed in PhoneGap is custom means that widgets are implemented inside of the HTML tags like a web application. It is an advantage that not necessary to develop layouts according to the different OS like IOS, or Android.

Appium is working very well with Hybrid applications developed by PhoneGap because Fresh-Food-Finder app was tested without any problems. It is obvious that Appium is a completely multi-platform automated testing tool that is working at a higher level of abstraction. It means that Appium does not care about the structure of the application like low-level components like a web application. Therefore, it is not looking for a native component to capture the application layout. Appium can reach the WebView Elements by inspecting also directly from a web browser. The only drawback for Appium is the setup process of it, because it requires many different configurations both in a PC system, and especially AppiumDoctor which tries to diagnose and fix common Node, iOS, and Android configuration issues before starting Appium as well. Running test was very basic like capturing all web elements easily thanks to web browser support of Appium. Consequently, Appium works perfectly to capture all WebView elements of the Fresh-Food-Finder app and run all test cases.

Robotium is working limited with Hybrid applications like developed as inside of the WebView component not developed by PhoneGap. Hence, Robotium is able to work with the Native WebView component of android apps instead of Web Elements(HTML) which are developed in PhoneGap. It is the reason that Robotium didn't get any update until September 2016 [**70**]. Additionally, an issue found about Robotium and Cordova compatibility on GitHub[**87**] which is the place of Robotium.jar file. In this issue, it was mentioned that "Cordova uses their custom web client and therefore it is not compatible with Robotium", also this issue closed as "won't fix". In light of this information, it is clear that Robotium cannot test any hybrid app developed by Cordova(PhoneGap). Therefore there is this compatibility problem that occurs during a real experiment on testing Fresh-Food-Finder app.

During the experiment, a simple app that has Native Webview was developed to understand Robotium is able to test any hybrid app or not. The app had appPackageName and activityName inside of the Manifest.xml, additionally, a WebView variable was created to point a real web page. In theoretically, it was a hybrid app that opens in an android device but it had native codes to declare WebView element. After all, Robotium was able to find a WebView element of this simple app and the test was achieved successfully. According to literature reviews, Robotium is able to test hybrid apps that are only developed as a native app with WebView element; not supporting apps developed by Cordova(PhoneGap).

UIAutomator is mainly used for cross-platform functional testing for Android apps by using Java libraries to interact with native elements. Even if it supports black-box testing and compatible API level improvements, it works limited as Robotium because it supports only native WebView element in android applications. It means that UIAutomator does not support the testing of hybrid apps developed by PhoneGap. Hence, Fresh-Food-Finder has HTML web elements instead of native WebView elements like buttons or other widgets as well. Development by Phonegap prevents to test the app with UI Automator.

Selendroid is another test automation framework for cross-platform mobile applications like native and hybrid. Similarly, it has compatibility problems like Robotium. Selendroid supports corresponding API levels between 10 to 19. However, the Fresh-Food-Finder API level is more than 19 and it caused problems on this testing framework. The reason for this API level support is that Selendroid is not updated as new improvements on Android. The latest improvements on Selendroid are more than 3 years and this time difference is very crucial. On the other hand, Selendroid libraries are not improved according to Android ADV libraries, therefore, it failed during the setup phase of the experiment. Additionally, there are no good conflict guidelines to help testers because all documents are also outdated like more than 3 years. According to these problems, even if Selendroid seems one of the cross-platform testing frameworks, it didn't use for this experiment.

At the beginning of the study, there were 4 different cross-platform testing tools/frameworks to test Fresh-Food-Finder hybrid app, but only one of them which is Appium corresponds to the testing activity of the app. Comparison of testing tools concerning development feature of the hybrid apps is shown in the Table 3.5.

Test Tool/ Dev. Tool	Appium	Robotium	UIAutomator	Selendroid
Native Webview	Yes	Yes	Yes	Yes
Cordova(PG)	Yes	No	No	No
Titanium[**89**]	Yes	No	No	No
Xamarin[**90**]	Yes	No	No	No
React-Native[**91**]	Yes	No	No	No

Table 3.5: Compatibility Table of Testing Tools vs Development Approaches

As a result of all-out findings, Appium is the best tool for testing hybrid applications developed by PhoneGap, and other development frameworks as well. On the other hand, Robotium, UI Automator, and Selendroid didn't work well apps developed because of the compatibility problems which are covered in the challenges part of this study.

Chapter 4

Conclusion and Future Work

Hybrid apps can be developed with many frameworks but all of them expose issues that are discussed in the present thesis. Firstly, many literature reviews were collected in order to answer three research questions which were discussed in the second chapter. The most used open-source development tool is Apache Cordova(known as Phone-Gap) and the testing tool is Appium. Furthermore, these development and testing tools have many challenges. Mainly "User Interface" is the more significant difficulty for development challenges perspective, on the other hand, "Platform Specific Features" is the most critical challenge belongs to testing tools.

After literature review results, this study focused on Source Repository Analysis which is explained in the third chapter. Source Repository analysis have done by collected 9872 open-source projects from GitHub. During this analysis, distribution of development and testing tools for hybrid mobile has been evaluated. Unfortunately, the result is not sufficient to support literature review results. According to results of this mining shows that Robotium is the most used tool which are developed by Apache Cordova(Phonegap) Development Framework. Since, only 0.009% of projects have been pointed both development and testing tools which are described in the literature review results.

Lastly, Exploratory Study have been conducted because Source Repository Analysis was not sufficient. An open-source hybrid app (Fresh-Food-Finder) developed by PhoneGap was found from GitHub to be tested among selected several testing tools. Based on this study, Hybrid applications which are developed by PhoneGap are totally different than native android applications because they are implemented with HTML, CSS, Javascript instead of Native Android style with Java and inside WebView component. According to the development framework selected and the application type, there are several alternative testing tools that can be leveraged. Referring to literature review results and source repository analysis; Appium, Robotium, UIAutomator, and Selendroid are the selected tools for this experiment. Even if taking only one open-source hybrid application from GitHub and tried to be tested with these tools, some compatibility problems have been encountered for all tools except Appium. Briefly, these problems are categorized;

- Outdated tools, and libraries
- API availability
- Specific Component and Widget declaration like web elements(HTML)
- Lack of guidelines and official tutorials

These problems also appeared in the challenges mentioned in the literature review results. This demonstrates that some issues on one random app happens on these challenges happens. This is a result even more problematic mobile app are not tested sometimes and cross-platform apps are more problematic than others. The results of the thesis confirm that the fast-growing technology and the diversity of development patterns for Android should be coupled with parallel advances from the testing community to ensure better testability for hybrid mobile applications.

As future work, there may be some improvements on this study. It can be updated cross-platform test automation tools which do not have compatibility problems like it appears in this study. Additionally good and compatible guidelines can be developed for helping developers to avoid common issues for hybrid app development and testing. Furthermore, the efforts from researchers in the area of mobile app testing should be focused on finding ways to overcome compatibility, layout and widget specification for WebView, documentation, etc problems which are described in the taxonomy of challenges.

Bibliography

[1] Statista,*Number of mobile phone users worldwide 2015-2020 Statista* [Online]. Available:https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/.[Accessed:31-Mar-2019].

[2] Statista, 2019, *Annual number of mobile app downloads worldwide 2022 Statistic* [Online]. Available:https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.[Accessed:31-Mar-2019].

[3] Ahmad, A., Li, K., Feng, C., Asim, S., Yousif, A. and Ge, S.,*An Empirical Study of Investigating Mobile Applications Development Challenges*, IEEE Access,2018, pp.17711-17728.

[4] Ali, M. and Mesbah, A.,*Mining and characterizing hybrid apps. Proceedings of the International Workshop on App Market Analytics*, WAMA 2016.

[5] Malavolta, I., Ruberto, S., Soru, T. and Terragni, V.,*Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation*, 2015 2nd ACM International Conference on Mobile Software Engineering and Systems.

[6] Alamri, Hammoudeh & Mustafa, Balsam.,*Software Engineering Challenges in Multi-Platform Mobile Application Development*, 2014 2Advanced Science Letters. 20. 10.13140/2.1.5122.7523.

[7] Sauce Labs.,*Native vs. Web vs. Hybrid Mobile Apps: Testing Tools and Techniques*, 2019 [online]. Available:https://saucelabs.com/blog/native-vs-web-vs-hybrid-mobile-apps-testing-tools-and-techniques/. [Accessed 20 Feb. 2019].

[8] Medium.,*App Development Decisions: Native, Web or Hybrid? – Imaginovation – Medium.*, 2019 [online]. Available:https://medium.com/@Imaginovation/app-development-decisions-native-web-or-hybrid-31c103f9b4e1/. [Accessed 20 Feb. 2019].

[9] T. Gronli, J. Hansen, G. Ghinea and M. Younas.,2014*Mobile Application Platform Heterogeneity: Android vs Windows Phone vs IOS vs Firefox OS*, 2014 IEEE 28th International Conference on Advanced Information Networking and Applications. Available: 10.1109/aina.2014.78

[10] C. Rahul Raj and Seshu Babu Tolety,*A study on approaches to building cross-platform mobile applications and criteria to select appropriate approach*,2012 Annual IEEE India Conference (INDICON), 2012. Available: 10.1109/indcon.2012.6420693

[11] S. Xanthopoulos and S. Xinogalos,*A comparative analysis of cross-platform development approaches for mobile applications*, Proceedings of the 6th Balkan Conference in Informatics on - BCI '13, 2013. Available: 10.1145/2490257.2490292

[12] K. Haller, *Mobile Testing* ACM SIGSOFT Software Engineering Notes, vol. 38, no. 6, pp. 1-8, 2013. Available: 10.1145/2532780.2532813

[13] J. Gao, X. Bai, W. Tsai, and T. Uehara, *Mobile Application Testing: A Tutorial* Computer, vol. 47, no. 2, pp. 46-55, 2014. Available: 10.1109/mc.2013.445

[14] H. Muccini, A. Di Francesco and P. Esposito, *Software testing of mobile applications: Challenges and future research directions*

2012 7th International Workshop on Automation of Software Test (AST), 2012. Available: 10.1109/iwast.2012.6228987

[15] P. Tramontana, D. Amalfitano, N. Amatucci and A. Fasolino, *Automated functional testing of mobile applications: a systematic mapping study* Software Quality Journal, vol. 27, no. 1, pp. 149-201, 2018. Available: 10.1007/s11219-018-9418-6

[16] G. Bae, G. Rothermel, and D.-H. Bae, *Comparing model-based and dynamic event-extraction based GUI testing techniques: An empirical study*, Journal of Systems and Software, vol. 97, pp. 15–46, Nov. 2014.

[17] Myers, B.A., *User interface software tools* ACM Trans. on Comput.-Hum. Interact., 2(1):64–103, 1995.

[18] P. Brooks, B. Robinson and A. Memon, *An Initial Characterization of Industrial Graphical User Interface Systems*, 2009 International Conference on Software Testing Verification and Validation, 2009. Available: 10.1109/icst.2009.11

[19] P. Aho and T. Vos, *Challenges in Automated Testing Through Graphical User Interface*, in 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2018.

[20] E. H. Marinho and R. F. Resende, *Native and Multiple Targeted Mobile Applications*,in Computational Science and Its Applications – ICCSA 2015, Springer International Publishing, 2015, pp. 544–558.

[21] Z. Liu, Y. Hu, and L. Cai, *Software Quality Testing Model for Mobile Application*, in Mobile Web Information Systems, Springer International Publishing, 2014, pp. 192–204.

[22] Glaser, B.G. 1992. Basics of Grounded Theory Analysis: Emergence vs Forcing. Sociology Press.

[23] K. Stol, P. Ralph and B. Fitzgerald, "Grounded theory in software engineering research", Proceedings of the 38th International Conference on Software Engineering - ICSE '16, 2016. Available: 10.1145/2884781.2884833

[24] Z. Zhai, B. Cheng, M. Niu, Z. Wang, Y. Feng and J. Chen, "An end-user oriented tool suite for development of mobile applications", Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016, 2016. Available: 10.1145/2970276.2970279

[25] M. Pichiliani and C. Hirata, "Adaptation of Single-user Multi-touch Components to Support Synchronous Mobile Collaboration", Mobile Networks and Applications, vol. 19, no. 5, pp. 660-679, 2014. Available: 10.1007/s11036-014-0512-0

[26] M. Lachgar and A. Abdali, "Generating Android graphical user interfaces using an MDA approach", 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), 2014. Available: 10.1109/cist.2014.7016598

[27] J. Giron, S. Mendoza and C. Torres-Huitzil, "Mechanism for dynamic deployment of plastic mobile cross-platform user interfaces", 2011 8th International Conference on Electrical Engineering, Computing Science and Automatic Control, 2011. Available: 10.1109/iceee.2011.6106613

[28] A. Nestor Ribeiro and C. Rogério Araújo, "An Automated Model Based Approach to Mobile UI Specification and Development", Lecture Notes in Computer Science, pp. 523-534, 2016. Available: 10.1007/978-3-319-39510-4_48

[29] M. Martinez and S. Lecomte, "Towards the Quality Improvement of Cross-Platform Mobile Applications", 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2017. Available: 10.1109/mobilesoft.2017.30

[30] Y. Chang and S. Oh, "A study on the development of one source multi use cross-platform based on zero coding", Multimedia Tools and Applications, vol. 74, no. 7, pp. 2219-2235, 2014. Available: 10.1007/s11042-014-1886-5

[31] X. Shi and W. Zhang, "Qt-based mobile application GUI style for smart phone operating system", 2010 IEEE International Conference on Software Engineering and Service Sciences, 2010. Available: 10.1109/icsess.2010.5552446

[32] G. Mesfin, G. Ghinea, D. Midekso, and T.-M. Grønli, “Evaluating Usability of Cross-Platform Smartphone Applications,” Mobile Web Information Systems Lecture Notes in Computer Science, pp. 248–260, 2014.

[33] M. Palmieri, I. Singh, and A. Cicchetti, “Comparison of cross-platform mobile development tools,” 2012 16th International Conference on Intelligence in Next Generation Networks, 2012.

[34] S. Chadha, A. Byalik, E. Tilevich, and A. Rozovskaya, “Facilitating the development of cross-platform software via automated code synthesis from web-based programming resources,” Computer Languages, Systems & Structures, vol. 48, pp. 3–19, 2017.

[35] Gokhale, P. and Singh, S. (2014). Multi-platform strategies, approaches and challenges for developing mobile applications. 2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA).

[36] C.-K. Diep, Q.-N. Tran, and M.-T. Tran, “Online model-driven IDE to design GUIs for cross-platform mobile applications,” Proceedings of the Fourth Symposium on Information and Communication Technology - SoICT 13, 2013.

[37] I. Dalmasso, S. Datta, C. Bonnet and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools", 2013 9th International Wireless Communications

and Mobile Computing Conference (IWCMC), 2013. Available: 10.1109/iwcmc.2013.6583580

[38] W. Yang, M. Prasad and T. Xie, "A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications", Fundamental Approaches to Software Engineering, pp. 250-265, 2013. Available: 10.1007/978-3-642-37057-1_19

[39] C. Hu and I. Neamtiu, "A GUI bug finding framework for Android applications", Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11, 2011.

[40] D. Amalfitano, A. Fasolino and P. Tramontana, "A GUI Crawling-Based Technique for Android Mobile Application Testing", 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011.

[41] D. Amalfitano, A. Fasolino, P. Tramontana, S. De Carmine and G. Imparato, "A toolset for GUI testing of Android applications", 2012 28th IEEE International Conference on Software Maintenance (ICSM), 2012.

[42] I. Salihu, R. Ibrahim, B. Ahmed, K. Zamli and A. Usman, "AMOGA: A Static-Dynamic Model Generation Strategy for Mobile Apps Testing", IEEE Access, vol. 7, pp. 17158-17173, 2019. Available: 10.1109/access.2019.2895504.

[43] S. Anand, M. Naik, M. Harrold and H. Yang, "Automated concolic testing of smartphone apps", Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12, 2012.

[44] S. Linan, L. Bello-Jimenez, M. Arevalo and M. Linares-Vasquez, "Automated Extraction of Augmented Models for Android Apps", 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018.

[45] Y. Baek and D. Bae, "Automated model-based Android GUI testing using multi-level GUI comparison criteria", Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016, 2016.

[46] S. Singh, R. Gadgil and A. Chudgor, "Automated Testing of Mobile Applications using Scripting Technique: A Study on Appium", 2014 INPRESSCO International Journal of Current Engineering and Technology, Vol.4, No.5 (Oct-2014)

[47] C. Hu and I. Neamtiu, "Automating gui testing for android applications", Proceeding of the 6th international workshop on Automation of software test - AST '11, 2011.

[48] M. Hesenius, T. Griebe, S. Gries and V. Gruhn, "Automating UI tests for mobile applications with formal gesture descriptions", Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services - MobileHCI '14, 2014. Available: 10.1145/2628363.2628391.

[49] D. Amalfitano, V. Riccio, N. Amatucci, V. Simone and A. Fasolino, "Combining Automated GUI Exploration of Android apps with Capture and Replay through Machine Learning", Information and Software Technology, vol. 105, pp. 95-116, 2019. Available: 10.1016/j.infsof.2018.08.007.

[50] Meiliana, I. Septian, R. Alianto and Daniel, "Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study", Procedia Computer Science, vol. 135, pp. 736-748, 2018. Available: 10.1016/j.procs.2018.08.211.

[51] M. Joorabchi, M. Ali and A. Mesbah, "Detecting inconsistencies in multi-platform mobile apps", 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), 2015. Available: 10.1109/issre.2015.7381838.

[52] M. Linares-Vasquez, "Enabling Testing of Android Apps", 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015. Available: 10.1109/icse.2015.242.

[53] "Robotium", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Robotium. [Accessed: 28- Apr- 2019].

[54] N. us Saqib and S. Shahzad, "Functionality, Performance, and Compatibility Testing: A Model Based Approach", 2018 International Conference on Frontiers of Information Technology (FIT), 2018. Available: 10.1109/fit.2018.00037.

[55] W. Choi, G. Necula and K. Sen, "Guided GUI testing of android apps with minimal restart and approximate learning", ACM SIGPLAN Notices, vol. 48, no. 10, pp. 623-640, 2013. Available: 10.1145/2544173.2509552.

[56] M. Linares-Vasquez, M. White, C. Bernal-Cardenas, K. Moran and D. Poshyvanyk, "Mining Android App Usages for Generating Actionable GUI-Based Execution Scenarios", 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015. Available: 10.1109/msr.2015.18

[57] "monkeyrunner | Android Developers", Android Developers, 2019. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner. [Accessed: 28- Apr- 2019].

[58] D. Amalfitano, A. Fasolino, P. Tramontana, B. Ta and A. Memon, "MobiGUITAR: Automated Model-Based Testing of Mobile Apps", IEEE Software, vol. 32, no. 5, pp. 53-59, 2015. Available: 10.1109/ms.2014.55

[59] I. Bayley, D. Flood, R. Harrison and C. Martin, "MobiTest:A Cross-Platform Tool for Testing Mobile Applications", IARIA, no. 978-1-61208-230-1, p. ICSEA 2012 : The Seventh International Conference on Software Engineering Advances, 2012

[60] Y. Ma, Y. Huang, Z. Hu, X. Xiao and X. Liu, "Paladin", Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications - HotMobile ’19, 2019. Available: 10.1145/3301293.3302363

[61] A. Sadeghi, R. Jabbarvand and S. Malek, "PATDroid: permission-aware GUI testing of Android", Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017, 2017. Available: 10.1145/3106237.3106250

[62] K. Mao, M. Harman and Y. Jia, "Sapienz: multi-objective automated testing for Android applications", Proceedings of the 25th International Symposium on Software Testing and Analysis - ISSTA 2016, 2016. Available: 10.1145/2931037.2931054

[63] G. Shah, P. Shah and R. Muchhala, "Software Testing Automation using Appium",INPRESSCO, International Journal of Current Engineering and Technology, vol. 4, no. 5, 2014. Available: http://inpressco.com/category/ijcet.

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

[65] S. Gunasekaran and V. Bargavi, "Survey on Automation Testing Tools for Mobile Applications", International Journal of Advanced Engineering Research and Science (IJAERS), vol. -2, no. -11, 2015.

[66] I. Salihu and R. Ibrahim, "Systematic Exploration of Android Apps’ Events for Automated Testing", Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media - MoMM ’16, 2016. Available: 10.1145/3007120.3011072

[67] A. HUSSAIN, H. RAZAK and E. MKPOJIOGU, "THE PERCEIVED USABILITY OF AUTOMATED TESTING TOOLS FOR MOBILE APPLICATIONS", Journal of Engineering Science

and Technology Special Issue on ISSC'2016, no. 42017, pp. 86 - 93, 2017.

[68] I. Malavolta, S. Ruberto, T. Soru and V. Terragni, "End Users' Perception of Hybrid Mobile Apps in the Google Play Store", 2015 IEEE International Conference on Mobile Services, 2015. Available: 10.1109/mobserv.2015.14

[69] L. Corral, A. Janes and T. Remencius, "Potential Advantages and Disadvantages of Multiplatform Development Frameworks–A Vision on Mobile Environments", Procedia Computer Science, vol. 10, pp. 1202-1207, 2012. Available: 10.1016/j.procs.2012.06.173

[70] M. Joorabchi, A. Mesbah and P. Kruchten, "Real Challenges in Mobile App Development", 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 2013. Available: 10.1109/esem.2013.9

[71] S. Chen, L. Fan, T. Su, L. Ma, Y. Liu and L. Xu, "Automated Cross-Platform GUI Code Generation for Mobile Apps", 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile), 2019. Available: 10.1109/ai4mobile.2019.8672718

[72] R. Coppola, M. Morisio, M. Torchiano and L. Ardito, "Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes", Empirical Software Engineering, vol. 24, no. 5, pp. 3205-3248, 2019. Available: 10.1007/s10664-019-09722-9 [Accessed 3 September 2019].

[73] "Introduction - Appium", Appium.io, 2019. [Online]. Available: http://appium.io/docs/en/about-appium/intro/?lang=en. [Accessed: 03- Sep- 2019].

[74] https://github.com/RobotiumTech/robotium

[75] "UI Automator | Android Developers", Android Developers, 2019. [Online]. Available:

https://developer.android.com/training/testing/ui-automator. [Accessed: 21- Sep- 2019].

[76] D. Dary, "Selendroid: Selenium for Android", Selendroid.io, 2019. [Online]. Available: http://selendroid.io. [Accessed: 21- Sep- 2019].

[77] D. Dary, "Selendroid: Getting started", Selendroid.io, 2019. [Online]. Available: http://selendroid.io/setup.html. [Accessed: 21- Sep- 2019].

[78] https://github.com/selendroid/selendroid/issues/1034

[79] https://github.com/appium/appium/releases

[80] https://github.com/RobotiumTech/robotium

[81] https://developer.android.com/jetpack/androidx/releases/archive/test

[82] https://github.com/selendroid/selendroid/releases

[83] http://appium.io/docs/en/about-appium/platform-support/

[84] https://github.com/RobotiumTech/robotium/search?q=API&unscoped_q=A

[85] https://developer.android.com/training/testing/ui-automator

[86] https://github.com/selendroid/selendroid/search?q=
19&unscoped_q=19

[87] https://github.com/RobotiumTech/robotium/issues/757

[88] https://github.com/selendroid/selendroid/issues?utf8=&q=
is%3Aissue+is%3
Aopen+cordova

[89] https://medium.com/adamtarmstrong/build-test-deploy-a-titanium-cross-platform-app-with-fastlane-3099ae01a07f

[90] https://forums.xamarin.com/discussion/26942/experiences-with-automated-ui-testing-with-xamarin-for-android

[91] https://medium.com/@ronak8036/react-native-testing-tools-f38d715adb57

Appendix

```
/**
 * Test Case 1 – Find Markets Near Me with in Map View
 */
driver.findElementById("nearMe").click();
String nearme = driver.findElementById("contentRoot")
.getText();
nearme.contains("location");


/**
 * Test Case 2 – Search for a Market by city –
 Correct input
 *"Birmingham"
 */
driver.findElementById("search").click();
driver.findElementById("search\_searchPhrase").sendKeys
("birmingham");
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);


/**
 * Test Case 3 – earch for a Market by state –
 Wrong input
 *"Torino"
 */
driver.findElementById("search").click();
driver.findElementById("search\_searchPhrase").sendKeys
("torino");
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertEquals(size, 0);


/**
 * Test Case 4 - Search for a Market by market name -
 Correct
 *input "East Lake Farmers Market
 */
driver.findElementById("search").click();
driver.findElementById("search\_searchPhrase").sendKeys
("East Lake Farmers Market");
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertEquals(size, 1);


/**
 * Test Case 5 - Search for a Market by market name -
 Wrong
 input "Torino markets"
 */
driver.findElementById("search").click();
driver.findElementById("search\_searchPhrase").sendKeys
("Torino Market");
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
```

```
Assert.assertEquals(size, 0);


/**
 * Test Case 6 - Search for a Market by using
 state picker
 *"California"
 */
driver.findElementById("search").click();
Select states = new Select(driver.findElementById
("search\_state"));
states.selectByIndex(4);
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);


/**
 * Test Case 7 - Search for a Market by filtering
 no products
 *and no payments
 */
driver.findElementById("search").click();
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);


/**
 * Test Case 8 - Search for a Market by filtering
 products
 *as "Baked Goods" and no payments
 */
driver.findElementById("search").click();
driver.findElementById("search\_bakedGoods").click();
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);


/**
 * Test Case 9 - SSearch for a Market by filtering no
 product and "Credit Cards Accepted" in payments
 */
driver.findElementById("search").click();
Point point = driver.findElementById("search\_credit").
getLocation();
TouchAction action1 = new TouchAction(driver);
action1.press(PointOption.point(816, 1711))
.moveTo(PointOption
.point(816,0)).release().perform();
driver.manage().timeouts().implicitlyWait(2,
TimeUnit.SECONDS);
driver.findElementById("search\_credit").click();
TouchAction action2 = new TouchAction(driver);
action2.press(PointOption.point(816, 300))
.moveTo(PointOption
.point(816,1811)).release().perform();
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);


/**
 * Test Case 10 - Search for a Market by filtering
 products
 *as "Baked Goods" and "Credit Cards Accepted" payments
 */
```

```
driver.findElementById("search").click();
driver.findElementById("search\_bakedGoods").click();
Point point = driver.findElementById("search\_credit")
.getLocation();
TouchAction action1 = new TouchAction(driver);
action1.press(PointOption
.point(816, 1711)).moveTo(PointOption.point(816,0))
.release().perform();
driver.manage().timeouts().implicitlyWait(2,
TimeUnit.SECONDS);
driver.findElementById("search\_credit").click();
TouchAction action2 = new TouchAction(driver);
action2.press(PointOption.point(816, 300))
.moveTo(PointOption
.point(816,1811)).release().perform();
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView").
findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);



/**
 * Test Case 11 - Open Map View of the Selected Market
 */
driver.findElementById("search").click();
driver.findElementById("search\_flowers").click();
Point point = driver.findElementById("search\_credit")
.getLocation();
TouchAction action1 = new TouchAction(driver);
action1.press(PointOption.point(816, 1711))
.moveTo(PointOption
.point(816,0)).release().perform();
driver.manage().timeouts().implicitlyWait(2,
TimeUnit.SECONDS);
driver.findElementById("search\_credit").click();
```

```
TouchAction action2 = new TouchAction(driver);
action2.press(PointOption.point(816, 300))
.moveTo(PointOption
.point(816,1811)). release().perform();
driver.findElementById("searchButton").click();
int size = driver.findElementById("searchResultsView")
.findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);
driver.findElementById("searchResultsView").findElement
(By. cssSelector
("ul > li:nth-child(1)")).click();
driver.findElementById("marketDetailsView").findElement
(By.className("button")).click();



/**
 * Test Case 12 - Getting directions of the
 Selected Market
 */
driver.findElementById("search").click();
driver.findElementById("search\_flowers").click();
Point point = driver.findElementById("search\_credit")
.getLocation();
TouchAction action1 = new TouchAction(driver);
action1.press(PointOption.point(816, 1711))
.moveTo(PointOption
.point(816,0)). release().perform();
driver.manage().timeouts().implicitlyWait(2,
TimeUnit.SECONDS);
driver.findElementById("search\_credit").click();
TouchAction action2 = new TouchAction(driver);
action2.press(PointOption
.point(816, 300)).moveTo(PointOption.point(816,1811))
.release().perform();
driver.findElementById("searchButton").click();
```

```
int size = driver.findElementById("searchResultsView")
.findElement(By.tagName("ul")).getSize().height;
Assert.assertNotEquals(size, 0);
driver.findElementById("searchResultsView").findElement
(By. cssSelector("ul > li:nth-child(1)")).click();
driver.findElementById("marketDetailsView").findElements
(By.className("button")).get(1).click();
```