Automated Testing of Hybrid Mobile Apps in Multiple Settings

Abstract*—*Background: **Currently, mobile devices are available in several formats; their applications can be grouped into native, Web and hybrid apps. Hybrid apps use Web technologies namely, HTML, CSS and Javascript, and are able to access device native capabilities. The hybrid apps are developed with features that allow their cross-platform execution, but the mechanisms to automatically test them are not. Therefore, different test scripts are needed for each platform.** ***Objective:*** **We aim to introduce mechanisms to automate testing of hybrid mobile apps. In particular, we provide support for generating a single test script capable of testing an hybrid app running in multiple settings. This script can also be used to detect inconsistencies between different settings.** ***Methods:*** **Our approach consists in selecting two devices, one running Android and other iOS. A model to express the test cases and query expressions are defined to select GUI elements for generation a single test script. The test script can be run in multiple settings. The status of each setting is recorded and compared in order to identify inconsistencies between them. Our approach has been implemented in a prototype tool.**

1. Introduction

Currently mobile devices are part of the daily life and are available in various formats, such as smartphones, tablets, and wearables. They are equipped with powerful processors, large storage capacity and various sensors [1]. Modern operating systems (OSs) control the hardware of those devices. In the survey from International Data Corporation (IDC) [2] about the Market Share in the use of mobile OSs, Android [3] and Apple iOS [4] platforms were the most consumed in 2015. In other survey, Gartner [5] introduced sales of smartphones in the first quarter of 2016; the Android OS was the market leader, followed by Apple's iOS. Those modern OSs serve as platform for execution of a wide variety of software called mobile apps. The Statista site offers Statistics of July 2015 [6] and 2016 [7] on the number of apps available for download in the main distribution stores. Android has the largest number of apps available to its users, also followed by iOS.

The development of mobile apps can be classified in three groups: native apps, browser-based Web apps and hybrid apps [8], [9]. Native apps are developed using the Mobile OS Software Developement Kit (SDK), enabling direct access to the functions of the device as well as the OS itself. Web apps are developed with technologies used in building software for the Web like HTML5 [10], CSS3 [10] and Javascript [10]. They are stored on a Web server, run under a browser and they do not have access to advanced features of the mobile OS. Finally, the hybrid apps combine common resources of Web apps like HTML5, CSS3, Javascript and with direct support to OS native Application Programming Interfaces (APIs), such as native apps. The app is divided into two parts, a native one known as WebView, responsible for performing Web resources contained in the second part [11]. Some commercial and open source frameworks provide support to the development of hybrid apps, such as Cordova [12], Phonegap [13], Sencha Touch [14], IONIC [15], Intel XDK [16] and AppBuilder [17]. In a previous study, it was identified that the Cordova is the basis for all other related frameworks [18].

Hybrid apps stand out by their run in various OSs (cross-platform), eliminating the need to rewrite the app to meet a set of different devices. This feature introduces a challenge in testing due to variability of multiple devices configurations on the market [1], [19], [20]. To illustrate the variability in the context of mobile devices, Figure 1 uses a Feature Model (FM) to represent all features of a fictional set of mobile devices. The model is a hierarchical representation of the characteristics and relations among them.

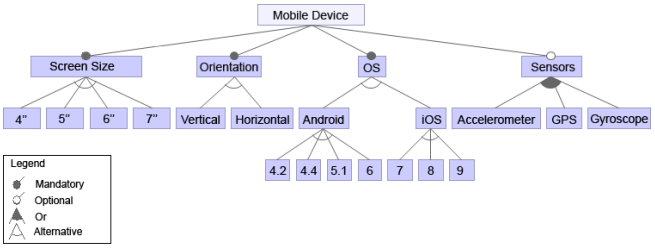

Figure 1. Variability of characteristics of a mobile device

Each product represents a configuration that meets to be tested. This paper defines multiple settings as variabilities (features that vary in relation to various products) of mobile devices with different settings, such as OS, screen size, several types of sensors and hardware. The diversity to test

apps for multiple settings is a challenge. The current testing frameworks and tools do not provide the same level of support in different platforms to test features involving mobility, location services, sensors, different types of gestures and data entries [20]. In addition, the test in a single device does not guarantee correct operation in other [21].

In the scenario of cross-platform apps, the use of hybrid apps is considered a visible solution to avoid the fragmentation in various OSs [22], since their main feature is the ability of running on multiple platforms. However, the testing of hybrid apps can be more challenging, since, in addition to the execution of the tests to identify faults inserted during the development phase, tests are needed to find faults of inconsistency from multiplatform app feature. Faults related to compatibility of the execution of the same app on different platforms can arise [23]. The app can work correctly on platform X and fails on platform Y. Moreover, the processing of HTML, CSS and Javascript among the different WebView accompanying mobile platforms can be slightly different and also cause a failure in the app [24]. An inconsistency fault can be defined as any difference in the behavior of the app execution on multiple devices from multiple settings [23].

In this context, this study aims to, first, provide mechanisms to automate the testing of hybrid mobile apps, supporting test script generation able to test the same app running in multiple settings. Second, using the automated test previously, identify inconsistencies arising from the cross-platform feature of this type of app.

This paper is structured as follows: Section 2 motivates the research problem. Section 3 provides an overview of our approach to test hybrid mobile apps. Section 4 introduces our prototype tool and the experimental evaluation is shown in Section 5. Section 6 presents some related work. Section 7 summarizes conclusions and future work.

2. Motivating Problems

The Graphical User Interface (GUI) of hybrid app is built using HTML elements and during its running it is transformed into an XML structure by the mobile platform, mapping each interface element. This structure differs between Android and iOS platforms, as shown in Figure 2. In a previous study [18], the comparative of such a structure of the app among different versions of the same platform (Android 5.1 vs Android 6.0.1 and iOS 7.1 vs iOS 9.3), has also shown differences. A brief mapping between HTML and XML GUI elements generated by the Android and iOS platforms is presented in Table 1.

The XML structure nodes are composed of key attributes that contain descriptive information of GUI elements and are viewed by users of the app. For Android platform such identified attributes were *"resource-id"*, *"content-desc"* and *"text"*, while for the iOS platform the attributes are *"name"*, *"label"* and *"value"*. The list of HTML attributes and their equivalents in the respective platforms were mapped and presented in Table 2.

Figure 2. HTML Element mapping for Android and iOS GUI element (XML)

Table 1. MAPPING HTML ELEMENTS TO XML GUI ELEMENTS

HTML Element Type	Android Element	iOS Element
input button	android.widget.Button	UIAButton
input submit	android.widget.Button	UIAButton
div	android.widget.View	UIAStaticText
span	android.widget.View	UIAStaticText
label	android.widget.View	UIAStaticText
select	android.widget.Spinner	UIAElement
input text	android.widget.EditText	UIATextField
textarea	android.widget.EditText	UIATextField
a (anchor)	android.widget.View	UIALink

Element selectors are used in automated GUI testing of mobile apps. Selectors are "patterns" or "models" which provide mechanisms to locate elements (or nodes) in computational structures, such as XML or HTML [25]. A computational mechanism for selecting XML elements are query expressions like XPath[1], enabling automated GUI testing.

It is possible to observe when reviewing Figure 2 that the manufacturers of the platforms do not define an XML matched structure, and such difference reflects negatively on the test activities for cross-platform apps. The app is developed with features that enable its cross-platform execution, but the mechanisms to test it are not. In this way, different test scripts are required, each one with appropriate selectors and given the specificity of each platform and their versions. To illustrate, the XPath selector required to select the highlighted element in Figure 2 (button "Search for a market") suitable for Android 6 is ***//android.view.View "[@content-desc = ' Search For a Market ']"*** and for iOS 9 is ***"//UIAApplication [1]/UIAWindow [1]/UIAScrollView [1]/UIAWebView [1]/UIALink [2]"***.

Assuming that the hybrid app is the same for different platforms and the test is not, the mapping showed at the beginning of this Section, allows to implement mechanisms of automated tests and reuse them in a set of multiple devices settings, thus avoiding rewritte test scripts. The idea is to keep a single test script for the app, independent from the execution environment.

Other problem of hybrid apps is related to incompatibil-

1. XPath is a query language for selecting elements (nodes) in computational structures named by tree which represent XML documents [26].

Table 2. MAPPING HTML ATTRIBUTES TO XML GUI ATTRIBUTES

HTML Attributes	Android	iOS
name	-	-
id	resource-id	-
value	content-desc	name, label, value (input text, textarea)
title	content-desc	name
class	-	-
alt	content-desc	name, value

ity between the versions of the same app in multiple settings. Some records that show such inconsistencies can be found on BugTracker of hybrid app Moodle Mobile [27]:

- The scroll bar in specific function related to the content of the app is not displayed in the iOS.
- The button to delete one of the messages in the iOS is viewed on a high position on the screen.
- A non-expected edge is viewed in Android.

3. Approach Overview

Our approach defines a mechanism to automate the testing of hybrid apps and the construction of a test script able to test the same app running in multiple settings. Morover, we aim to identify inconsistencies originated from cross-platform feature of this class of app. The approach consists of four steps. The first step (Figure 3-a, Section 3.1) is based on the selection of two mobile devices of reference, one running the Android OS, called *CRef1* and other running iOS, called *CRef2*. The second step (Figure 3-b, Section 3.2) consists of the definition of a model to express the test cases, the sequence of testable events of the app and the construction of expressions for GUI elements selection. A test case consists of a specification of test input values, conditions of running and expected results, designed to achieve a specific goal, such as identifying a fault, force a path to be traversed within the software or ensure compliance with the requirements of the software [28]. The third step (Figure 3-c, Section 3.3) proposes a single test script generation, matched with multiple devices settings. And finally, the fourth step (Figure 3-d, Section 3.4) consists in the execution of the test script in multiple settings, followed by recovery and comparison of their respective status to identify inconsistencies between the multiple settings. An overview of our approach is illustrated in Figure 3 and in the sections that follow its steps are detailed.

3.1. Device selection

Step 1 consists of selecting two mobile devices, an Android and an iOS, based on popularity among users of the platform. Some studies ([29], [30]) propose the choice of mobile devices for testing of apps based on their popularity of use. Such devices are named as reference configuration (*CRef1* and *CRef2*). In addition, a hybrid mobile app under test *(AUT)* should be installed on those devices.

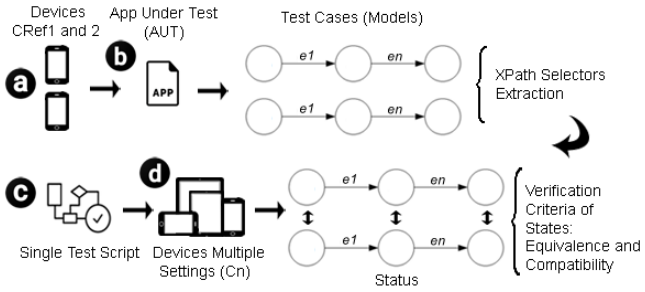

Figure 3. Approach Overview.

3.2. GUI element selection and Test Model Definition

Step 2 is the definition of a model to express the test cases for an app. A test model represents the sequence of events simulates exactly the sequence of the user's interactions. For the construction of a test model, a modeling technique is used to express the sequence of events under test. An Event Sequence Graph (ESG) is a directed graph that can be used to model events and features of a software [31]. We use ESGs in our approach, which represent a linear test case demonstrating the sequence of GUI events (nodes) of the app connected by edges. For each event, a GUI element is selected, and some data are provided, such as the action type to be performed (for example, click or input text). This step is reproduced in two devices, *CRef1* and *CRef2*, and the final two ESG compatibles are generated (Figure 4).

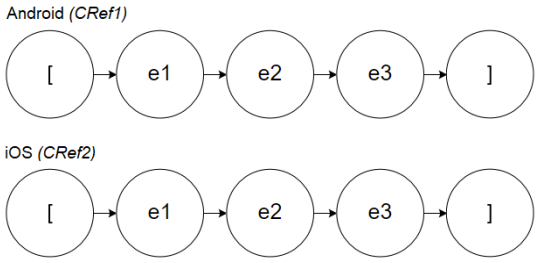

Figure 4. ESG mapping events under test.

In the process of elements selection, the GUI XML structure of the app is extracted. And based on this structure, each element of the GUI has identified its type (text box, button, anchor, etc.), as well as their key attributes with corresponding values are stored. Those data provide subsidies to build three XPath query expressions able to select the element. The first expression is cross-platform, that is, prepared to select a particular GUI element of the app independent of its platform. It is based on the key attributes of the elements and their respective values, discussed in the previous section. The second expression is specific to each platform and is built based on the type of the selected element and combined with the platform key attributes, thus resolving issues related to lack of compatibility among

versions of the same platform. Finally, the third expression is also specifics of the platform and is based on the elements absolute path within the XML structure. To illustrate, the example of Figure 2 is used again to show the query expressions to select the button "Search for a market":

Expression 1 - Cross-platform XPath

//*[@resource-id='search' or @name='Search For a Market']

Expression 2 - XPaths based on type of the element and its attributes keys-Android and iOS

//*/android.view.View[@resource-id='search' and @content-desc='Search For a Market']
//*/UIALink[@name='Search For a Market' and @label='Search For a Market']

Expression 3 - XPaths based in element absolute path of within the XML structure of the GUI-Android and iOS

hierarchy/android.widget.FrameLayout /android.widget.LinearLayout/android.widget.FrameLayout /android.webkit.WebView/android.webkit.WebView /android.view.View/android.view.View /android.view.View[2]/android.view.View /android.view.View/android.view.View[3]
AppiumAUT/UIAApplication/UIAWindow /UIAScrollView/UIAWebView/UIALink[2]

Expressions are used to select the elements during the automated tests. In failure of the first expression (cross-platform), the other expressions can be used to select element and do not interrupt the test execution. A tool was implemented based on this approach, and it assists in the construction of the ESG and the query expressions, generated automatically when the tester interacts with the app and indicates the GUI elements for testing. Details on the tool are available in the Section 4.

3.3. Single Test Engine

Step 3 is the formalization of a common mechanism for automated GUI testing in multiple settings, in this case, a common test script among different platforms. ESG is the basis for its construction, since each event contains data of a GUI element under test, as well as, its query expression to select it. It is prioritized the use of the first XPath query expression due to its suitability for cross-platform (line 14), and on its fail to try to select the element, the other specific expressions of the platform are executed (lines 15-19). A pseudocode of the test script proposed by the approach is presented in the Algorithm 1. The *ExecModel* method receives, as parameters, the events mapped in the ESG, and for each event it reposts on its respective XPath query expressions to the *ExecAction* method, in which locates an element in the XML structure of the GUI of the app using the expressions. In the end, the found element (by some of the expressions) is executed according to the action indicated by the tester (lines 20-26). The test script can be used in other devices from multiple settings *(Cn)*, ensuring the app test in a greater breadth of device. In this case, one option is to use the test script in clouds environments tests, in which offer a large number of devices.

Algorithm 1 Script test generation

```
1: procedure ExecModel(events[], deviceConfig)

2:     input events[] - AUT events mapping in ESG
3:     input deviceConfig - Data about device under test

4:     for each evt in events do
5:         ExecAction(evt.xPathSelectorsCrossPlatform,
       evt.xPathSelectorsAndroid, evt.xPathSelectorsiOS,
       deviceConfig.platform);
6:     end for
7: end procedure
8:
9: procedure ExecAction(xPathSelectorsCrossPlatform[],
   xPathSelectorsAndroid[], xPathSelectorsiOS[], platform)

10:    input xPathSelectorsCrossPlatform[] - Elements selectors
       Crossplatform
11:    input xPathSelectorsAndroid[] - Elements selectors for Android
12:    input xPathSelectorsiOS[] - Elements selectors for iOS
13:    input platform - Platform name under test

14:    xPathSelectors[] = xPathSelectorsCrossPlatform;

15:    if platform = "Android" then
16:        xPathSelectors[] += xPathSelectorsAndroid;
17:    else if platform = "iOS" then
18:        xPathSelectors[] += xPathSelectorsiOS;
19:    end if

20:    for each selector in xPathSelectors[] do
21:        e = FindElementByXPath(selector);
22:        if e != null then
23:            break;
24:        end if
25:    end for
26:    e.action();
27: end procedure
```

3.4. Comparison Model

Step 4 includes test script execution in multiple settings and comparison of its respective status to identify inconsistencies among the multiple settings. A hybrid cross-platform app can present inconsistency faults from different behaviors in each device configuration. The execution of events mapped in ESGs generates a new status containing verifiable results of inconsistencies. Algorithm 2 illustrates the process of comparison of these status. The algorithm gets the app under test (AUT) and the data on the multiple settings used in the test, two ESGs mapped to the app test and the criteria set for verification. The algorithm performs every event and extracts its status to verify inconsistencies (lines 9-24). At the end of executing such inconsistencies are returned by the algorithm.

We propose two criteria to find inconsistencies faults arising from cross-platform feature of app , equivalence and compatibility. The equivalence criteria are related to non-equivalent status among the runs, which is indicative of inconsistency fault. The compatibility criteria are related to

Algorithm 2 Comparison model

```
1: procedure COMPARE(AUT, Cn1, Cn2, Events[], Criterias)
2:     input AUT - App under Test
3:     input Cn1 - Model Config. Device 1
4:     input Cn2 - Model Config. Device 2
5:     input Events[] - AUT events mapping in ESG
6:     input Criterias - Verification criteria
7:     output Inconsistences[] - Inconsistencies found
8:     continue = true;
9:     for each (e_i in Events and continue) do
10:        exec(AUT, Cn1, e_i);
11:        sN1 = getState(AUT, Cn1, e_i);
12:        exec(AUT, Cn2, e_i);
13:        sN2 = getState(AUT, Cn2, e_i);
14:        for each c_i in Criterias do
15:            if !ApplyCriteria(c_i, sN1, sN2) then
16:                if c_i.equivalence then
17:                    Inconsistences[] = "Inconsistency fault", sN1, sN2;
18:                    continue = false;
19:                else if c_i.compatibility then
20:                    Inconsistences[] = "Incompatibility Indicator", sN1,
   sN2;
21:                end if
22:            end if
23:        end for
24:    end for
25:    return inconsistencies[];
26: end procedure
```

the status compatibility of the executions. Non-compliant status are indicative of difference and do not necessarily identify faults, but it needs to be inspected. In this case they do not compromise the continuity of the running of the verification process. The criteria set considered in this approach are listed below.

The equivalence criteria are:

- *Successful Run **(SR)**:* This criterion is based on the full execution and success of the event. The execution status are compared in order to check their equivalency.
- *Compare Attributes Values **(CAV)**:* This criterion is based on attribute mapping among the elements of the events (Table 1), in which the status are compared to check their equivalency.

The compatibility criteria are:

- *Text Similarity - Hybrid WebView Container **(TSHC)**:* From the node WebView, we extract all text contained in the XML structure of the GUI. The WebView contains HTML elements mapped to XML. The textual difference among the different devices can indicate some inconsistency;
- *Text Similarity Total **(TST)**:* Similar to the previous criterion, but it uses all text contained in any part of the XML structure;
- *Screenshot Similarity **(SS)**:* The capture of screenshot executions of events allows comparison with differencing algorithms, aiming at establishing the level of similarity among the GUIs. Differences in similarity can compromise the running of automated testing;
- *Screenshot OCR (Optical Character Recognition) Similarity **(SOCRS)**:* Similar to the previous criterion, but considers the text extracted by OCR of screenshot to establish levels of GUIs similarity;
- *Runtime Percentage Difference **(Runtime)**:* The difference in the proportional running time of the events in each device configuration can be an indicative of inconsistency, damaging the automated test.

4. Tool Implementation

Our approach has been implemented in a prototype tool to support software testing in hybrid mobile app that allows compatibility test script to run under different platforms.

Our tool was based on the Appium Framework [32], which is open source tool to automate test in native mobile apps, Web or hybrid. In addition, it is cross-platform and makes it possible to automate tests for iOS and Android platforms, using a Selenium WebDriver API[2]. Appium provides support for our tool connect itself to two devices, an Android and other iOS, and also accesses to app GUI elements. The events indicated by the tester are recorded and the GUI XML is extracted and treated, as our approach of making the test script, using the appropriate XPath. The tool generate a test project for Microsoft Visual Studio encoded in C# with support for Unit Testing Framework [33]. Test project contains the test script appropriate to test the app in multiple settings. One last feature of the tool is to indicate the tester if an element has the features allowing its selection on multiple platforms. The test script is made to test the same app running under Android and iOS platforms and in its different versions. Figure 5 shows a screenshot of tool and offers the functionality to check the GUI elements, definition of its actions (click or text input) for later comparison, extracting of the status and test script generation. However, the prototype tool currently does not support complex user interactions, such as multi-touch and gestures (e.g., pinch). Listing 1 shows a code snippet generated by the tool and made compatible to test the same app under execution in multiple settings. First XPath query expression is prioritized due to your ability for select elements of GUI in multiple settings. The other expressions are used in an attempt to select the element if the first expression to fail. If the element is found, then the action indicated by the testator is performed.

To build the approach and the Appium-based tool, we hope to build a test infrastructure that works in environments to test apps in multiple settings such as Amazon Device Farm [34], Bitbar [35] and TestObject [36]. These services offer a large number of real devices that can be connected and used in testing of cross-platform mobile apps.

5. Preliminary Evaluation

To evaluate the proposed approach we conducted an empirical evaluation, which addresses the following research questions (RQs) from a software testing perspective:

2. http://www.seleniumhq.org/projects/webdriver/

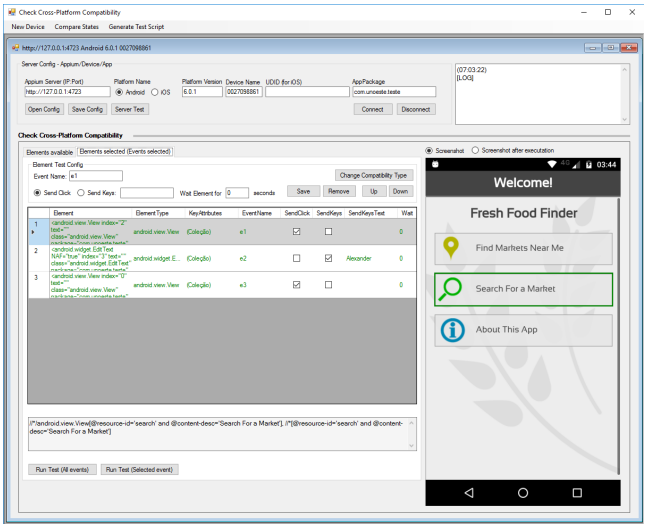

Figure 5. Screen tool for test element/events selection.

RQ1. Among the three query expressions defined in the approach, which is the success level in the selection of GUI elements during the test script execution?

RQ2. What is the accuracy regarding the execution of the test script in apps under test in multiple settings?

5.1. Experimental Objects and Procedure

To answer our RQs we proceeded as follows. We have selected a set of hybrid apps, two industrial apps and three samples apps to run on Android and iOS platforms. The industrial apps projects containing the necessary assets to build on Android and iOS were provided by IT companies. The sample apps projects were obtained from books about developing hybrid apps and repositories on GitHub. Table 3 lists the apps included in our evaluation and some characteristics of their respective projects. Features as the type of app, number of HTML and CSS files, Javascript number of lines (LOC), Javascript framework and cross-platform development framework are presented. The projects of these apps were recompiled for Android and iOS platforms, the result was the binary for installation. The native development environment for both platforms were set up, once the build occurs by the owner of the platform environment.

Table 3. APPS UNDER TEST

App	Type	HTML Files	CSS Files	JS LOC	JS Framework	Cross-Platform Framework
Fresh Food Finder	Industrial	13	8	13.824	jQuery and Mustache	Cordova
Order App	Open Source	3	4	71.565	AngularJS	Cordova
MemesPlay	Industrial	1	4	5.484	ReactJS	Cordova
Agenda	Open Source	1	6	1.038	Material Design Lite	Cordova
TodoList	Open Source	1	1	9.304	AngularJS	Cordova

The apps have been installed on four real devices, both Android and iOS, described in Table 4. Each app had its elements under test mapped with the use of the tool as approach discussed earlier.

Listing 1. Test script generated by the tool.

```
1  e1SendClick_Test(); //e1
2
3  public static void e1SendClick_Test()
4  {
5    string[] xPathSelectorsCrossPlatform = new string[]
         {"//*[@resource-id='search' or @name='Search
       For a Market']"};
6
7    string[] xPathSelectorsAndroid = new string[] {"
       //*/android.view.View[@resource-id='search' and
        @content-desc='Search For a Market']",
8    "//*[@resource-id='search' and @content-desc='
       Search For a Market']",    "hierarchy/android.
       widget.FrameLayout/android.widget.LinearLayout/
       android.widget.FrameLayout/android.webkit.
       WebView/android.webkit.WebView/android.view.
       View/android.view.View/android.view.View[2]/
       android.view.View/android.view.View/android.
       view.View[3]"};
9
10   string[] xPathSelectorsiOS = new string[] {"//*/
       UIALink[@name='Search For a Market' and @label
       ='Search For a Market']",
11   "//*[@name='Search For a Market' and @label='Search
        For a Market']", "AppiumAUT/UIAApplication/
       UIAWindow/UIAScrollView/UIAWebView/UIALink[2]"
       };
12
13   IWebElement e = FindElementByXPath(
         xPathSelectorsCrossPlatform,
         xPathSelectorsAndroid, xPathSelectorsiOS);
14
15   e.Click();
16 }
17
18 public IWebElement FindElementByXPath(string[]
       xPathSelectorsCrossPlatform, string[]
       xPathSelectorsAndroid, string[] xPathSelectorsiOS
       )
19 {
20   IWebElement e = null;
21
22   List<string> selectors = new List<string>();
23   selectors.AddRange(xPathSelectorsCrossPlatform);
24
25   if (_platform == "Android")
26     selectors.AddRange(xPathSelectorsAndroid);
27   else if (_platform == "iOS")
28     selectors.AddRange(xPathSelectorsiOS);
29
30   foreach(string selector in selectors)
31   {
32     try
33     {
34       e = Driver.FindElementByXPath(selector);
35     }
36     catch{}
37
38     if (e != null)
39       break;
40   }
41
42   return e;
43 }
```

Table 4. DEVICES EVALUATED

Device	OS	Screen (inch)	Processor	RAM
Motorola G4	Android 6.0.1	5.5	Octa Core 1,4 GHz	2 GB
Motorola G1	Android 5.1	5	Quad Core 1.2GHz	1 GB
iPhone 4	iOS 7.1.2	3.5	Duo Core 1 GHz	512 MB
iPad 2	iOS 9.3	9,7	Duo Core 1 GHz	512 MB

Our tool connects to Appium which, in turn, connects to mobile devices offering means for executing the test script and extracting data (for later comparison). An Appium specific version (according to the mobile OS) has been installed and configured in two test servers for network access to local computers. In detail, the tools used in the

experiment were:

- XCode 7.3;
- Android SDK r24.4.1;
- VisualStudio 2015;
- Appium 1.4.3 (Windows OS) e 1.5 (MAC OS);
- Selenium WebDriver for C# 2.53.0;
- Appium WebDriver 1.5.0.1;
- Windows 10 OS;
- MAC OS X Sierra.

5.2. Preliminary Results and Findings

This section presents and discusses the preliminary results collected by running the method described in the previous section, in order to answer the research questions in this present study. Five apps were submitted to GUI automated testing in multiple configurations by using the single test script generated by our tool. Table 5 presents details taken from the test procedure, such as the amount of test cases by app, the number of events per test case and the test script LOC.

Table 5. APPS EVALUATION DETAILS

App	Test Case	Events Under Test	Test Script LOC
Fresh Food Finder	1	3	157
Order App	1	5	170
MemesPlay	1	3	152
Agenda	1	7	208
TodoList	1	8	229

During script execution was verified which query expression returned successfully the selected element to the test. The first expression is based on key attributes of the elements and is prepared to select a particular GUI element independent of its platform. While the other expressions are based on XPath specific selectors of the platform under test. The evaluation involved four devices listed in Table 4. Table 6 presents results collected when evaluating the three types of XPath expressions and the test script accuracy.

In general, the first XPath expression succeeded to 82.65% in its ability to select GUI elements in apps under test running in multiple device configurations. If the first expression to fail, other expressions (platform specific) were performed, and the amount totaling 7.58% success together in the elements selection. The failure of all expressions (ESF) summarizes 9.76%. Figure 6 summarizes the percentages obtained and discussed above. Such analyses answer QP1.

The following remarks respond to QP2. The test script run under the multiple settings was successfully completed at 90.24% of executions (in overall). The comparative between same platform devices and different versions (for example, Android 5.1 vs Android 6.0.1 and iOS 7.1 vs iOS 9.3), the evaluation showed that the test script was run

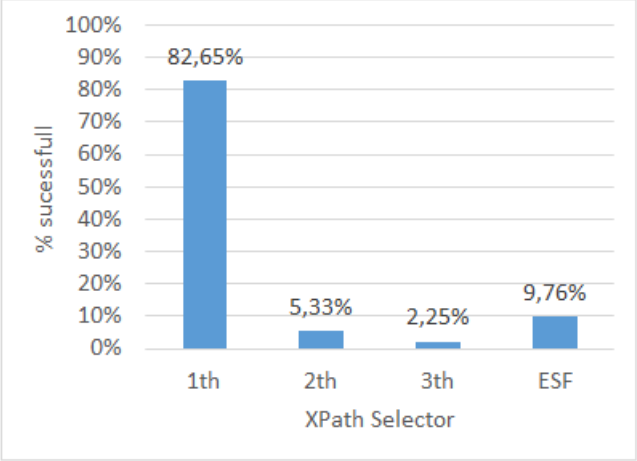

Figure 6. XPath selector evaluation.

successfully in 97.17% for the Android platform and 83.33% for the iOS platform. However, the remaining percentage (9.76% overall) demonstrates that the script did not complete the execution due to failures in the selection of elements. In these cases, none of the query expressions was able to select one or more elements. The failures are detailed as follows:

- For the Agenda app, an element related to the main menu was positioned negatively, that is, outside the GUI viewing area. Such positioning prevented the element selection in the two versions of Android devices, which did not occur in the iOS versions. This analysis suggests a problem in the Appium test engine of the Android platform.
- For the Order App running on iOS 7.1, the XML structure only contained data about then container elements, making the selection of any other internal element impossible. Such behavior did not occur in iOS 9.3. Other apps under test in iOS 7.1 did not present such problem.
- For the MemesPlay app running on the two versions of iOS, a text box element was selected, but the Appium failed in an attempt to simulate the text input. On the Android platform, the text input has occurred correctly.

During the evaluation, we identified that the use of the key attributes in GUI elements in development phase, ensures a higher level of success in the selection of these elements during the tests in multiple settings.

5.3. Threats to Validity

This section introduces, briefly, possible threats to validity. The first point to be considered is the researchers set up and execute all the experiment; some bias might be. Initially, we have a small number of mobile devices and apps used in the evaluation, undermining the validity of the obtained results. To mitigate this threat we selected industrial and open source apps. One last threat is related

Table 6. XPATH SELECTOR EVALUATION

Devices	Apps																								
	Fresh Food Finder					Pedido App					MemesPlay					Agenda					TodoList				
	EUT	**1th**	**2th**	**3th**	**ESF**	**EUT**	**1th**	**2th**	**3th**	**ESF**	**EUT**	**1th**	**2th**	**3th**	**ESF**	**EUT**	**1th**	**2th**	**3th**	**ESF**	**EUT**	**1th**	**2th**	**3th**	**ESF**
Android 5.1	3	3				5	4	1			3	3				7	6			1	8	7		1	
Android 6.0.1		3					4	1				3					6			1		7		1	
iOS 7.1		2	1							5		2			1		7					8			
iOS 9.3		2	1				4		1			2			1		7					8			
EUT: Elements Under Test, 1th: First XPath (crossplatform), 2th: Second XPath, 3th: Third XPath, ESF: Element Selection Fail																									

to the proposed criteria to identify inconsistencies and compatibility of equivalence between the captured status after running of the events. These criteria were not evaluated. As regards reproduction of the experiment, the tool and the objects of the experiment are publicly available in a website [37].

6. Related Work

In this section, we describe the most relevant research from area. Fazzini et al. [38] have implemented a technique that consists of three main phases: (i) recording the user interaction with the app with the goal of testing its functionality. It is offered a convenient interface to define assertion-based oracles; (ii) test case generation phase based recorded interactions; (iii) The technique executes the generated test cases on multiple devices and summarizes the test results is a report. In details, the test case generation phase produces as output a test case that faithfully reproduces the actions performed by the user during the test case recording phase. The generated test case is an Android UI test case based on the Espresso framework [39]. At the end of the execution, the technique generates a test case execution report that contains the outcome of the test case on each device, the execution time, and debug information if an error or failure occurred during execution. The oracles are generated based on the properties of the elements. The technique was implemented in a framework called Barista, which was evaluated a in empirical study involving 15 human subjects and 15 real-world Android apps. Those 206 test cases recorded were executed on seven (real) devices. Overall, the average compatibility rate across all apps and devices was 99.2%. Two test cases did not work because some device adds additional space at the bottom element of a TableLayout element. In comparing cross-device compatibility of test cases generated using Barista against another related tool (TestTroid Record [40]), Barista was superior. Barista does not currently offer support events for WebView elements. Our work is different to build software testing unified mechanisms for hybrid apps running on Android and iOS platforms.

Lee et al. [41] developed a framework for static code analysis of hybrid apps specific for the Android platform. The hybrid apps development frameworks use mapping mechanisms or bridge to perform from Javascript calls to native code. Baptized HybriDroid, the framework aims to analyze the intercommunication between Java and Javascript (native and Web environment) used in hybrid apps. The analysis was conducted exploring Java classes extracted from apps. For validation, 88 hybrid real-world apps downloaded directly from the Google store were collected. Overall, 14 apps were reported with 31 faults identified by HybriDroid, 24 were observed as true and 7 false positives. Most of the faults were classified as *MethodNotFound*, that is, when an invocation to a Java method from Javascript is not found. Other faults found in lower incidence were: *MethodNotExecuted* - when the return of a Java method is not compatible with the Javascript data type, *TypeOverloadedBridgeMethod* - when the mapping mechanism does not know what a Java method overload should be used and *IncompatibleTypeConversion* - when a Java method has arguments types not supported by Javascript. Different from our approach, this work does not cover the apps execution and automated testing. Moreover, only apps for the Android platform were analyzed.

Wei et. al. [42] claim that Android ecosystem is heavily fragmented. The numerous combinations of different device models and OS versions make it impossible for Android app developers to exhaustively test their apps. As a result, various compatibility issues arise, causing poor user experience. The researchers conducted an empirical study to understand and characterize fragmentation issues in Android apps at the source code level (static analysis). The study led to a technique called FicFinder to automatically detect compatibility issues. To validate FicFinder, the authors investigated 191 real compatibility issues collected from five popular open-source Android apps to understand their root causes, symptoms, and fixing strategies. The five major root causes of compatibility issues in Android apps, of which the platform API evolution and problematic hardware driver implementation are most prominent. Compatibility issues can cause both functional and non-functional consequences such as app not functioning, performance and user experience degradation. Issue fixes are usually simple and demonstrate common patterns: checking device information and availability of software/hardware components before invoking issue-inducing APIs/methods. Our approach goes beyond, due to executation of the app code and define common mechanisms of compatibility between different ecosystems. We aim to identify faults from a test executation perspective.

Joorabchi et. al. [23] claim that, ideally, a given cross-platform app must provide the same functionality and behaviour in the various platforms. In this scenario, the researchers proposed a tool called Checking Compatibility Across Mobile Platforms (CHECKCAMP), able to detect and show inconsistencies between versions of the same native app encoded for different platforms, namely, Android and iOS. During the execution of the app by the researchers, a code parser dynamically intercepts calls to methods imple-

mented, and captures data from the GUI after the return of these methods (state). Based on the data, the tool generates a template for both platforms and maps their nodes. In the end, these models are compared looking for inconsistencies. In its evaluation, 14 pairs of apps (apps contained in the two platforms) showed that the approach based in the GUI model provides an effective solution. The CHECKCAMP inferred correctly the models with a high accuracy rate. Besides, the tool was able to detect 32 valid inconsistencies during the comparison of the models. The inconsistencies were classified into functional and data. Functional inconsistencies are related to differences among the models for each platform, while data inconsistencies are related to differences in the values stored in the GUI components such as buttons and labels. Our work presents a similar approach, but without the need for code instrumentation (black-box approach) and with a focus on hybrid apps. We assume that the tester will track and write the events, creating your own tests.

Boushehrinejadmoradi et al. [1] performed tests on a framework of cross-platform mobile app development. The paper suggests that there are two classifications for cross-platform app development frameworks: Web-based framework and Native framework. This last classification uses a home/main platform and one or more target plataforms in the construction of the app, for example, iOS, Android or Windows Phone - using native resources, and subsequently the home is translated and compiled to another platform (target platform). Examples of such framewoks are Xamarin [43] and Apportable [44]. Focusing on Xamarin, they developed a tool called X-Checker that uses the methods of sequencing the app classes in the definition of test cases. The X-Checker goal is to discover framework inconsistencies by comparing the performance of different builds (one for Android and the other iOS) through differential testing. For tool validation, it generated 22,645 test cases, which invoke 4,758 methods implemented in 354 classes across 24 Xamarin DLL. Overall, they found 47 inconsistencies in Xamarin code. Our work differs from that to focus on software testing in hybrid apps based on Web technologies, not in the app development framework.

Mesbah and Prasad [45] defined the cross-browser compatibility problem and proposed a systematic, fully-automated approach for cross-browser compatibility testing that can expose a substantial fraction of the cross-browser issues in modern dynamic Web apps. A navigation model is used to compare page to page in two Web apps based on trace equivalence and screen equivalence. Their testing approach was implemented in a tool called CrossT, that can to provide a very potent cross-browser compatibility tester for end-user. As Web apps, GUI hybrid apps are built with HTML, CSS and Javascript code, sharing issues related to software testing and cross-browser (mobile platform) compatibility. In addition, the hybrid apps structure has increased complexity due to access to native resources of the mobile OS.

Appium Studio [46] is an Integrated Development Environment (IDE) designed for mobile test automation development and execution using on Appium framework [32] and Selenium WebDriver API. It supports to write, record and execute tests for apps in Android and iOS devices. The tool eliminates a large majority of Appium rigid dependencies model and prerequisites such as the requirement to develop iOS tests on MAC OSX machines or the inability to run tests in parallel on real/simulated iOS Devices. Our approach complements Appium Studio to build test scripts adapted for running on multiple devices of different configurations. A plugin can be developed as future work.

7. Conclusion and Future Work

This study presented and evaluated an automated testing approach to hybrid mobile apps. Our approach consists in the formalization of a common mechanism for automated GUI testing on multiple devices settings. The aim is the construction of a common test script able to test the same app across different platforms. For this purpose, three query expressions of GUI elements are defined by approach. The first expression is based on attributes of the elements and is prepared to select a particular GUI element independent of its execution platform. While the other expressions are based on specific selectors of the mobile platform under test. A tool was implemented based on that approach, which provides a test project configured with the script to run in multiple settings. The results suggested that the executation of the common test script was completed successfully in 90.24% of the times, and that the query expression cross-platorm also had a success of 82.65% when selecting GUI elements among the multiple settings.

In future works we intend to replicate the evaluation experiment in more hybrid apps. Moreover, increasing the breadth of devices when using the test script in test environments in clouds. In a second future work, we plan to address the equivalence and compatibility criteria to identify inconsistencies faults arising from the characteristic of cross-platform of the hybrid apps. A third intended work consists of the using of this approach in native apps built by development frameworks of cross-platforms apps, such as Xamarim [43] and React Native [47].

References

[1] Nader Boushehrinejadmoradi, Vinod Ganapathy, Santosh Nagarakatte, and Liviu Iftode. Testing Cross-Platform Mobile App Development Frameworks. In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference*, pages 441–451, 2015.

[2] IDC. Smartphone OS Market Share, 2015 Q2. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2015. Accessed on 2017-03-01.

[3] Android. Android. https://www.android.com, 2017. Accessed on 2017-03-01.

[4] Apple Inc. Apple. https://www.apple.com, 2017. Accessed on 2017-03-01.

[5] Gartner. Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016.

http://www.gartner.com/newsroom/id/3323017, 2016. Accessed on 2017-02-26.

[6] Statista. Number of apps available in leading app stores as of July 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores, 2015. Accessed on 2017-02-25.

[7] Statista. Number of apps available in leading app stores as of June 2016. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2016. Accessed on 2017-02-25.

[8] Mounaim Latif, Younes Lakhrissi, El Habib Nfaoui, and Najia Es-Sbai. Cross platform approach for mobile application development: A survey. *2016 International Conference on Information Technology for Organizations Development (IT4OD)*, pages 1–5, 2016.

[9] IBM. Native, web or hybrid mobile-app development. ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf. Accessed on 2017-03-09.

[10] W3C. W3C. https://www.w3.org/. Accessed on 2017-03-17.

[11] Spyros Xanthopoulos and Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA, 2013. ACM.

[12] Cordova. Cordova. https://cordova.apache.org/, 2017.

[13] PhoneGap. PhoneGap. http://phonegap.com/, 2017.

[14] Sencha. Sencha. https://www.sencha.com/, 2017.

[15] IONIC. IONIC. http://ionicframework.com/, 2017.

[16] Intel XDK. Intel XDK. https://software.intel.com, 2017.

[17] Telerik. AppBuilder. http://www.telerik.com/platform/appbuilder, 2017.

[18] Anonymous Authors. Omitted per double blind review. Oct 2016.

[19] Tor-Morten Gronli and Gheorghita Ghinea. Meeting Quality Standards for Mobile Application Development in Businesses: A Framework for Cross-Platform Testing. In *49th Hawaii International Conference on System Sciences (HICSS 2016)*, pages 5711–5720. IEEE, jan 2016.

[20] M E Joorabchi, A Mesbah, and P Kruchten. Real Challenges in Mobile App Development. *IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013)*, pages 15–24, 2013.

[21] Meiyappan Nagappan and Emad Shihab. Future Trends in Software Engineering Research for Mobile Apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 21–32. IEEE, mar 2016.

[22] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. End Users' Perception of Hybrid Mobile Apps in the Google Play Store. *Proceedings - 2015 IEEE 3rd International Conference on Mobile Services, MS 2015*, (iii):25–32, 2015.

[23] Mona Erfani Joorabchi, Mohamed Ali, and Ali Mesbah. Detecting inconsistencies in multi-platform mobile apps. In *IEEE 26th International Symposium on Software Reliability Engineering (ISSRE 2015)*, pages 450–460. IEEE, nov 2015.

[24] Daniel Cortez. Mobile App Development: Native vs. Hybrid vs. Mobile Websites. http://www.uptopcorp.com/post/mobile-app-development-native-vs-hybrid-vs-mobile-websites. Accessed on 2017-01-22.

[25] W3C. Selectors Level 3. https://www.w3.org/TR/2011/REC-css3-selectors-20110929/. Accessed on 2017-04-05.

[26] W3C. XML Path Language (XPath) Version 1.0. https://www.w3.org/TR/xpath/. Accessed on 2017-03-13.

[27] Moodle. Moodle Tracker. https://tracker.moodle.org/projects/mobile, 2017. Accessed on 2017-04-07.

[28] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*, volume 121990. IEEE, 1990.

[29] S Vilkomir, K Marszalkowski, C Perry, and S Mahendrakar. Effectiveness of Multi-device Testing Mobile Applications. In *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on*, pages 44–47, 2015.

[30] Sergiy Vilkomir and Brandi Amstutz. Using Combinatorial Approaches for Testing Mobile Applications. *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pages 78–83, 2014.

[31] Fevzi Belli, Christof J. Budnik, and Lee White. Event-based modelling, analysis and testing of user interactions: Approach and case study. *Softw. Test. Verif. Reliab.*, 16(1):3–32, March 2006.

[32] Appium. APPIUM. http://appium.io/, 2017.

[33] Microsoft. Unit Testing Framework. https://msdn.microsoft.com/en-us/library/ms243147(vs.80).aspx, 2017. Accessed on 2017-03-15.

[34] Amazon. AWS Device Farm - Amazon Web Services. https://aws.amazon.com/en/device-farm, 2017. Accessed on 2017-04-09.

[35] BitBar. Mobile App Testing - Testdroid Technology by Bitbar. http://www.bitbar.com/testing, 2017. Accessed on 2017-04-09.

[36] TestObject. TestObject. https://testobject.com, 2017. Accessed on 2017-04-09.

[37] Anonymous Authors. (our tool) omitted per double blind review. http://ase2017tool.gear.host, 2017. Accessed on 2017-04-28.

[38] Mattia Fazzini, Eduardo Noronha de A. Freitas, Shauvik Roy Choudhary, and Alessandro Orso. From Manual Android Tests to Automated and Platform Independent Test Scripts. *Computing Research Repository (CoRR)*, abs/1608.0, 2016.

[39] Google. Espresso. https://google.github.io/android-testing-support-library, 2017. Accessed on 2017-04-

14.

[40] Testdroid. Testdroid Recorded. http://www.testdroid.com, 2017. Accessed on 2017-04-14.

[41] Sungho Lee, Julian Dolby, and Sukyoung Ryu. HybriDroid : Analysis Framework for Android Hybrid Applications. *31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*, pages 250–261, 2016.

[42] Lili Wei, Yepang Liu, and Shing-chi Cheung. Taming Android Fragmentation : Characterizing and Detecting Compatibility Issues for Android Apps. *ASE '16 (31st IEEE/ACM International Conference on Automated Software Engineering)*, pages 226–237, 2016.

[43] Microsoft. Xamarin. https://www.xamarin.com/, 2017. Accessed on 2017-04-05.

[44] Apportable. Apportable. http://www.apportable.com/, 2016. Accessed on 2017-03-09.

[45] Ali Mesbah and Mukul R. Prasad. Automated cross-browser compatibility testing. *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 561, 2011.

[46] Experitest. Appium Studio. https://experitest.com/appium-studio/, 2017. Accessed on 2017-04-03.

[47] Facebook. React Native. https://facebook.github.io/react-native/, 2017. Accessed on 2017-04-05.