Automated Testing of Cross-Platform Mobile Apps in Multiple Configurations

***Abstract*—*Background:* ....**

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 a survey from International Data Corporation (IDC) [2] about the Market Share 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: Android was the market leader, followed by Apple's iOS. Those modern OSs serve as platform for execution of a wide variety of software applications called mobile apps. The Statista site offers Statistics of March 2017 [6] on the number of apps available for download in the main distribution stores. Android has the largest number of apps available to its users (arround 2.8 million apps), also followed by iOS (arround 2.2 million apps).

The development of mobile apps can be classified in three groups: native apps, browser-based Web apps and hybrid apps [7, 8]. 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, CSS3 and Javascript [9]. They are stored on a Web server, run under a client browser and they do not have access to advanced features of the mobile OS. Finally, hybrid apps combine common resources of Web apps like HTML5, CSS3 and Javascript with direct support to OS native Application Programming Interfaces (APIs), similarity to native apps. Such kind of App is divided into two parts, a native one known as WebView, responsible for executing Web resources contained in the second part [10]. Some commercial and open source frameworks provide support to the development of hybrid apps, such as Cordova [11], Phonegap [12], Sencha Touch [13], IONIC [14], Intel XDK [15] and AppBuilder [16].

Unlike native apps, hybrid apps have the advantage to execute across multiple platforms. Cross-platform apps stand out by their run in various OSs and eliminating the need to rewrite the app to meet a set of different devices. In the context of native apps, some frameworks, such as[17], ReactNative [18], ReactXP [19], NativeScript [20] and Weex [21], provide development mobile native cross-platform apps using languages such as C# and Javascript, and native elements of Graphical User Interface (GUI). These frameworks are able to translate source code of the mobile platform's specific language or use a non-native language. The final product is a native and cross-platform app [7, 10, 22, 23]. In this work was adopted the term native cross-platform for this type of apps.

Cross-plaform feature introduces a challenge in testing due to variability of multiple devices configurations on the market [1, 24, 25]. To illustrate the variability in the context of mobile devices, Figure 1 uses a Feature Model (FM) to represent characteristics of a fictional set of mobile devices. The model is a hierarchical representation of the characteristics and relations among them.

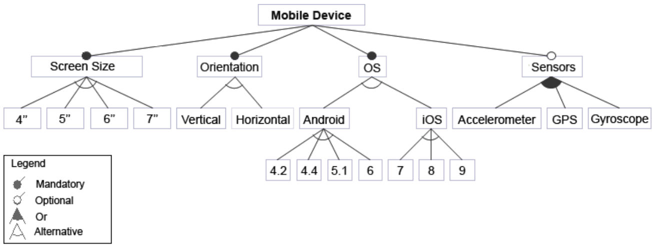


Figure 1. Characteristics of a set of mobile devices.

Each product represents a configuration that needs to be tested. This paper defines multiple settings as variabilities of mobile devices with different settings, for example, OS, screen size, several types of sensors and hardware. In such diversity, testing apps 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 [25]. Even within the same platform there is no homogeneity [26]. In addition, the test in a single device does not guarantee correct operation in other [27].

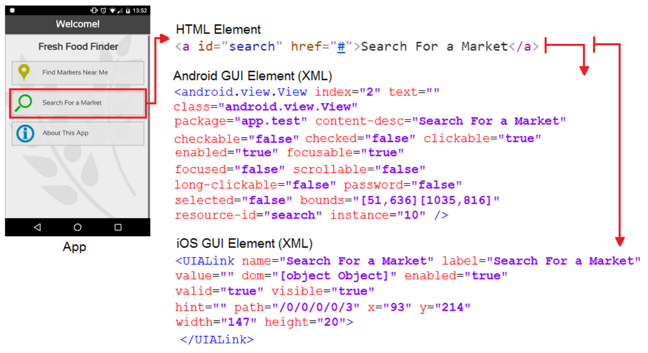
In the scenario of cross-platform apps, the use of hybrid apps is considered a visible solution to avoid the fragmentation in various OSs [28], since their main feature is the ability of running on multiple platforms. Another strategy to avoid such a problem is the use of frameworks for development native cross-platforms apps. However, cross-plaform apps testing can be more challenging, since, in addition to the execution of the tests to identify faults inserted during the development phase, tests required to execute in several platforms or configurations. Faults related to compatibility of the execution of the same app on different platforms can arise [29]. The app can work correctly on platform X and fails on platform Y, even in different versions of the same platform.

The app is developed with features that enable its cross-platform execution, but the mechanisms to test it are not, therefore different test scripts are required. There are tools for automating test on specific platforms, however, there is no effort to generate an automated test for cross-platform. In this context, this paper aims to, automates the testing of cross-platform mobile apps, supporting test script generation able to test the same app running in multiple configuration. In evaluated, nine cross-platform mobile apps were subjected to testing on four real devices.

This paper is structured as follows: Section 2 motivates the research problem. Section 3 introduces an approach to test cross-platform mobile apps. Section 4 describes our prototype tool and the experimental evaluation is shown in Section 5. Section 6 brings some related work. Finally, Section 7 concludes and sketches future work.

2. Motivating Problems

The GUI of a 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 [30], 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 native GUI elements generated by the Android and iOS platforms is presented in Table 1.



HTML Element

```
<a id="search" href="#">Search For a Market</a>
```

Android GUI Element (XML)

```
<android.view.View index="2" text=""
class="android.view.View"
package="app.test" content-desc="Search For a Market"
checkable="false" checked="false" clickable="true"
enabled="true" focusable="true"
focused="false" scrollable="false"
long-clickable="false" password="false"
selected="false" bounds="[51,636][1035,816]"
resource-id="search" instance="10" />
```

iOS GUI Element (XML)

```
<UIALink name="Search For a Market" label="Search For a Market"
value="" dom="[object Object]" enabled="true"
valid="true" visible="true"
hint="" path="/0/0/0/0/3" x="93" y="214"
width="147" height="20">
 </UIALink>
```

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

Table 1. MAPPING HTML ELEMENTS TO XML NATIVE 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

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.

Table 2. MAPPING HTML ATTRIBUTES TO XML NATIVE 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

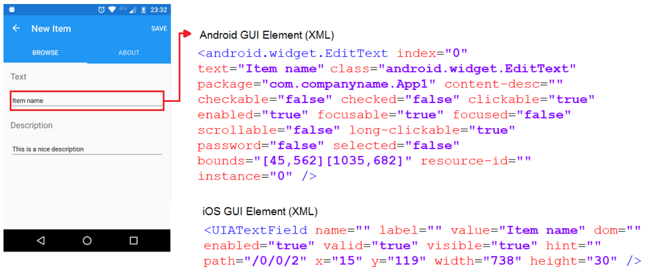
No caso dos frameworks de desenvolvimento de apps native cross-platform, o mapeamento ocorre, mas o HTML não é a origem, uma vez que a GUI é construída com os elementos nativos e originais da plataforma móvel. Para cada plataforma uma versão da aplicação é gerada. A Figura 3 demonstra a estrutura XML de um elemento de GUI em uma aplicação desse tipo construída com Xamarin. É possível perceber a diferença dessa estrutura entre as plataformas Android e iOS.

For apps developed with native cross-platform frameworks, the mapping occurs, but HTML is not the source, because the GUI is constructed with mobile platform native elements. For each platform a app version is generated. Figure 3 shows an XML structure of a GUI element in an app of this type made with Xamarin. It is possible to perceive the difference of this structure between Android and IOS platforms.

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 [31]. A computational mechanism for selecting XML elements are query expressions in e.g. XPath[1], enabling automated GUI testing.

It is possible to observe when reviewing Figure 2 and Figure 3 that the platform manufacturers do not allow an

[1] XPath is a query language for selecting elements (nodes) in computational structures named by tree which represent XML documents [32].



Android GUI Element (XML)

```
<android.widget.EditText index="0"
text="Item name" class="android.widget.EditText"
package="com.companyname.App1" content-desc=""
checkable="false" checked="false" clickable="true"
enabled="true" focusable="true" focused="false"
scrollable="false" long-clickable="true"
password="false" selected="false"
bounds="[45,562][1035,682]" resource-id=""
instance="0" />
```

iOS GUI Element (XML)

```
<UIATextField name="" label="" value="Item name" dom=""
enabled="true" valid="true" visible="true" hint=""
path="/0/0/2" x="15" y="119" width="738" height="30" />
```

Figure 3. XML representing the GUI elements of native cross-platform apps.

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]"***.

Assumindo que uma aplicação cross-platform é a mesma para diferentes plataformas e o teste não é, faz-se necessário a definição de um abordagem de testes automatizados e reaproveitá-los em um conjunto de dispositivos de múltiplas configurações, assim evitando a rescrita do script de teste. A ideia é manter um único script de teste para a aplicação, independe de seu ambiente de execução.

3. Approach Overview

Our approach defines a mechanism to automate the testing of cross-platform apps and the construction of a test script able to test the same app running in multiple settings. The approach consists of three steps. The first step (Figure 4-a, Section 3.1) is based on the selection of two mobile devices of reference, one running the Android OS, called C_{Ref1} and other running iOS, called C_{Ref2}. The second step (Figure 4-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, running conditions 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 [33]. The third step (Figure 4-c, Section 3.3) proposes a single test script generation adequate for mobiles apps testing in multiples devices. An overview of our approach is illustrated in Figure 4 and in the sections that follow its steps are detailed.

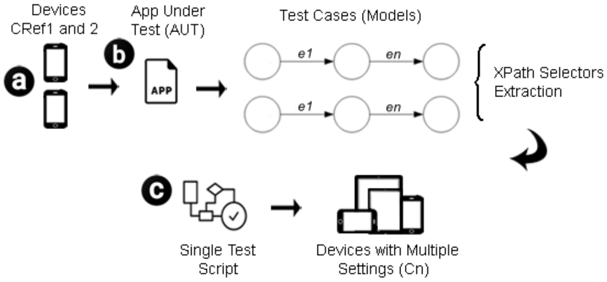


Figure 4. Approach Overview.

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 (e. g. [34, 35]) propose the choice of mobile devices for testing of apps based on their popularity of use. Such devices are named as reference configuration (C_{Ref1} andC_{Ref2}). In addition, a cross-platform mobile *app under test (AUT)* should be installed on those devices.

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 simulates the sequence of events 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. We employed ESGs in our approach to represent a linear test case demonstrating the sequence of GUI events (nodes) of the app connected by edges. An Event Sequence Graph (ESG) is a directed graph that can be used to model events and features of a software [36]. For each event, a GUI element is selected, and some action is executed (for example, click or input text). This step is reproduced in two devices, C_{Ref1} and C_{Ref2}, and the final two compatibles ESGs are generated (Figure 5). The ESG compatibility is related to equality of events number such as also in the equality of data provided for each elements.

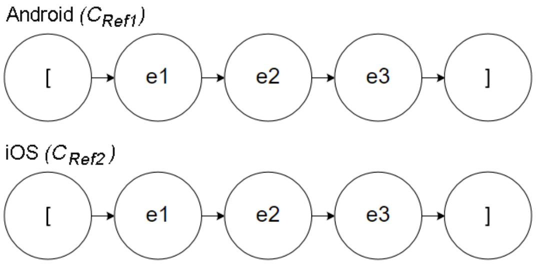


Figure 5. ESG mapping events under test.

No processo de seleção dos elementos para o caso de teste e definição do modelo de teste, a estrutura XML da GUI da aplicação é extraída. Com base nessa estrutura, cada elemento da GUI tem seu tipo identificado (caixa texto, botão, âncora, etc), como também seus atributos chaves com respectivos valores são armazenados. Esses dados fornecem subsídios para a construção de dois tipos diferentes de expressões de consulta XPath capazes de localizar o elemento para o teste automatizado. O primeiro tipo consiste em uma expressão XPath absoluta, baseada no caminho absoluto do elemento a partir do nó raiz da estrutura XML. Os demais tipos consistem em expressões relativas, baseadas em caminhos alternativos gerados a partir de combinações de dados do elemento e/ou de elementos ancestrais. Este artigo propõe o uso de seis expressões. A seguir as expressões e alguns exemplos extraídos da GUI da Figure 2 são apresentados.

Caminho absoluto (AbsPath): É uma expressão específica da plataforma, e baseia-se no caminho absoluto do elemento dentro da estrutura XML a partir do nó raiz. Em alguns casos, índices são usados para identificar a posição do elemento dentro da estrutura XML. Essa abordagem tem sido empregada em aplicações Web para localizar elementos na estrutura DOM nesse tipo aplicação [37].

Android: 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]

iOS: AppiumAUT/UIAApplication/UIAWindow/UIAScrollView/UIAWebView/UIALink[2]

Atributos identificadores (IdAttr): É uma expressão baseada nos valores de atributos que identificam o elemento, tais como, *resource-id* para plataforma Android e *name* para plataforma iOS. Abordagem igual é utilizada em aplicações Web [37, 38], nesse caso os elementos HTML são identificados pelos atributos Id e/ou Name.

Android: //*[@resource-id='search']

iOS: //*[@name='Search For a Market']

Multiplataforma (CrossPlat): É uma expressão proposta neste estudo, na qual objetiva a definição de uma expressão única entre plataformas diferentes. Tal expressão é preparada para selecionar um determinado elemento da GUI da aplicação independente da sua plataforma de execução. Ela é baseada na combinação dos atributos chaves (Android: *content-desc* ou *text* e iOS: *label* ou *value*) dos elementos e em seus respectivos valores em ambas plataformas, discutidos na Seção 2.

Android e iOS: //*[@content-desc='Search For a Market' or @label='Search For a Market']

Tipo do elemento (ElemType): É uma expressão preparada para localizar um elemento baseado na combinação de seu tipo e em seus respectivos atributos chaves e identificadores da plataforma específica.

Android: //*/android.view.View[@content-desc= 'Search For a Market']

iOS: //*/UIALink[@label='Search For a Market']

Ancestral + índice (AncestInd): É uma expressão específica da plataforma e baseada no índice do elemento desejado contido em seu elemento ancestral. O índice define a exata posição do elemento, nesse caso, dentro do elemento ancestral. Essa expressão é híbrida ao considerar que o elemento ancestral é localizado por uma expressão de consulta relativa (baseada nos atributos chaves e identificadores) e o elemento interno é localizado pelo índice, portanto, posicionamento absoluto. Esse tipo de expressão é importante para localizar o elemento quando as expressões baseadas em atributos falharem devido aos valores dinâmicos de seus elementos alterados a cada nova execução da aplicação.

Android: //*[@resource-id='defaultView']/*[3]

iOS: //*[@label='Fresh-Food-Finder']/*[4]/*[1]/*[1] /*[1]

Ancestral + atributos (AncestAttr): Semelhante a expressão anterior, mas a localização por índice é substituída por uma localização baseada nos valores dos atributos chaves da plataforma específica.

Android: //*[@resource-id='defaultView']/* [@content-desc='Search For a Market']

iOS: //*[@label='Fresh-Food-Finder']/*[@label= 'Search For a Market']

Nem sempre uma expressão é aplicável em todas plataformas e suas versões. Cada plataforma pode apresentar uma estrutura de XML de GUI diferente, o que implica na seleção dos elementos. Com exceção da expressão multiplataforma, cada expressão possui uma versão adequada para cada plataforma (Android e iOS). O mecanismo único de teste a ser definido na próxima Seção, utiliza o tipo da plataforma para direcionar qual expressão utilizar. As expressões são utilizadas para selecionar os elementos durante o teste automatizado de um caso de teste.

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. The test case represented by the ESG is the basis for its construction, since each event contains data of a GUI element, as well as, its query expression to select it. A seguir são apresentadas duas estratégias que combinam os seis tipos de expressões para localizar corretamente o elemento para o teste automatizado.

Expressions in Order: As expressões são ordenadas por seu tipo e executadas sequencialmente. Na falha da primeira expressão, a próxima é executada, e assim sucessivamente. A ideia é garantir que o elemento seja localizado para não interromper a execução do caso de teste quando um elemento não é encontrado. A ordem definida para execução das expressões dá prioridade para expressões relativas, iniciando pela expressão multiplataforma devido sua adequação para selecionar elementos de GUI no Android e no iOS. Alguns estudos [39, 40, 41] sugerem fragilidades na localização de elemento por expressões absolutas, sendo assim, está ao final lista. A sequência definida para execução foi:

1° Multiplataforma;
2° Tipo do elemento;
3° Atributos identificadores;
4° Ancestral + atributos;
5° Ancestral + index;
6° Caminho absoluto.

A pseudocode of the test script proposed by the approach is presented in the Algorithm 1. The *ExecModel* method (line 2) receives, as parameters, the events mapped in the ESGs, and for each event it forwards its respective set of six XPath query expressions to the *ExecInOrder* method (line 9). This last method locates an element in the XML GUI structure of app using the expressions (lines 16-21). In the end, the found element (by some of the expressions) is executed according to the action indicated by the tester (line 25). When the element is not found an exception is thrown to fail the executing test case (line 23). Conditional logical structures lead the script flow to satisfy each platform.

Algorithm 1 Elements location based-Expressions in Order

```
 1: Input
    $events[]$ - AUT events mapping in ESG $deviceConfig$ - Data about
    device under test
 2: procedure ExecModel($events[]$, $deviceConfig$)

 3:     for each $evt$ in $events$ do
 4:         ExecAction($evt.xPathCrossPlatform$,
    $evt.xPathsAndroid$, $evt.xPathsiOS$, $deviceConfig.platform$);
 5:     end for
 6: end procedure
 7:
 8: Input
    $xPathCrossPlatform$ - Element XPath Cross-platform (CrossPlat)
    $xPathsAndroid[]$ - Element XPaths for Android (AbsPath, IdAttr,
    ElemType and AncestInd, AncestAttr)
    $xPathsiOS[]$ - Element XPaths for iOS (AbsPath, IdAttr, ElemType,
    AncestInd and AncestAttr)
    $platform$ - Platform name under test
 9: procedure ExecInOrder($xPathCrossPlatform$,
    $xPathsAndroid[]$, $xPathsiOS[]$, $platform$)

10:     $xPathSelectors[]$ = $xPathCrossPlatform$;

11:     if $platform$ = "Android" then
12:         $xPathSelectors[]$ += $xPathsAndroid$;
13:     else if $platform$ = "iOS" then
14:         $xPathSelectors[]$ += $xPathsiOS$;
15:     end if

16:     for each $selector$ in $xPathSelectors[]$ do
17:         $e$ = FindElementByXPath($selector$);
18:         if $e$ != null then
19:             break;
20:         end if
21:     end for

22:     if $e$ == null then
23:         throw new exception("Element not found");
24:     else
25:         $e.action()$;
26:     end if
27: end procedure
```

Multi-Locator: Todas as expressões são executadas e o elemento é selecionado a partir de critérios de votação. Tais critérios foram propostos baseados em estratégias de confiabilidade para determinar pesos para cada tipo de expressão. Essa abordagem foi adaptada de Leotta et al. [39], a qual é empregada no teste de aplicações Web. A seguir são apresentados os critérios de definição de pesos para as expressões. E além disso, a Tabela 3 relaciona os pesos definidos para cada expressão.

- Expressões de consulta baseadas em **valores de atributos visualizados na GUI** possuem peso alto, devido a constância em manter seus valores e não dependem de índices e/ou caminho absoluto. Exemplos desses atributos são o *content-desc*, *text*, *label* e *value*. As expressões contidas nesse grupo são: **Tipo do Elemento** e **Multiplataforma**;
- Expressões de consulta baseadas em **valores de atributos identificadores**, tais como *resource-id* (Android) e *name* (iOS) têm peso médio. Em alguns *frameworks* de desenvolvimento de apps, os valores dos atributos identificadores de elementos sofrem alterações durante as diferentes execuções, dessa forma a confiabilidade na seleção do elemento é comprometida. Um exemplo desse tipo de expresão é a **Ancestral + Index**;
- Expressões de consulta baseadas em **caminhos absolutos** ou que utilizam índices (posicionamento) têm peso pequeno. Possui uma confiança baixa, devido sua fragilidade em manter-se consistente após a evolução do aplicação [39, 40, 41]. Exemplos dessas expressões são: **Caminho absoluto** e **Ancestral + Index**.

Table 3. Tipos de expressão de consulta e pesos.

Tipo de Expressão	Confiabilidade	Peso
Multiplataforma	Alta	0,25
Tipo do elemento	Alta	0,25
Atributos identificadores	Média	0,15
Ancestral + atributos	Média	0,25
Caminho absoluto	Baixa	0,05
Ancestral + index	Baixa	0,05

A pseudocode of the test script proposed by the approach is presented in the Algorithm 2. Equal to approach Expression in Order, the *ExecModel* method (line 2) receives, as parameters, the events mapped in the ESGs, and for each event it forwards its respective set of six XPath query expressions to the *ExecMultiLocator* method (line 9) for locates an element in the XML GUI structure of app. Para cada elemento encontrado seu peso é extraído de acordo com o tipo da sua respectiva expressão de consulta. Elementos iguais retornados por expressões diferentes, têm os pesos acumulados. Ao final, o elemento localizado com maior votação é utilizado na execução do caso de teste (lines 31-32). When no element is found an exception is thrown to fail the executing test case (line 29).

Algorithm 2 Elements location based Multi-Locator

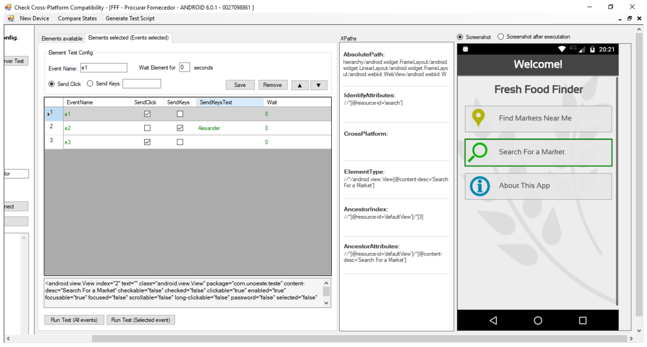
```
 1: Input
    $events[]$ - AUT events mapping in ESG $deviceConfig$ - Data about device
    under test
 2: procedure ExecInModel($events[]$, $deviceConfig$)

 3:     for each $evt$ in $events$ do
 4:         ExecAction($evt.xPathCrossPlatform$, $evt.xPathsAndroid$,
    $evt.xPathsiOS$, $deviceConfig.platform$);
 5:     end for
 6: end procedure
 7:
 8: Input
    $xPathCrossPlatform$ - Element XPath Cross-platform (CrossPlat)
    $xPathsAndroid[]$ - Element XPaths for Android (AbsPath, IdAttr,
    ElemType and AncestInd, AncestAttr)
    $xPathsiOS[]$ - Element XPaths for iOS (AbsPath, IdAttr, ElemType,
    AncestInd and AncestAttr)
    $platform$ - Platform name under test
 9: procedure ExecMultiLocator($xPathSelectorsCrossPlatform[]$,
    $xPathSelectorsAndroid[]$, $xPathSelectorsiOS[]$, $platform$)

10:     $xPathSelectors[] = xPathSelectorsCrossPlatform$;

11:     if $platform$ = "Android" then
12:         $xPathSelectors[]$ += $xPathSelectorsAndroid$;
13:     else if $platform$ = "iOS" then
14:         $xPathSelectors[]$ += $xPathSelectorsiOS$;
15:     end if

16:     $elements[]$;
17:     $voting[]$;

18:     for each $selector$ in $xPathSelectors[]$ do
19:         e = FindElementByXPath($selector$);
20:         if $e$ != null then
21:             if !elements.Contains(e) then
22:                 elements.Add(e);
23:                 voting[e] = 0;
24:             end if
25:             $voting[e] = voting[e]$ + ExtractWeight(selector);
26:         end if
27:     end for
28:     if $elements.length$ == null then
29:         throw new Exception("Element not found")
30:     else
31:         e = MaxElement($voting$);
32:         $e.action()$;
33:     end if
34: end procedure
```

4. Tool Implementation

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

Our tool was based on the Appium Framework [42], which is open source tool to automate test in native, Web or hybrid apps. 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 the app GUI elements. The events indicated by the tester are recorded and the GUI XML is extracted and treated by our approach for building of the test script using the appropriate XPath. The tool generates a test project for Microsoft Visual Studio encoded in C# with support for Unit Testing Framework [43]. Test project contains the test script appropriate to test the app in multiple settings. One last feature of the tool is to indicate for 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 6 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). Listings 1 and 2 show a code snippet generated by the tool and made compatible to test the same app under execution in multiple settings. Listings 1 uses strategy of expressions in order, and the second uses strategy multi-locator.

The test script can be used in other devices from multiple configurations *(Cn)*, ensuring the app test in a greater universe of devices. In this case, one option is to use it in clouds environments tests compatible with Appium, such as Amazon Device Farm [44], Bitbar [45] and TestObject [46]. These services offer a large number of real devices that can be connected and used in testing of cross-platform mobile apps.



Event Name: e1 Wait Element for 0 seconds

● Send Click ○ Send Keys:

	EventName	SendClick	SendKeys	SendKeysText	Wait
1	e1	☑	☐		0
2	e2	☐	☑	Alexander	0
3	e3	☑	☐		0

<android.view.View index="2" text="" class="android.view.View" package="com.unoeste.teste" content-desc="Search For a Market" checkable="false" checked="false" clickable="true" enabled="true" focusable="true" focused="false" scrollable="false" long-clickable="false" password="false" selected="false"

AbsolutePath: hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.webkit.WebView/android.webkit.W

IdentifyAttributes: //*[@resource-id='search']

CrossPlatform:

ElementType: //*/android.view.View[@content-desc='Search For a Market']

AncestorIndex: //*[@resource-id='defaultView']/*[3]

AncestorAttributes: //*[@resource-id='defaultView']/*[@content-desc='Search For a Market']

● Screenshot ○ Screenshot after executation

Figure 6. Screen tool for test element/events selection.

5. Evaluation

To evaluate the proposed approach, It is necessary to understand the performance of the expressions considered, to analyze their behavior with different apps, and to verify the results obtained from different configurations (in addition to the reference). In this way, we conducted an empirical evaluation, which addresses the following research questions (RQs) from a software testing perspective:

RQ1. *Qual a eficiência das abordagens na localização de elementos de GUI durante a execução dos casos de teste?*

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

Listing 1. Test script generated by the tool.

```
string[] selectors = new string[0];

if (ProjectConfig.PlataformName == "Android")
{
  selectors = new string[] {
    "//*[@content-desc='Search For a Market' or
         @label='Search For a Market']",
    "//*/android.view.View[@content-desc='Search For
         a Market']",
    "//*[@resource-id='search']", "//*[@resource-id='
         defaultView']/*[@content-desc='Search For a
         Market']",
    "//*[@resource-id='defaultView']/*[3]",
    "hierarchy/android.widget.FrameLayout/.../android
         .view.View/android.view.View[3]"};
}
else if (ProjectConfig.PlataformName == "iOS")
{
  selectors = new string[] {
    "//*[@content-desc='Search For a Market' or
         @label='Search For a Market']",
    "//*/UIALink[@label='Search For a Market']",
    "//*@name='Search For a Market'",
    "//*[@label='Fresh-Food-Finder']/*[@label='Search
          For a Market']",
    "//*[@label='Fresh-Food-Finder
         ']/*[4]/*[1]/*[1]/*[1]",
    "AppiumAUT/UIAApplication/UIAWindow/UIAScrollView
         /UIAWebView/UIALink[2]"};
}

string[] selectorsType = new string[] {"AbsolutePath",
      "IdentifyAttributes", "CrossPlatform", "
     ElementType", "AncestorIndex", "
     AncestorAttributes"};

IWebElement e = _locator.FindElementByXPathInOrder(
     selectors, selectorsType);

e.Click();

/*Insert your assert here*/
```

Listing 2. Test script generated by the tool.

```
string[] selectors = new string[0];

if (ProjectConfig.PlataformName == "Android")
{
  selectors = new string[] {
    "hierarchy/android.widget.FrameLayout/.../android
         .view.View/android.view.View[3]",
    "//*[@content-desc='Search For a Market' or
         @label='Search For a Market']",
    "//*/android.view.View[@content-desc='Search For
         a Market']",
    "//*[@resource-id='defaultView']/*[3]",
    "//*[@resource-id='defaultView']/*[@content-desc
         ='Search For a Market']"};
}
else if (ProjectConfig.PlataformName == "iOS")
{
  selectors = new string[] {
    "//*[@content-desc='Search For a Market' or
         @label='Search For a Market']",
    "//*/UIALink[@label='Search For a Market']",
    "//*@name='Search For a Market'",
    "//*[@label='Fresh-Food-Finder']/*[@label='Search
          For a Market']",
    "//*[@label='Fresh-Food-Finder
         ']/*[4]/*[1]/*[1]/*[1]",
    "AppiumAUT/UIAApplication/UIAWindow/UIAScrollView
         /UIAWebView/UIALink[2]"};
}

string[] selectorsType = new string[] {"AbsolutePath",
      "IdentifyAttributes", "CrossPlatform", "
     ElementType", "AncestorIndex", "
     AncestorAttributes"};


IWebElement e = _locator.
     FindElementByXPathMultiLocator(selectors,
     selectorsType);

e.Click();

/*Insert your assert here*/
```

O objetivo dessa primeira RQ é comparar a eficiência da localização de elementos de GUI entre os 6 tipos de expressões de consulta geradas pela ferramenta. Tais expressões estão agrupadas em dois grupos, absoluta e relativa. Uma expressão absoluta é mais simples, pois é baseada no caminho absoluto a partir do nó raiz, enquanto a relativa é baseada em caminhos alternativos, gerados a partir de combinações de caminhos, atributos e valores. A Seção 3 apresenta detalhes dos tipos de expressões de consulta definidas neste trabalho.

A métrica usada é o número de elementos localizados com sucesso por cada tipo de expressão na execução dos casos de teste dividido pelo total de elementos de GUI que são selecionados para executar o caso de teste por completo. A comparação entre esses valores indica o nível de eficiência das expressões na tentativa de localizar um elemento. Em complementação, será medido o número de execuções de casos de teste completadas com sucesso (sem ocorrência de falhas na execução do script de teste). Essas duas métricas serão computadas considerando os diversos dispositivos (múltiplas configurações).

QP2. *Qual o impacto das abordagens no tempo de execução dos casos de teste?*

O objetivo dessa RQ é demonstrar o tempo requerido para execução de um conjunto de casos de testes automatizados para execução nas múltiplas configurações.

A métrica usada é o tempo médio em milissegundos calculados a partir de 30 execuções dos casos de teste nas múltiplas configurações.

QP3. *Qual a combinação de tipos de expressão possui o melhor custo-benefício?*

Essa questão considera avaliar os tipos de expressões combinados, a fim de identificar a melhor combinação. Para responder a esta QP, os resultados da RQ1 e RQ2 serão analisados. No entanto, pode-se argumentar que uma combinação de quatro ou menos expressões individuais pode ser satisfatória. O script de teste será mantido principalmente pelos desenvolvedores à medida que o aplicativo evolui. É intuitivo que menos expressões sejam mais fáceis de manter.

A métrica usada é as expressões individuais executadas com sucesso (retornou um elemento esperado). As expressões sem sucesso (falhou) pode ser um indicativo de quais expressões realmente são necessárias dentro do conjunto de expressões combinadas.

5.1. Experimental Objects and Procedure

To answer our RQs we proceeded as follows. We have selected a set of cross-platform apps, three industrial apps and six 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 and native multi-platform apps and repositories on GitHub. Table 4 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 4. 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 5. Each app had its elements under test mapped with the use of the tool as approach discussed earlier.

Table 5. 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.

Quatro participantes independentes tiveram como objetivo executar as apps em seus dispositivos móveis e para cada uma identificar três funcionalidades. Além disso, para cada funcionalidade uma sequência de eventos representando exatamente a sequência de interações do usuário foi mapeada para um modelo ESG com auxílio da ferramenta MBTS4MA [47] e posteriormente reconstruída em nossa ferramente para geração do script de teste único. Essa sequência representa um de caso de teste capaz de testar a funcionalidade.

Para tal, utilize a ferramenta FourMa. Veja um exemplo a seguir:

A ferramenta FourMA será utilizada na construção dos modelos ESG para representar a sequência de eventos de uma funcionalidade sob teste. Ela pode ser obtida em [https://goo.gl/qh9KE3]. Realize seu download e execute-a usando o comando java -jar FourMA.jar. Ela possui a seguinte interface

5.2. Analysis of Results and Findings

...

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 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 [48].

6. Related Work

In this section, we describe the most relevant research from area. Fazzini et al. [26] 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 [49]. 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 [50]), 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. [51] 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. [52] 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. [29] 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 implemented, 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 [**?**] and Apportable [53]. 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 [54] 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 [55] is an Integrated Development Environment (IDE) designed for mobile test automation development and execution using on Appium framework [42] 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

...

Bibliography

[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 March 2017. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2017. Accessed on 2017-09-11.

[7] 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.

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

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

[10] 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.

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

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

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

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

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

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

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

[18] ReactNative. ReactNative. https://facebook.github.io/react-native/, 2017. Accessed on 2017-04-05.

[19] ReactXP. ReactXP. https://microsoft.github.io/reactxp/, 2017. Accessed on 2017-04-05.

[20] NativeScript. NativeScript. https://www.nativescript.org, 2017. Accessed on 2017-04-05.

[21] Weex. Weex. https://weex.incubator.apache.org/, 2017. Accessed on 2017-04-05.

[22] Michiel Willocx, Jan Vossaert, and Vincent Naessens. Comparing performance parameters of mobile app development strategies. In *Proceedings of the International Workshop on Mobile Software Engineering and Systems - MOBILESoft '16*, pages 38–47, 2016.

[23] M Willocx, J Vossaert, and V Naessens. A Quantitative Assessment of Performance in Mobile App Development Tools. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 454–461, 2015.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Mona Erfani Joorabchi, Mohamed Ali, and Ali Mesbah. Detecting inconsistencies in multi-platform mo-

bile apps. In *IEEE 26th International Symposium on Software Reliability Engineering (ISSRE 2015)*, pages 450–460. IEEE, nov 2015.

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

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

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

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

[34] 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.

[35] 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.

[36] 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.

[37] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. Comparing the maintainability of selenium webdriver test suites employing different locators: A case study. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation*, JAMAICA 2013, pages 53–58, 2013.

[38] M Leotta, D Clerissi, F Ricca, and C Spadaro. Repairing Selenium Test Cases: An Industrial Case Study about Web Page Element Localization. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 487–488, 2013.

[39] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Using Multi-Locators to Increase the Robustness of Web Test Cases. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, number April, pages 1–10. IEEE, apr 2015.

[40] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Reducing web test cases aging by means of robust XPath locators. *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, pages 449–454, 2014.

[41] Gururaja Rao and Arvind Pachunoori. Optimized Identification Techniques Using XPath. Technical Report MSU-CSE-00-2, IBM Developerworks, 2013.

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

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

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

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

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

[47] Guilherme de Cleva; Farto and Andre Takeshi Endo. Reuse of model-based tests in mobile apps. *Simposio Brasileiro de Engenharia de Software (SBES)*, 2017.

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

[49] Google. Espresso. https://google.github.io/android-testing-support-library, 2017. Accessed on 2017-04-14.

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

[51] 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.

[52] 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.

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

[54] 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.

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