

Uma avaliação de testes automatizados para aplicações móveis híbridas

André Augusto Menegassi^{*†}, André Takeshi Endo^{*}

^{*}UTFPR - Universidade Tecnológica Federal do Paraná

Campus Cornélio Procopio - PR

[†]UNOESTE - Universidade do Oeste Paulista

Presidente Prudente - SP

Email: ^{*}andremenegassi@unoeste.br, [†]andreendo@utfpr.edu.br

Resumo—Atualmente, os dispositivos móveis estão disponíveis em vários formatos; suas aplicações podem ser agrupadas em três categorias: aplicações nativas, aplicações Web e aplicações híbridas. As aplicações híbridas utilizam recursos da programação Web, como HTML, CSS e Javascript, e são capazes de acessar recursos nativos do dispositivo, tais como GPS, câmera e agenda de contatos. Diversas pesquisas tratam de teste de software automatizado em aplicações nativas e tal fato indica uma carência de abordagens de teste para aplicações híbridas. O objetivo do artigo foi investigar o processo de automatização de teste de interface de usuário (GUI) em aplicações híbridas sob execução em diversas plataformas móveis. Três aplicações foram testadas em cinco dispositivos móveis (Android e iOS) diferentes. As inspeções manuais indicaram funcionamento correto das funcionalidades sob teste. As execuções automatizadas encontraram algumas diferenças entre as plataformas, tais como na interface de usuário e na codificação do script de teste.

Abstract—Currently, mobile devices are available in several formats; their applications can be grouped into three categories: native applications, Web applications and hybrid applications. Hybrid applications use Web programming features, such as HTML, CSS and Javascript, and are able to access native device capabilities, such as GPS, camera and contacts. Several works automated software testing in native applications and this fact suggests a lack of testing approaches for hybrid applications. In this paper, we investigate test automation process in hybrid applications running graphical user interface (GUI) under various mobile platforms. Three applications were tested in five different mobile devices (Android and iOS). The manual inspections indicated correct functioning of the features under test. Automated executions found some differences between the platforms, such as in the user interface and test script coding.

I. INTRODUÇÃO

Os dispositivos móveis estão disponíveis em vários formatos como *smartphones*, *tablets* e *wearables*¹, e equipados com poderosos processadores, ampla capacidade de armazenamento de dados e diversos sensores [1]. Modernos sistemas operacionais controlam o hardware destes dispositivos e servem como plataforma para o desenvolvimento de softwares denominados aplicações móveis ou simplesmente "*apps*". Na pesquisa da International Data Corporation [2] sobre o *Market Share* no uso dos sistemas operacionais, as plataformas Android, Apple iOS e WindowsPhone foram as mais consumidas no ano de 2015.

¹Dispositivos eletrônicos vestíveis como óculos e relógios.

Desde o lançamento do primeiro iPhone pela Apple, a computação móvel tem avançado a nível de hardware e software. No mercado de desenvolvimento de software, as aplicações para dispositivos móveis ganharam destaque, sendo ofertadas e disponibilizadas em modernas lojas de distribuição. O site Statista [3] oferece estatísticas de julho de 2015 sobre o número de aplicações disponíveis para download nas principais lojas de distribuição. A plataforma Android² possui o maior número de aplicações disponíveis para seus usuários, em segundo está Apple³ e na última colocação a loja da Microsoft⁴.

As aplicações móveis podem ser agrupadas em três categorias: (i) *aplicações nativas*: Desenvolvidas utilizando a plataforma nativa do dispositivo, e com acesso direto a funções do sistema operacional móvel sem utilização de recursos intermediário; (ii) *aplicações Web*: Desenvolvidas com tecnologias empregadas na construção de software para Web como HTML5, CSS3 e Javascript. São executadas sob o navegador de Internet e não tem acesso a recursos avançados do sistema operacional; e (iii) *aplicações híbridas*: Combina recursos comuns das aplicações Web como HTML5, CSS3, Javascript e com suporte direto às APIs nativas do sistema operacional, assim como as aplicações nativas. Uma vantagem é desenvolver uma vez e implantá-la em diversas plataformas.

Vários requisitos exclusivos distinguem aplicações móveis das aplicações convencionais. As aplicações móveis são oferecidas para um conjunto de dispositivos e plataformas diferentes, com características diferentes tais como tamanho de telas, recursos de bateria, canais de entrada (teclado, voz e gestos), e ainda devem prover uma experiência rica de interface ao usuário [4]. Para Amalfitano et al. [5], a qualidade das aplicações móveis é pobre, e na maioria das vezes devido ao rápido processo de desenvolvimento adotado, no qual as atividades de teste são negligenciadas ou aplicadas de forma superficial por serem consideradas complexas, difíceis de automatizar, caras e/ou demoradas. Os autores ainda acrescentam que a crescente complexidade das aplicações móveis exigem novas técnicas e ferramentas de teste, bem como novos processos da engenharia

²<https://www.android.com>

³<http://www.apple.com>

⁴<https://www.microsoft.com/pt-br/windows/phones>

de software, tudo para assegurar o desenvolvimento seguro e de alta qualidade das aplicações móveis.

Como ocorre no desenvolvimento de software tradicional, as aplicações móveis precisam passar por atividades de teste para assegurar a qualidade do produto final. Se um defeito foi encontrado após disponibilização na loja da plataforma, o usuário poderá classificá-la negativamente. Após a correção, uma nova submissão deverá ser realizada para nova aprovação, o que pode demorar alguns dias, retardando o acesso dos usuários a nova versão. E também não é possível obrigar o usuário a realizar a atualização [6].

Estas atividades aumentam o custo e o tempo de desenvolvimento da aplicação, principalmente em projetos que objetivam a execução da aplicação em ambientes multiplataformas. O emprego de testes automatizados é uma boa prática para o aumento da produtividade e para diminuição dos custos, além de serem reproduzidos diversas vezes com possibilidade de avaliação dos resultados gerados [7].

No mapeamento sistemático conduzido por Zein et al. [8], com propósito de identificar pesquisas existentes para técnicas de teste em aplicações móveis, 79 pesquisas foram incluídas, avaliadas e discutidas. As técnicas de testes identificadas estão focadas em aplicações móveis nativas, salvo a exceção do trabalho de Gao et al. [4], no qual faz um comparativo sobre testes de software entre aplicações nativas e aplicações Web. Dessa forma, existe uma carência de estudos para técnicas de testes automatizados em aplicações móveis híbridas e multiplataformas.

O objetivo geral deste estudo é investigar o processo de automatização de teste de interface gráfica de usuário (GUI) em aplicações híbridas sob execução em diversas plataformas móveis (*cross-platform*). Especificamente, com intento de antecipar a identificação de defeitos e incompatibilidade entre plataformas, será investigado a compatibilidade de comportamento e apresentação visual para usuário em diferentes plataformas e dispositivos móveis das aplicações híbridas.

As seções deste artigo estão organizadas da seguinte forma. Na Seção 2 é apresentada uma breve revisão bibliográfica sobre as categorias de aplicações móveis. Na Seção 3, são apresentados os métodos utilizados para testar as aplicações móveis híbridas em diversos dispositivos. Na Seção 4, os resultados desta pesquisa são apresentados e discutidos. Na Seção 5, os principais trabalhos relacionados são discutidos. E por fim, a Seção 8 conclui o artigo.

II. REVISÃO BIBLIOGRÁFICA

A construção de aplicações móveis está classificada em três grupos, aplicações nativas, aplicações Web e aplicações híbridas [2]:

(a) aplicações nativas: Aplicações nativas possuem arquivos binários executáveis instalados e armazenados diretamente para o dispositivo. A maneira mais popular para obter e instalar uma aplicação nativa é visitando a loja de distribuição de aplicações da plataforma móvel. Uma aplicação nativa é codificada em um *Software Development Kit (SDK)*. Além disso, normalmente fornecido pela organização proprietária

do sistema operacional móvel. A aplicação tem acesso direto às funções do sistema operacional, sem qualquer recurso intermediário, sendo livre para acessar todas as *Application Programming Interface (API)* que são disponibilizadas [2]. As diferentes ferramentas, linguagens de programação e o formato de empacotamento da aplicação estão relacionados na Tabela I.

Tabela I: Sistema Operacionais e detalhes de desenvolvimento (adaptada de [2])

	iOS	Android	WindowsPhone
Linguagem	Objective-C, C, C++ e swift	Java (em alguns casos C e C++)	C# e VB.NET
Ferramentas	XCode	Android SDK e Android Studio	Visual Studio e Windows Phone development tools
Formato empacotamento	.app	.apk	.xap

Uma desvantagem deste tipo de aplicação é a incompatibilidade como os diversos sistemas operacionais. Por exemplo, uma aplicação escrita para Apple iOS não é executável em outro sistema operacional, sendo necessário uma nova implementação em um novo ambiente de desenvolvimento e linguagem de programação.

O desenvolvimento de uma aplicação para somente uma plataforma diminui o possível número de usuários. No entanto, o desenvolvimento para diversas plataformas envolve a criação de diversos times de desenvolvedores específicos para cada plataforma, gerando redundância e aumentando os custos envolvidos no projeto [9].

Uma solução para as desvantagens citadas é o uso de *frameworks* de desenvolvimento nativo capazes de traduzir uma linguagem nativa para outra linguagem compatível com outras plataformas. Um exemplo deste tipo de *framework* é o Xamarin⁵ e o Apportable⁶ [1]. No caso do Xamarin, a aplicação é desenvolvida para Windows Phone e traduzida para Apple iOS ou Android, e o Apportable traduz código construído para Apple iOS para Android.

(b) aplicações Web: Os atuais dispositivos móveis oferecem modernos e consistentes navegadores Web, com suporte aos novos recursos do HTML5 [10], CSS3 [10] e Javascript [10]. A especificação do HTML5 inclui recursos de geolocalização, armazenamento local *off-line*, formatos de mídias (áudio e vídeo), além de avançados componentes de interface de usuário [2]. É um tipo de aplicação construída com recursos da Web e para executar diretamente no navegador (baseada no navegador/*browser-based*) do dispositivo. Uma desvantagem deste tipo de aplicação é a ausência de acesso aos vários recursos oferecidos pelo sistema operacional móvel, limitando o tipo de aplicação que pode ser construída. Além disso, as aplicações não estão disponíveis nas lojas de distribuição. Como vantagem é possível citar a disponibilização da mesma aplicação para todas as plataformas móveis (*cross-platform*),

⁵<https://xamarin.com>

⁶<http://www.apportable.com>

como também o clássico *Desktop*, devido às características de portabilidade oferecidas pelas tecnologias Web utilizadas.

(c) **aplicações híbridas:** Combina características das aplicações nativas e Web. As aplicações são desenvolvidas usando os recursos comuns das aplicações Web como HTML5, CSS3, Javascript e com suporte direto às APIs nativas do sistema operacional móvel por meio do uso de uma ponte que permite a aplicação híbrida tirar o máximo de todas as características que o dispositivo têm para oferecer [2]. A aplicação é dividida em duas partes, uma nativa e outra conhecida como *WebView*, responsável por executar o HTML, CSS e Javascript.

Neste contexto, são utilizados *frameworks* de desenvolvimento multiplataforma como PhoneGap⁷/Cordova⁸, Sencha⁹ e IBM MobileFirst¹⁰. O desenvolvedor especifica a lógica da aplicação e interface do usuário usando componentes do desenvolvimento Web. No entanto, estes componentes não possuem suporte nativo a todos recursos disponibilizados pelo dispositivo e seu sistema operacional, assim o *framework* multiplataforma provê suporte aos recursos nativos do dispositivo [1]. O resultado final é uma aplicação híbrida que pode ser disponibilizada nas lojas de distribuição de todas as plataformas móveis. A porção Web das aplicações híbridas pode ser uma página Web armazenada num servidor ou arquivos HTML, CSS e Javascript empacotados e armazenados localmente dentro da aplicação.

III. CONFIGURAÇÃO DO ESTUDO

O objetivo geral deste estudo é investigar o processo de automatização de teste GUI em aplicações híbridas sob execução em diversas plataformas móveis. Especificamente, com intento de antecipar a identificação de defeitos e incompatibilidade entre plataformas.

Do ponto de vista do teste de software, foram consideradas as seguintes questões de pesquisa (QP):

- **QP1** - Uma aplicação móvel híbrida é realmente compatível com diversas configurações (diferentes plataformas e dispositivos)?
- **QP2** - Uma aplicação móvel híbrida é executada, do ponto de vista funcional, corretamente (se comporta da mesma forma) em cada configuração?
- **QP3** - Uma aplicação móvel híbrida apresenta um desempenho similar em cada configuração durante o teste automatizado?
- **QP4** - Uma aplicação móvel híbrida se apresenta para usuário da mesma forma (GUI) em cada configuração?
- **QP5** - A escrita (codificação) e execução do *script* de teste é compatível em cada configuração?

As questões objetivam obter respostas sobre algumas hipóteses relacionadas às atividades de testes em aplicações móveis híbridas. A **QP1** avalia se o *framework* de desenvolvimento é realmente capaz de gerar aplicações multiplataformas para

diversas configurações. A **QP2** avalia as funcionalidades da aplicação sob teste, identificando se continuam a atender seus requisitos em plataformas diferentes. A **QP3** avalia o desempenho durante a execução nas diversas configurações resultando o tempo de execução dos casos de teste. A **QP4** objetiva responder se a interface gráfica da aplicação, sob a percepção do usuário, mantém características similares nas diversas configurações. Por fim, a **QP5** objetiva responder se o *script* de teste codificado é executado igualmente nas distintas configurações.

Para responder tais questões alguns passos foram necessários.

Os testes foram aplicados nas duas principais plataformas móveis de mercado e em algumas diferentes configurações apresentadas na Tabela II.

Tabela II: Plataformas e dispositivos utilizados na pesquisa

Dispositivo	Modelo	Sistema Operacional	Tela (pol.)	Processador	Memória RAM
SPAnd6_4I	Smart-phone Nexus 5	Android 6.0.1 (versão atual)	4,95	2,27 GHz (4 core)	1.854 GB
SPAnd5_5I	Smart-phone MotoG1	Android 5.1	5	1.19 GHz (4 core)	1 GB
TBAnd4_7I	Tablet Samsung	Android 4.4.4	7	1.3 GHz (4 core)	1 GB
SPIOS7_3I	iPhone 4	iOS 7.1.2	3,5	1 GHz (2 core)	512 MB
TBIO59_9I	iPad 2	iOS 9.3 (versão atual)	9,7	1 GHz (2 core)	512 MB

Inicialmente para realização dos testes, os pesquisadores selecionaram algumas aplicações híbridas desenvolvidas com suporte a execução multiplataforma e com código-fonte disponível no GITHUB¹¹. O código-fonte de uma terceira aplicação proprietária foi gentilmente cedida pela Escolha Tecnologia [11]. As aplicações foram recompiladas para as plataformas Android e iOS, o resultado foi o binário para a instalação. Os pesquisadores configuram o ambiente de desenvolvimento nativo para ambas plataformas, uma vez que a compilação ocorre pelo ambiente proprietário da plataforma.

As aplicações selecionadas para teste estão relacionadas na Tabela III, na qual destaca o propósito da aplicação, *framework* de desenvolvimento híbrido multiplataforma utilizado, bibliotecas Javascript utilizadas, e endereço eletrônico do repositório do código-fonte no GITHUB.

Um cenário de teste foi definido para coletar os resultados por meio da instalação, execução manual e automatizada das aplicações selecionadas. Para tal, foi utilizado uma máquina com o sistema operacional Windows 10 para executar os *scripts* de testes codificados em linguagem C# (com suporte a Unit Testing Framework [13]) e o Selenium WebDriver¹². E dois computadores desempenhando o papel de servidores de testes, sendo um operando com sistema Windows 10 (servidor 1) e outro operando com o sistema OS X El Capitan (servidor 2).

⁷<http://phonegap.com>

⁸<https://cordova.apache.org>

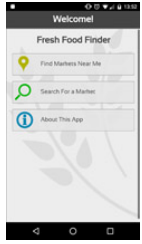
⁹<https://www.sencha.com>

¹⁰<http://www.ibm.com/mobilefirst/us/en>

¹¹<https://github.com>

¹²<https://www.nuget.org/packages/Selenium.WebDriver>

Tabela III: Aplicações selecionadas

Aplicações	Tela Inicial
Fresh Food Finder É uma aplicação para localizar mercados de alimentos frescos. Framework: Apache Cordova/PhoneGap Bibliotecas: jQuery GITHUB: https://github.com/triceam/Fresh-Food-Finder	
PedidoApp Aplicação de exemplo extraída de [12]. Framework: Apache Cordova/PhoneGap Bibliotecas: AngularJS e Ionic GITHUB: https://github.com/sergiolopes/pedidapp	
MemesPlay Aplicação que reúne vídeos engraçados e virais da Internet. Framework: Apache Cordova/PhoneGap Bibliotecas: ReactJS e jQuery Propriedade da Escolha Tecnologia [11]. SITE: http://www.escolhatecnologia.com.br	

A ferramenta Selenium¹³ foi selecionada devido sua ampla disseminação na comunidade técnicas, com diversos exemplos disponíveis em sites especializados e suporte à geração de *scripts* de teste em diversos ambientes de desenvolvimento. O *framework open source* Appium¹⁴, juntamente com a Selenium dão suporte a automação de teste em aplicações móveis nativas, aplicações Web e híbridas em ambientes multiplataformas, sendo esta última o foco deste estudo.

Uma versão específica do Appium (de acordo com o sistema operacional) foi instalada e configurada em dois servidores de teste para acesso via rede de computadores local. Os dispositivos móveis (*smartphones* e *tablets*) foram conectados aos servidores de teste por cabo de dados específicos para seu hardware, e assim as aplicações sob teste foram instaladas. A visão geral do cenário de teste preparado para execução dos testes das aplicações móveis está ilustrada na Figura 1.

O *script* de teste foi executado na Máquina de Teste, e via rede de computadores conecta-se ao Appium instalado nos servidores de teste. O Appium recebe as requisições vindas da Máquina de Teste (também pode ser chamada de cliente Appium), e atende e executa comandos diretamente no dispositivo móvel.

Os softwares utilizados no cenário de teste foram:

- XCode 7.3;

¹³<http://www.selenium.org>

¹⁴<http://appium.io/>

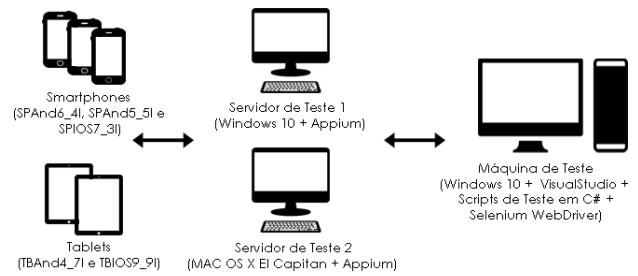


Figura 1: Cenário de teste

- Android SDK r24.4.1;
- VisualStudio 2015;
- Appium 1.4.3;
- Selenium WebDriver for C# 2.53.0;
- Appium WebDriver 1.5.0.1;
- Windows 10;
- MAC OS X El Capitan.

As interfaces de usuário das aplicações são representadas por estruturas XML. Tais estruturas mapeiam cada elemento de interface, sendo acessados programaticamente por uso de expressões de consulta XPath¹⁵, utilizadas no *script* de teste.

Para responder a QP4 foram capturados *screenshots* durante a execução dos *scripts* de teste, e suas dimensões padronizadas para resolução de 512 pixels de largura por 853 pixel de altura. As imagens foram comparadas com um algoritmo diferencial da biblioteca Resemble.js[14], resultando em uma imagem diferencial e um valor percentual. Tais dados demonstram a diferença entre as imagens. Somente foi utilizado o *screenshot* da primeira tela de cada aplicação. Nesta abordagem foi necessário estabelecer uma referência (imagem matriz) para realização das comparações. A *screenshot* do dispositivo SPAnd6_4I foi selecionada como referência.

IV. ANÁLISE DOS RESULTADOS

Nesta Seção são apresentados e discutidos os resultados coletados por meio da execução do método descrito na Seção anterior, objetivando responder as questões de pesquisa do presente estudo.

Existe um empenho da comunidade em desenvolver abordagens que automatizem a geração de casos de teste tornando o processo de teste sistemático e formal, e neste contexto o Teste Baseado em Modelo (TBM) vem sendo pesquisado [15]. No TBM, os casos de testes são derivados a partir de um modelo usado para descrever o software a ser testado ou parte dele (tal como uma funcionalidade) [15]. Uma técnica para construção de modelagens é o *Event Sequence Graph (ESG)*, que mapeia os eventos e as sequências válidas de transição entre os eventos da funcionalidade sob teste. Um caso de teste e um ESG foi definido para uma funcionalidade de cada aplicação. A execução manual da funcionalidade foi conduzida por um pesquisador. Para o teste automatizado

¹⁵Xpath é uma linguagem de consulta para selecionar elementos (nós) em estruturas computacionais chamadas de árvore representando documentos XML. Site: <https://www.w3.org/TR/xpath>.

foi desenvolvido um único *script* de teste (por aplicação) maximizando a compatibilidade de sua execução nos diversos dispositivos.

a) Fresh Food Finder

Para a aplicação Fresh Food Finder foi elaborado um caso de teste com objetivo de testar a funcionalidade referente a localização de um determinado fornecedor de comida. O caso de teste está descrito na Tabela IV.

Os eventos e as sequências válidas de transição entre os eventos da funcionalidade sob teste da presente aplicação móvel estão apresentados no ESG da Figura 2. Por fim, o *script* de teste escrito em linguagem C# para as plataformas Android e iOS está apresentado no Algoritmo 1 e também disponível em um repositório no GITHUB [16].

Tabela IV: Caso de teste da aplicação Fresh Food Finder

Teste Id	FFF-01
Objetivo	Encontrar um fornecedor de comida.
Pré-condição	Fornecedores cadastrados na base de dados.
Entradas 1	"xpto"
Saídas 1	Lista de resultados vazia.
Entradas 2	"alexander"
Saídas 2	Lista com dados de ao menos um fornecedor.

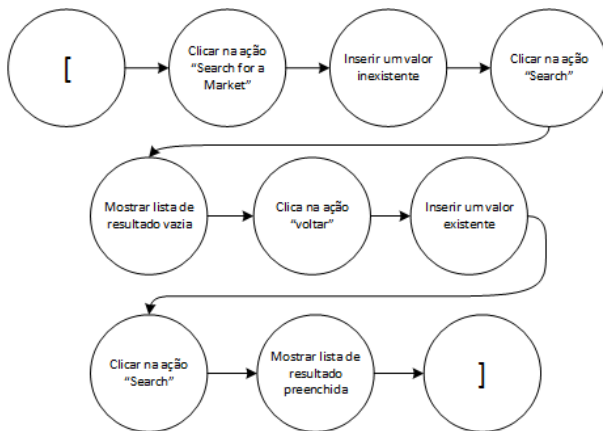


Figura 2: ESG aplicado ao caso de teste FFF-01

Algoritmo 1: Trecho do script do caso de teste FFF-01 aplicado à plataforma Android e iOS

```

1 //-----
2 //EVENTO: CLICAR NA OPÇÃO "SEARCH FOR A MARKET"
3 //-----
4 By btnSearchMarketXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
5 {
6     XPathAndroid4 = "//android.view.View[@content-desc='Search For a Market Link']";
7     XPathAndroid5 = "//android.view.View[@content-desc='Search For a Market']";
8     XPathAndroid6 = "//android.view.View[@content-desc='Search For a Market']";
9     XPathAndroid7 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
10    XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
11    XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
12};
13 AppiumHelpers.Wait(10, btnSearchMarketXPath, true);
14 AppiumHelpers.GetScreenshot();
15 var btnSearchMarket = _driver.FindElement(btnSearchMarketXPath);
16 AppiumHelpers.GetCountXMLNodes(_driver.PageSource, true);
17 AppiumHelpers.Tap(btnSearchMarket);
18
19 //-----
20 //EVENTO: INSERIR UM VALOR INEXISTENTE
21 //-----

```

```

23 By inputSearchXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
24 {
25     XPathAndroid4 = "//android.view.View/android.widget.EditText",
26     XPathAndroid5 = "//android.view.View/android.widget.EditText",
27     XPathAndroid6 = "//android.view.View/android.widget.EditText",
28     XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]",
29     XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
30};
31 AppiumHelpers.Wait(20, inputSearchXPath);
32 AppiumHelpers.GetScreenshot();
33 var inputSearch = (_driver).FindElement(inputSearchXPath);
34 AppiumHelpers.Tap(inputSearch);
35 inputSearch.SendKeys("xpto");
36
37 //-----
38 //EVENTO: CLICAR NA AÇÃO "SEARCH"
39 //-----
40 By btnSearchXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
41 {
42     XPathAndroid4 = "//android.view.View[@content-desc='Search Link']";
43     XPathAndroid5 = "//android.view.View[@content-desc='Search']";
44     XPathAndroid6 = "//android.view.View[@content-desc='Search']";
45     XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
46     XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
47};
48 AppiumHelpers.Wait(10, btnSearchXPath);
49 AppiumHelpers.GetScreenshot();
50 var btnSearch = _driver.FindElement(btnSearchXPath);
51 AppiumHelpers.GetCountXMLNodes(_driver.PageSource);
52 AppiumHelpers.Tap(btnSearch);
53
54 //-----
55 //EVENTO: MOSTRAR LISTA DE RESULTADO VAZIA
56 //-----
57 By listViewItemsXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
58 {
59     XPathAndroid4 = "//android.webkit.WebView/android.widget.ListView/android.view.View";
60     XPathAndroid5 = "//android.webkit.WebView/android.widget.ListView/android.view.View";
61     XPathAndroid6 = "//android.webkit.WebView/android.widget.ListView/android.view.View";
62     XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
63     XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
64};
65 AppiumHelpers.GetScreenshot();
66 IReadOnlyList<IWebElement> listViewItems = _driver.FindElements(listViewItemsXPath);
67 AppiumHelpers.GetCountXMLNodes(_driver.PageSource);
68 if (AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.IOS9)
69     Assert.AreEqual(listViewItems.Count, 4); //vazia
70 else if (AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.IOS9)
71     Assert.AreEqual(listViewItems.Count, 5); //vazia
72 else Assert.AreEqual(listViewItems.Count, 0); //vazia
73
74 //-----
75 //EVENTO: CLICAR NA AÇÃO "VOLTAR"
76 //-----
77 if (AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.IOS9)
78 {
79     _driver.Tap(1, 25, 25, 1000);
80 }
81 else
82 {
83     By btnBackXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
84     {
85         XPathAndroid4 = "//android.view.View[@content-desc='Back']";
86         XPathAndroid5 = "//android.webkit.WebView/android.view.View";
87         XPathAndroid6 = "//android.webkit.WebView/android.view.View";
88         XPathIOS7 = "ERROR: elemento nao disponivel no XML";
89         XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[2]";
90     };
91 AppiumHelpers.Wait(10, btnBackXPath);
92 AppiumHelpers.GetScreenshot();
93 IReadOnlyList<IWebElement> btnBack = _driver.FindElements(btnBackXPath);
94 AppiumHelpers.GetCountXMLNodes(_driver.PageSource);
95
96 if (AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.Android4 || AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.IOS9)
97     AppiumHelpers.Tap(btnBack[0]);
98 else AppiumHelpers.Tap(btnBack[1]);
99
100 //-----
101 //EVENTO: INSERIR UM VALOR EXISTENTE
102 //-----
103 inputSearchXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
104 {
105     XPathAndroid4 = "//android.view.View/android.widget.EditText";
106     XPathAndroid5 = "//android.view.View/android.widget.EditText";
107     XPathAndroid6 = "//android.view.View/android.widget.EditText";
108     XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
109     XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAWebView[1]/UIAWebView[1]/UIAStaticText[1]";
110};
111 AppiumHelpers.Wait(10, inputSearchXPath);
112 AppiumHelpers.GetScreenshot();
113 inputSearch = _driver.FindElement(inputSearchXPath);
114 AppiumHelpers.Tap(inputSearch);

```

```

120 inputSearch.SendKeys("");
121 inputSearch.Clear();
122 inputSearch.SendKeys("alexander");
123
124
125 //-----
126 //EVENTO: CLICAR NA AÇÃO "SEARCH"
127 //-----
128 btnSearchXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
129 {
130     XPathAndroid4 = "//android.view.View[@content-desc='Search Link']",
131     XPathAndroid5 = "//android.view.View[@content-desc='Search']",
132     XPathAndroid6 = "//android.view.View[@content-desc='Search']",
133     XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAScrollView[1]/UIAWebView
134         [1]/UIALink[1]/UIAStaticText[1]",
135     XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAScrollView[1]/UIAWebView
136         [1]/UIALink[1]"
137 };
138 AppiumHelpers.Wait(10, btnSearchXPath);
139 AppiumHelpers.GetScreenshot();
140 btnSearch = _driver.FindElement(btnSearchXPath);
141 AppiumHelpers.GetCountXMLNodes(_driver.PageSource);
142 AppiumHelpers.Tap(btnSearch);
143
144 //-----
145 //EVENTO: MOSTRAR LISTA DE RESULTADO PREENCHIDA
146 //-----
147 listViewItemsXPath = AppiumHelpers.BuildXPathPlatform(new XPathPlatform())
148 {
149     XPathAndroid4 = "//android.webkit.WebView/android.view.View/android.widget.
150         ListView/android.view.View",
151     XPathAndroid5 = "//android.webkit.WebView/android.widget.ListView/android.
152         view.View",
153     XPathAndroid6 = "//android.webkit.WebView/android.widget.ListView/android.
154         view.View",
155     XPathIOS7 = "//UIAApplication[1]/UIAWindow[1]/UIAScrollView[1]/UIAWebView
156         [1]/*",
157     XPathIOS9 = "//UIAApplication[1]/UIAWindow[1]/UIAScrollView[1]/UIAWebView
158         [1]/*"
159 };
160 AppiumHelpers.Wait(10, listViewItemsXPath);
161 AppiumHelpers.GetScreenshot();
162 listViewItems = _driver.FindElements(listViewItemsXPath);
163 AppiumHelpers.GetCountXMLNodes(_driver.PageSource);
164 //Verifica se a quantidade de elementos do resultado eh diferente de zero (0).
165 Se sim, entao foram encontrados resultados.
166 if (AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.IOS7)
167     Assert.AreEqual(listViewItems.Count, 4); //tem resultado
168 else if (AppiumHelpers.PlatformTesting == AppiumHelpers.Platforms.IOS9)
169     Assert.AreEqual(listViewItems.Count, 5); //tem resultado
170 else Assert.AreEqual(listViewItems.Count, 0); //tem resultado

```

b) PedidoApp

O método apresentado para aplicação móvel anterior também foi utilizado para a aplicação PedidoApp. O caso de teste está descrito na Tabela V, com objetivo de testar a funcionalidade referente à compra/pedido de um bolo.

Os eventos e as sequências válidas de transição entre os eventos da funcionalidade sob teste da presente aplicação móvel estão apresentados no ESG da Figura 3. Por fim, o *script* de teste escrito em linguagem C# para as plataformas Android e iOS está disponível em um repositório no GITHUB [16].

Tabela V: Caso de teste da aplicação PedidoApp

Teste Id	PAPP-01
Objetivo	Pedir um bolo.
Pré-condição	Ter bolos cadastrados na base de dados.
Entradas	Selecionar o primeiro item da lista de bolos ofertados.
Saídas	Confirmação da realização do pedido.

c) MemesPlay

Igualmente às outras aplicações sob teste, foi utilizado o mesmo método para realização dos testes para aplicação MemesPlay. O caso de teste está descrito na Tabela VI, com objetivo de testar a funcionalidade referente à pesquisa de conteúdo multimídia.

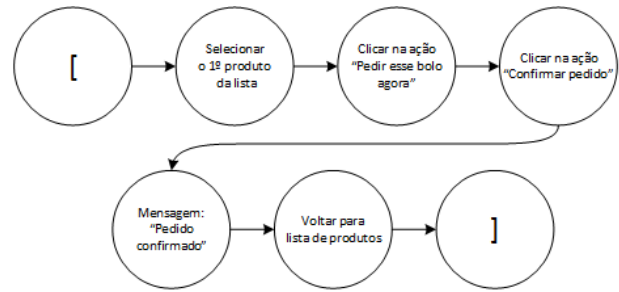


Figura 3: ESG aplicado ao caso de teste PAPP-01

Os eventos e as sequências válidas de transição entre os eventos da funcionalidade sob teste da presente aplicação móvel estão apresentados no ESG da Figura 4. Por fim, o *script* de teste escrito em linguagem C# para as plataformas Android e iOS está disponível em um repositório no GITHUB [16].

Tabela VI: Caso de teste da aplicação MemesPlay

Teste Id	MP-01
Objetivo	Localizar um conteúdo multimídia.
Pré-condição	Existir conteúdo disponível para pesquisa.
Entrada 1	"xpto"
Saída 1	Lista de resultado vazia.
Entrada 2	"vídeo"
Saída 2	Lista com o resultado da pesquisa.

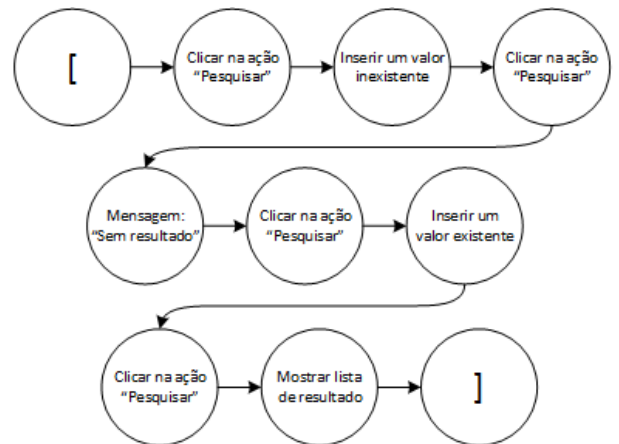


Figura 4: ESG aplicado ao caso de teste MP-01

A seguir, para cada questão de pesquisa são apresentados e discutidos os resultados extraídos durante a execução manual e automatizadas das aplicações.

QP1 - Uma aplicação móvel híbrida é realmente compatível com diversas configurações (diferentes plataformas e dispositivos)?

As aplicações selecionadas para teste foram desenvolvidas

com o *framework* de desenvolvimento de aplicações móveis híbridas e multiplataformas Cordova. Antes do processo de compilação, os *plug-ins* utilizados em cada aplicação foram instalados via NPM¹⁶. Diversas outras aplicações foram selecionadas, mas a instalação de alguns *plug-ins* depreciados, ou a incompatibilidade com a versão atual do ambiente de desenvolvimento da plataforma impediram sua compilação.

As aplicações foram compiladas pelo SDK do Android e iOS, instaladas e executadas nos dispositivos, observando a capacidade do *framework* de desenvolvimento de aplicações híbridas em gerar código compatível de compilação, instalação e execução nas plataformas testadas. Em inspeção manual, os pesquisadores identificaram que o *framework* foi capaz, como esperado de gerar aplicações compatíveis aplicações para as plataformas selecionadas. O tempo total dispendido em cada plataforma para completar os três processos apresentou diferenças pequenas. A Tabela VII mostra o resultado da investigação.

Para a plataforma iOS foi necessário assinar as aplicações utilizando um certificado credenciado pela Apple, fornecido para desenvolvedores homologados, tornando a geração do pacote de implantação mais complexo e demorado nesta plataforma.

Tabela VII: Resultado do processo de investigação da compilação, instalação, e execução das aplicações selecionadas.

Aplicação	Plug-ins	Compilação	Instalação	Execução	Plataforma	Tempo gasto (minutos)
Fresh Food F.	1	✓	✓	✓	Android	~8
					iOS	~12
PedidoApp	6	✓	✓	✓	Android	~7
					iOS	~9
MemesPlay	19	✓	✓	✓	Android	~10
					iOS	~14

QP2 - Uma aplicação móvel híbrida é executada, do ponto de vista funcional, corretamente (se comporta da mesma forma) em cada configuração?

A aplicação foi executada manualmente pelos pesquisadores, focando especificamente a funcionalidade sob teste e seguindo a sequência definida em seu respectivo ESG. As funcionalidades mantiveram o comportamento esperado em todas as plataformas durante o teste manual (Tabela VII). Durante a execução automatizadas dos testes, alguns eventos da aplicação PedidoApp demonstraram um comportamento não esperado, concluindo o teste parcialmente. Um exemplo é o evento "clique" vinculado ao elemento rotulado como botão "Confirmar Pedido". Este e outros eventos não dispararam. Em detalhes, foi identificado que a aplicação foi codificada com o *framework* desenvolvimento para Web AngularJS, que

¹⁶Node Package Manager (NPM) é um utilitário de linha de comando que interage com um repositório online de projetos de código aberto para o NodeJS, e ajuda na instalação de pacotes de software, gerenciamento de versão e gerenciamento de dependências de *plug-ins* (por exemplo) em projetos de software. Site: <https://www.npmjs.com>.

realiza um processo de vinculação de elementos HTML a seus eventos diferente de outros *frameworks* Javascript. Portanto, o *script* de teste não foi compatível com a maneira que o AngularJS vincula eventos aos elementos, indicando falta de suporte da ferramenta Selenium ou do *framework* Appium para tal *framework* Javascript. O mesmo comportamento foi observado em todas as plataformas selecionadas no estudo (Tabela VIII).

Tabela VIII: Teste manual x Teste automatizado

Aplicação		Dispositivos				
		SPAnd 6_4I	SPAnd 5_5I	TBAnd 4_7I	SPIOS 7_3I	TBIOS 9_9I
Fresh Food F.	TM	✓	✓	✓	✓	✓
	TA	✗	✓	✓	✓	✓
PedidoApp	TM	✓	✓	✓	✓	✓
	TA	✗	✗	✗	✗	✗
MemesPlay	TM	✓	✓	✓	✓	✓
	TA	✗	✓	✓	✓	✓

TM: Teste Manual TA: Teste Automatizado

Na plataforma Android 6.0.1 instalada no dispositivo SPAnd6_4I, todas as aplicações não completaram os testes automatizados. O Selenium e o Appium utilizam posicionamento absoluto para simular a ação clicar ou tocar nos elementos de interface de usuário. Foi observado uma inconsistência na localização do eixo X e Y dos elementos. O valor de X e Y ultrapassam a área do elemento, assim a simulação de clique ou toque ocorria fora da área exata do elemento, portanto, não disparando a ação esperada (clique/toque). Tal situação demonstra incompatibilidade do Selenium ou do Appium com a última versão do sistema operacional Android, utilizada no dispositivo. O recorte representado na Figura 5 foi extraído de uma das aplicações sob teste e demonstra o clique ou toque além da área do elemento de entrada de texto (*input*).

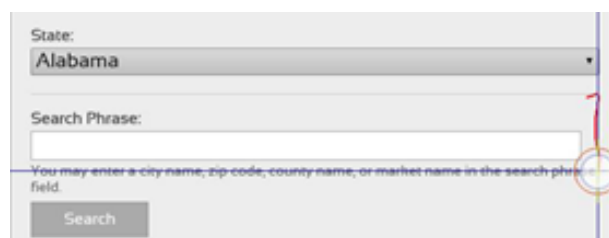


Figura 5: Clique ou toque fora da área do elemento.

Portanto, a execução manual das funcionalidades conforme planejado no caso de teste no ESG foi concluída com sucesso, não apresentando resultados negativos. Enquanto a execução manual apresentou situações que interromperam o teste automatizado, assim não testando a aplicação.

QP3 - Uma aplicação móvel híbrida apresenta um desempenho similar em cada configuração durante o teste automatizado?

Os testes automatizados foram executados sequencialmente por dez vezes. O *script* foi instrumentado para fornecer o tempo total de execução (somatório de tempo de cada execução). A Tabela IX apresenta o tempo dispendido na execução dos testes para cada aplicação que completaram o teste com sucesso.

O desempenho da execução foi diferente em cada dispositivo utilizado, fato justificado devido a configuração de hardware diferente. Outro fato importante a ser observado na análise de desempenho é o uso da Internet para recuperação dos dados utilizados nas aplicações:

- *Fresh Food Finder*: Realizada a busca de mercados de comida consultando fontes na Internet;
- *PedidoApp*: A relação de produtos vendidos também tem origem na Internet, neste caso um servidor da AmazonWS;
- *Memesplay*: Os vídeos disponíveis também estão hospedados remotamente.

Tabela IX: Desempenho da execução automatizada dos casos de testes nos dispositivos.

Aplicação	QE	Dispositivos x Tempo de execução (minutos)				
		SPAnd 6_4I	SPAnd 5_5I	TBAnd 4_7I	SPIOS 7_3I	TBIOS 9_9I
Fresh Food F.	10	EP	~10	~11	~16	~15
PedidoApp	10	EP	EP	EP	EP	EP
MemesPlay	10	EP	~10	~9	~21	~19

QE: Quantidade de execuções EP: Executou parcialmente (falhou)

QP4 - Uma aplicação móvel híbrida se apresenta para usuário da mesma forma (GUI) em cada configuração?

Para responder a está questão, o *screenshots* da primeira tela de cada aplicação foi analisado com o método descrito na Seção 3. Os *screenshots* gerados pelo dispositivo SPAnd6_4I foram utilizados como matrizes para comparações com as demais capturas dos outros dispositivos. Os resultados da análise estão apresentados na Tabela X, na qual mostra o percentual de diferença entre o *screenshot* matriz e os demais. As telas com percentual de diferença superior a 20% foram manualmente inspecionadas pelo pesquisador e nenhuma anomalia foi encontrada. O valor acima do limite estipulado se justificativa de três maneiras:

a) *Tamanho da tela do dispositivo*. Um dispositivo com tela maior apresentou uma área (espaço) de conteúdo maior, portanto, os elementos visuais da interface da aplicação são distribuídos nesta área maior. Por exemplo, uma lista de produtos para compra tem mais itens visíveis no dispositivo de tela maior (*tablet*) ao contrário de um dispositivo de tela menor (um *smartphone*).

b) *Conteúdo dinâmico*. As aplicações sob teste possuem conteúdo dinâmico, a cada execução um novo conteúdo substitui o anterior. Tal fato ocorreu na aplicação *PedidoApp*. A cada execução a relação de itens disponíveis para compra era diferente. Na aplicação *MemesPlay*, os pesquisadores apagaram o conteúdo, restando apenas as áreas estáticas.

c) *Elementos característicos do sistema operacional móvel*. Alguns elementos visuais do sistema operacional móvel contribuem na formação do percentual de diferença. É o caso da barra visível da área superior do sistema operacional Android, enquanto no sistema operacional iOS a barra fica sobreposta à aplicação.

Tabela X: Comparações da interface de usuário entre plataformas.

Aplicação	Dispositivos			
	SPAnd5_5I	TBAnd4_7I	SPIOS7_3I	TBIOS9_9I
Fresh Food F.	0,10	18,44	20,45	35,01
PedidoApp	4,72	35,89	38,60	36,18
MemesPlay	3,15	20,83	16,42	23,20

Portanto, os resultados da comparação das interfaces indicaram similaridade, mas com alguns graus de diferença, mas que não comprometeu o objetivo e a usabilidade da aplicação. A padronização defendida pelo W3C no uso de HTML e CSS garantem a compatibilidade da interface de usuário as aplicações móveis híbridas. Tais interfaces são construída com HTML e CSS, executados internamente numa área da aplicação conhecida como de *WebView*.

QP5 - A escrita (codificação) e execução do *script* de teste é compatível em cada configuração?

As interfaces de usuário são representadas por estruturas XML que mapeiam cada elemento de tela. Estes elementos são acessados programaticamente por meio de expressões de consulta XPath. Para cada aplicação foi codificado um *script* de teste em linguagem C#. Foi priorizado ao máximo a compatibilidade de estrutura lógica e computacional do código-fonte do *script* entre os diversos dispositivos utilizados no teste.

Foi identificado que a organização em estrutura de árvore dos elementos da interface de usuário da aplicação é diferente para o ambiente Android e iOS, e entre as versões do mesmo sistema operacional. Desta forma, foi necessário definir um critério de seleção de elementos em XPath para cada plataforma, além de pequenos ajustes do código-fonte do *script* para compatibilização entre as plataformas. Para comprovar tal diferença, a Tabela XI apresenta o somatório da quantidade de nós XML das interfaces de usuário das funcionalidades sob teste, evidenciando a diferenças na representação da interface de usuário nas diversas plataformas e dispositivos existentes. Assim, justificando a necessidade da compatibilização do *script* durante o processo de seleção do elemento de interface. A diferença a nível de estrutura não afetou a interface de

usuário, conforme respondido na questão de pesquisa anterior (QP4).

Quanto a execução dos *scripts*, salvo as adaptações para compatibilização entre as plataformas, igualmente mantiveram compatibilidade de execução nas distintas plataformas. Os testes automatizados para a aplicação PedidoApp e para as demais instaladas no dispositivo SPAnd6_4I foram executados parcialmente, devido a inconsistências relatadas nos resultados da QP2.

Tabela XI: Comparação de estruturas XML de interface das telas aplicações selecionadas.

Aplicação	Dispositivos x Quantidade de nós XML				
	SPAnd 6_4I	SPAnd 5_5I	TBAnd 4_7I	SPIOS 7_3I	TBIOS 9_9I
Fresh Food Finder	217	284	228	259	315
PedidoApp	125	138	108	102	235
MemesPlay	538	395	572	417	501

A. Ameaças a validade

Esta seção apresenta de forma sucinta possíveis ameaças a validade da presente pesquisa. O primeiro ponto a ser considerado é a quantidade de pesquisadores envolvidos, apenas dois, e apenas um pesquisador codificou e executou os testes.

As aplicações híbridas executam código HTML, CSS e Javascript, e fazem acesso a API do sistema operacional móvel. A pesquisa não considerou a realização testes em funcionalidades que fazem acesso aos recursos nativos do sistema operacional.

A quantidade de dispositivos e plataformas utilizadas para teste. O número de dispositivos foi reduzido, considerando apenas as últimas versões dos sistemas operacionais móveis.

A seleção de aplicações móveis construídas apenas com um *framework* de desenvolvimento de aplicações híbridas, podem comprometer a validade. O número reduzido de aplicações selecionadas para os testes, apenas três, também podem comprometer a validade.

V. TRABALHOS RELACIONADOS

Malavolta et al. [9] selecionaram mais de 11 mil aplicações na loja de distribuição da plataforma Android. O critério de seleção abordou as 500 primeiras aplicações mais baixadas em 25 categorias. Os autores identificaram que: (i) 445 aplicações móveis híbridas (3,73%) dentre as mais de 11 mil aplicações selecionadas para a pesquisa, (ii) observou que categorias como fotografia, música e áudio, jogos e personalização têm em menor número aplicações híbridas, provavelmente pela forte necessidade de alto-desempenho, e (iii) revelou o Apache Cordova como o *framework* mais utilizado na construção de aplicações híbridas, seguido do Appcelerator Titanium.

Boushehrinejadmoradi et al. [1] propuseram uma abordagem para testar aplicações móveis desenvolvidas por *frameworks* nativos multiplataformas, em específico o Xamarin. A ideia foi descobrir inconsistências no processo de tradução do código-fonte original para outras plataformas, comparando a execução da mesma aplicação em diferentes plataformas. A

abordagem deu origem a uma ferramenta chamada X-Checker, que executou 22.465 casos de teste construídos para uma determinada aplicação em diversas plataformas. Os resultados indicaram 47 inconsistências relacionadas tais como: (i) um método retornou valores diferentes entre as plataformas, (ii) uma exceção ocorreu em ambas plataformas, mas o tipo da exceção é diferente e (iii) a execução do teste retorna "sucesso" numa plataforma e na outra uma exceção.

Enquanto o estudo de Boushehrinejadmoradi et al. [1] objetivou encontrar inconsistência no código-fonte gerado pelo *framework* de desenvolvimento nativo multiplataformas, o estudo de Joorabchi et al. [17] escolheu uma abordagem para automatizar técnicas para detecção de inconsistências em aplicação iguais implementada para as plataformas iOS e Android. O resultado também foi positivo na identificação de inconsistências entre as aplicações.

Gao et al. [4] traçam um comparativo entre aplicações móveis nativas e Web, destacando a necessidade de ambientes de teste e automação específicos para cada tipo de aplicação. Para aplicações nativas, o sistema operacional móvel constitui o ambiente de teste, enquanto nas aplicações Web o navegador Web é o ambiente de teste. Os autores levantam questões tais com a falta de padronização na infraestrutura de teste de aplicações móveis, linguagens de *script* e protocolos de conectividade entre ferramentas de teste e plataformas. É sugerido o uso de computação em nuvem para simular diversos dispositivos conectados, gerando uma solução de controle e execução de teste automatizada que suporta simultâneos testes e em larga escala.

Por fim, Zein et al. [8] conduziram um mapeamento sistemático com propósito de identificar pesquisas existentes para técnicas de teste em aplicações móveis. Foram incluídas, avaliadas e discutidas 79 pesquisas. As técnicas de testes identificadas estão focadas em aplicações móveis nativas. Dessa forma, as pesquisas incluídas demonstram a carência em estudos para técnicas de testes automatizados em aplicações móveis híbridas e multiplataformas.

VI. CONCLUSÃO

Este estudo apresentou uma avaliação sobre teste automatizado de interface gráfica de usuário (GUI) em aplicações móveis híbridas. Para responder as questões de pesquisa, três aplicações móveis foram selecionadas e instaladas em cinco dispositivos móveis das plataformas Android e iOS com diferentes tipos de hardware.

As inspeções manuais indicaram que a compilação, instalação e execução manual das aplicações móveis híbridas para as plataformas Android e iOS são iguais. Portanto, o *framework* de desenvolvimento de aplicações móveis híbridas foi capaz, como esperado de gerar aplicações compatíveis para diferentes plataformas. Os testes manuais indicaram o funcionamento correto das aplicações nas diversas plataformas e dispositivos. Enquanto a execução automatizada dos testes indicou diferenças entre as plataformas. Alguns eventos não foram disparados pelo *script* de teste, demonstrando falta de suporte do Selenium ou do Appium. O mesmo comportamento

foi observado em todas as plataformas, e parece estar relacionado ao uso de *frameworks* Javascript, como o AngularJS. A execução automatizada foi executada sequencialmente por dez vezes, demonstrando desempenho similar entre os dispositivos utilizados no cenário de teste.

A compatibilização do mesmo *script* de teste entre as diversas plataformas foi uma tarefa complexa devido a diferença na organização estrutural da interface de usuário. Em alguns casos o *script* não foi capaz de testar partes das funções sob teste, ou por não encontrar o elemento, ou por não encontrar o posicionamento (eixos X e Y) correto do elemento dentro da interface para causar o disparo do evento.

Na análise comparativa das interfaces de usuário, foi utilizado comparações de capturas de telas (*screenshots*). A primeira tela foi capturada nos diversos dispositivos e posteriormente aplicado um algoritmo diferencial. Os resultados indicaram similaridade, mas com alguns graus de diferença. Na avaliação geral, a diferença encontrada não comprometeu o objetivo e a usabilidade da aplicação, sendo causada principalmente por características visuais do sistema operacional móvel.

Em trabalhos futuros, o foco do estudo será na realização de testes em recursos nativos (GPS, câmeras, agenda de contatos, etc) das aplicações móveis híbridas, com o objetivo de avaliar a capacidade do *framework* de desenvolvimento gerar aplicações móveis multiplataformas para diversos dispositivos e plataformas. Um segundo trabalho pretendido é avaliar testes automatizados em aplicações móveis híbridas que utilizam bibliotecas ou *frameworks front-end*, tais como jQuery, AngularJs e Reactjs, objetivando identificar a capacidade da biblioteca/*framework* em ser testável. Um terceiro trabalho pretendido trata da comparação de técnicas e ferramentas para testes automatizados em aplicações móveis híbridas e nativas, identificando semelhanças nos métodos. E por fim, confrontar o comportamento dos testes automatizados no emulador da plataforma móvel e no dispositivo real, objetivando identificar inconsistências.

REFERÊNCIAS

- [1] N. Boushehrinejadmoradi, V. Ganapathy, S. Nagarakatte, and L. Iftode. Testing cross-platform mobile app development frameworks (t). In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 441–451, Nov 2015.
- [2] IBM. Native, web or hybrid mobile-app development, 2012.
- [3] 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 06-03-2016.
- [4] J Gao, Xiaoying Bai, Wei-Tek Tsai, and T Uehara. Mobile Application Testing: A Tutorial. *Computer (Long Beach, Calif.)*, 47(2):46–55, 2014.
- [5] D Amalfitano, A R Fasolino, and P Tramontana. A GUI Crawling-Based Technique for Android Mobile Application Testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference*, pages 252–261, 2011.
- [6] Tor-Morten Gronli and Gheorghita Ghinea. Meeting Quality Standards for Mobile Application Development in Businesses: A Framework for Cross-Platform Testing. In *2016 49th Hawaii International Conference on System Sciences*, pages 5711–5720. IEEE, jan 2016.
- [7] Rafael Alves Paes de Oliveira. Apoio à automatização de oráculos de teste para programas com interfaces gráficas. Master’s thesis, USP - São Carlos, 2012.
- [8] Samer Zein, Norsaremah Salleh, and John Grundy. A Systematic Mapping Study of Mobile Application Testing Techniques. *Journal of Systems and Software*, apr 2016.
- [9] I Malavolta, S Ruberto, T Soru, and V Terragni. Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation. In *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM Int. Conf.*, pages 56–59, 2015.
- [10] W3C. W3C, 2016. Accessed on 2016-03-17.
- [11] Escolha Tecnologia. MemesPlay. <http://www.escolhatecnologia.com.br/>, 2016.
- [12] Sérgio Lopes. *Aplicações mobiles híbridas com Cordova e PhoneGap*. Casa do Código, 2016.
- [13] Microsoft. Unit Testing Framework. [https://msdn.microsoft.com/en-us/library/ms243147\(vs.80\).aspx](https://msdn.microsoft.com/en-us/library/ms243147(vs.80).aspx), 2016.
- [14] James Cryer. Resemble.js. <https://github.com/Huddle/Resemble.js/tree/master>, 2016. Accessed on 03-04-2016.
- [15] André Takeshi Endo. *Model based testing of service oriented applications*. PhD thesis, USP, 2013.
- [16] André Menegassi. Script de Teste Automatizado da Aplicação Fresh Food Finder. <https://github.com/andremenegassi/ScriptTestingPaper2016/blob/master/UnitTestProject>, 2016. Accessed on 11-05-2016.
- [17] M E Joorabchi, M Ali, and A Mesbah. Detecting inconsistencies in multi-platform mobile apps. In *Software Reliability Engineering (ISSRE), 2015 IEEE 26th Int. Symp.*, pages 450–460, nov 2015.