UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ANDRÉ AUGUSTO MENEGASSI

TESTES AUTOMATIZADOS PARA APLICAÇÕES MÓVEIS MULTIPLATAFORMA

DISSERTAÇÃO DE MESTRADO

CORNÉLIO PROCÓPIO

2018

ANDRÉ AUGUSTO MENEGASSI

TESTES AUTOMATIZADOS PARA APLICAÇÕES MÓVEIS MULTIPLATAFORMA

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Tecnológica Federal do Paraná - UTFPR como requisito parcial para a obtenção do título de "Mestre em Informática".

Orientador: Prof. Dr. André Takeshi Endo

CORNÉLIO PROCÓPIO

2018

Dados Internacionais de Catalogação na Publicação

M541 Menegassi, André Augusto

Testes automatizados para aplicações móveis multiplataforma / André Augusto Menegassi. – 2018.
82 f. : il. color. ; 31 cm.

Orientador: André Takeshi Endo.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Informática. Cornélio Procópio, 2018.
Bibliografia: p. 74-82.

1. Computação móvel. 2. Interfaces de usuário. 3. Software de aplicação. 4. Informática – Dissertações. I. Endo, André Takeshi, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD (22. ed.) 004

Biblioteca da UTFPR - Câmpus Cornélio Procópio
Bibliotecários/Documentalistas responsáveis:
Simone Fidêncio de Oliveira Guerra – CRB-9/1276
Romeu Righetti de Araujo – CRB-9/1676

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Cornélio Procópio
Programa de Pós-Graduação em Informática

Título da Dissertação Nº 44:

"TESTES AUTOMATIZADOS PARA APLICAÇÕES MÓVEIS MULTIPLATAFORMA".

por

André Augusto Menegassi

Orientador: **Prof. Dr. André Takeshi Endo**

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM INFORMÁTICA – Área de Concentração: Computação Aplicada, pelo Programa de Pós-Graduação em Informática – PPGI – da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Cornélio Procópio, às 14h do dia 07 de março de 2018. O trabalho foi _____ pela Banca Examinadora, composta pelos professores:

_____
Prof. Dr. André Takeshi Endo
(Presidente – UTFPR-CP)

_____
Prof. Dr. Bruno Barbieri de Pontes Cafeo
(UFMS)

_____
Profa. Dra. Katia Romero Felizardo Scannavino
(UTFPR-CP)

Visto da coordenação: _____
Danilo Sipoli Sanches
Coordenador do Programa de Pós-Graduação em Informática
UTFPR Câmpus Cornélio Procópio

A Folha de Aprovação assinada encontra-se na Coordenação do Programa.

AGRADECIMENTOS

A presente dissertação de mestrado não poderia chegar ao seu final sem o precioso apoio de várias pessoas.

Em primeiro lugar, não posso deixar de agradecer ao meu orientador, Professor André Takeshi Endo, por toda a paciência, empenho e sentido prático com que sempre me orientou.

Agradeço a Universidade do Oeste Paulista, em específico ao setor de Tecnologia da Informação pelo ambiente criativo e amigável que proporcionou apoio a esta pesquisa.

É claro, desejo igualmente agradecer aos meus amigos PNC's Alex, Eduardo, Marcão, Glauco e Isique.

Por último, quero agradecer à minha família pelo apoio incondicional que me deram, especialmente a minha esposa por sua compreensão.

RESUMO

MENEGASSI, André Augusto. TESTES AUTOMATIZADOS PARA APLICAÇÕES MÓVEIS MULTIPLATAFORMA. 81 f. Dissertação de Mestrado – Programa de Pós-graduação em Informática, Universidade Tecnológica Federal do Paraná. CORNÉLIO PROCÓPIO, 2018.

Contexto: Sistemas operacionais (SOs), como Android e iOS, controlam modernos dispositivos móveis e servem como plataforma para execução de uma ampla variedade de aplicações móveis. Essas aplicações podem ser multiplataforma, se destacando por sua capacidade de execução em vários SOs. Tais aplicações são desenvolvidas usando *frameworks*, como Apache Cordova, Xamarin e React Native. O teste desse tipo de aplicação é um desafio devido a diversidade de dispositivos e plataformas disponíveis no mercado. Como testar a aplicação em um único dispositivo ou plataforma não garante sua operação correta em outros, cada dispositivo representa uma configuração que precisa ser verificada. Entretanto, os mecanismos para automatizar o teste de interface de usuário (UI) não são multiplataforma e não oferecem suporte a várias configurações. Por exemplo, *scripts* de testes têm que ser escritos duas ou mais vezes dado que as representações da UI das plataformas são diferentes. **Objetivo:** esta dissertação tem como objetivo apresentar uma abordagem para gerar um *script* adequado para automatização do teste de UI em aplicações móveis multiplataforma. **Método:** a fim de ajustar os testes para executar em várias configurações, a abordagem adota dois dispositivos de referência: um executando o Android e outro o iOS. Como ambas plataformas têm sua própria representação de UI em XML, também investigou-se oito tipos de estratégias para localização de elementos de UI: seis expressões individuais e duas estratégias combinadas. Uma ferramenta chamada `x-PATeSCO` foi implementada para apoiar a abordagem proposta. Ela é preparada para gerar as oito estratégias de localização consideradas e incluí-las em um projeto de teste para execução em diversas configurações. A abordagem e a ferramenta foram avaliadas em um estudo experimental que utilizou nove aplicações móveis multiplataforma, comparando as estratégias de localização em seis dispositivos reais. **Conclusão:** os dados coletados na avaliação experimental mediram a taxa de aplicabilidade e executabilidade das estratégias de localização, e demonstraram que a abordagem e a ferramenta propostas contribuem para o teste de UI das aplicações em diversas plataformas. No geral, as estratégias combinadas (*ExpressionsInOrder* e *ExpressionsMultiLocator*) obtiveram os melhores resultados, alcançando 70,2% da executabilidade dos eventos e 65,5% de casos de teste executados com sucesso. Quando comparadas com expressões conhecidas e empregadas na prática, como *IdentifyAttributes* e *AbsolutePath*, as estratégias combinadas superam em executabilidade de eventos para 8,9% e 17,6%, respectivamente. No quesito tempo de execução, *ExpressionsInOrder* foi aproximadamente seis vezes mais rápida à *ExpressionsMultiLocator*.

Palavras-chave: Computação móvel, Aplicações móveis, Multiplataforma, Teste de software, Interface de usuário.

ABSTRACT

MENEGASSI, André Augusto. AUTOMATED TESTS FOR CROSS-PLATFORM MOBILE APPS. 81 f. Dissertação de Mestrado – Programa de Pós-graduação em Informática, Universidade Tecnológica Federal do Paraná. CORNÉLIO PROCÓPIO, 2018.

Background: Operating Systems (OSs), such Android and iOS, control modern mobile devices and are used as platform for executing a wide range of mobile apps. These apps can be cross-platform, highlighting for their ability to execute in multiple OSs. Such apps are developed using frameworks such as Apache Cordova, Xamarin and React Native. Testing this kind of apps is a challenge due to the diversity of devices and platforms available on the market. As testing the app on a single device or platform does not guarantee the correct operation on others, each device represents a configuration to be verified. However, mechanisms for automating the user interface (UI) tests are not cross-platform and do not support multiple configurations. For example, test scripts have to be written two or more times since the UI representations of each platforms are different. **Objective:** this master thesis aims to present an approach to generate a proper script to automate UI testing in cross-platform mobile apps. **Method:** to test in multiple configurations, the approach adopts two reference devices: one running Android and another iOS. Since both platforms have their own UI representation in XML, we also investigated eight strategies for locating UI elements: six individual expressions and two combined strategies. A tool called `x-PATeSCO` has been implemented to support the proposed approach. It is capable of generating the eight locating strategies and include them in a test project to execute in various configurations. The approach and the tool were evaluated in an experimental study with nine mobile cross-platform apps, comparing the locating strategies in six real devices. **Conclusion:** we measured the applicability and executability rates of locating strategies; the results gave evidence that the proposed approach and tool contribute to the UI testing of apps on various platforms. Overall, the combined strategies (*ExpressionsInOrder* and *ExpressionsMultiLocator*) obtained the best results, reaching 70.2% of the event executability and 65.5% of test cases executed successfully. When compared with known expressions employed in practice, like *IdentifyAttributes* and *AbsolutePath*, the combined strategies outperform in event executability from 8.9% to 17.6%, respectively. As for execution time, *ExpressionsInOrder* was approximately six times faster than *ExpressionsMultiLocator*.

Keywords: Mobile computing, Mobile apps, Cross-platform, Software testing, User interface.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

SO	Sistema Operacional
SDK	Software Developer Kit
API	Application Programming Interface
UI	Interface de Usuário (em inglês, *User Interface*)
ES	Engenharia de Software
PWA	Progressive Web Apps
ASF	Apache Software Foundation
DOM	Document Object Model
JSON	Javascript Notation Object
CLR	Common Language Runtime
TBM	Teste Baseado em Modelo
ESG	Event Sequence Graph
FSM	Finite State Machine
UML	Unified Modeling Language
DLL	Dynamic Link Library
AUT	App Under Test
LOC	Número de Linhas de Código
QPs	Questões de Pesquisa

SUMÁRIO

1 INTRODUÇÃO

Atualmente os dispositivos móveis fazem parte do dia-a-dia das pessoas e estão disponíveis em vários formatos como *smartphones*, *tablets* e *wearables*[1]. Os mesmos são equipados com poderosos processadores, ampla capacidade de armazenamento de dados e diversos sensores (BOUSHEHRINEJADMORADI et al., 2015). Modernos sistemas operacionais (SOs) controlam o hardware desses dispositivos. Na pesquisa apresentada pela *International Data Corporation* (IDC, 2017) sobre o *Market Share* no uso dos SOs móveis, Android[2] e Apple iOS[3] foram as plataformas mais consumidas no primeiro trimestre de 2017, com 85% e 14,7%, respectivamente. Em outra pesquisa, a GARTNER (2017) apresentou as vendas de *smartphones* no primeiro trimestre de 2017; a plataforma Android (86,1%) foi a líder de mercado, seguida pela plataforma iOS da Apple (13,7%).

Esses modernos SOs servem como plataforma para execução de uma ampla variedade de software denominado aplicações móveis. Normalmente elas estão disponíveis para download em lojas de distribuição mantidas pelas empresas proprietárias das plataformas. Segundo o site STATISTA (2017c), a plataforma Android possui o maior número de aplicações disponíveis para seus usuários 2,8 milhões e a Apple iOS 2,2 milhões. O desenvolvimento dessas aplicações pode ser classificado em três grupos: aplicações nativas, aplicações baseadas no navegador Web e aplicações híbridas (LATIF et al., 2016; IBM, 2012). As aplicações nativas são desenvolvidas utilizando o *Software Developer Kit* (SDK) do SO Móvel, o que possibilita acesso a sua *Application Programming Interface* (API), além de acesso aos recursos do dispositivo. As aplicações Web são desenvolvidas com tecnologias empregadas na construção de software para Web como HTML5 (W3C, 2017b), CSS3 (W3C, 2017b) e Javascript (W3C, 2017b). Elas são armazenadas em um servidor Web, acessadas e executadas sob o navegador *(Browser)* e não têm acesso a recursos avançados do dispositivo e do SO. E finalmente, as aplicações híbridas combinam tecnologias comuns das aplicações Web (HTML5, CSS3 e Javascript) com suporte direto às APIs nativas do SO, assim como as aplicações nativas. Sua principal característica é

[1] Dispositivos eletrônicos vestíveis como óculos e relógios.
[2] https://www.android.com
[3] http://www.apple.com

desenvolver uma vez e implantá-la em diversos dispositivos móveis com diferentes SOs.

No atual cenário, diversos SOs móveis estão disponíveis, cada qual com seu ambiente de desenvolvimento de aplicações nativas. Uma aplicação nativa não é capaz de ser executada diretamente em um outro SO. Essa fragmentação dos SOs é vista como uma barreira para o desenvolvimento de aplicações móveis, uma vez que diminui o possível número de usuários, mantendo-os em plataformas específicas. Além disso, o desenvolvimento nativo para múltiplas plataformas envolve a criação de diversos times de desenvolvedores específicos para cada plataforma móvel, ambientes de desenvolvimento diferentes, gerando redundância e aumentando os custos envolvidos no projeto (MALAVOLTA et al., 2015a; XANTHOPOULOS; XINOGALOS, 2013). BOUSHEHRINEJADMORADI et al. (2015) e JOORABCHI et al. (2015) sugerem que para atrair mais usuários, disponibilizar a mesma aplicação móvel em diferentes plataformas tornou-se uma prática comum da indústria e, portanto, pode gerar mais receita por meio da sua comercialização e venda de anúncios comerciais. Nesse sentido, alguns *frameworks* de desenvolvimento oferecem um processo único de construção de aplicações móveis. O produto final é uma aplicação multiplataforma capaz de executar em vários SOs.

Alguns requisitos de software são específicos ou ganham maior importância no processo de desenvolvimento das aplicações móveis (WASSERMAN, 2010), tais como: (i) capacidade de integração entre as aplicações, (ii) manipulação de sensores, (iii) uso de serviços externos por meio da redes de computadores ou Internet, (iv) suporte a múltiplas e diferentes plataformas de hardware, (v) segurança envolvendo acesso não permitido aos dados, (vi) a interface de usuário *(User Interface - UI)* deve atender às diretrizes da plataforma e (vii) otimização do consumo de energia. Especificamente para as aplicações multiplataforma, a compatibilidade de sua execução em diversas plataformas é um requisito relevante que deve ser considerado. Idealmente, esse tipo de aplicação deve fornecer a mesma funcionalidade e comportamento nas diferentes plataformas suportadas.

No desenvolvimento de aplicações móveis é essencial a prática da Engenharia de Software (ES) para assegurar a qualidade do produto final, como por exemplo a atividade de teste. MYERS (2004) define teste de software como um processo de execução de um programa com a intenção de encontrar defeitos. Se um defeito foi encontrado após a disponibilização da aplicação na loja, o usuário poderá classificá-la negativamente. Após a correção, uma nova submissão deverá ser realizada para aprovação, o que pode demorar alguns dias, retardando o acesso dos usuários à nova versão. Além disso, não é possível obrigar o usuário a realizar a atualização (GRONLI; GHINEA, 2016). Nesse quadro, teste de aplicações móveis vem ganhando importância na área da ES; tal fato pode ser observado pela quantidade de pesquisas

exploradas nesse contexto. ZEIN et al. (2016) e HOLL; ELBERZHAGER (2016) conduziram pesquisas abrangentes e em profundidade no contexto de teste de aplicações móveis, na qual exploram diversos estudos que evidenciam o interesse de pesquisadores no assunto.

Em comparação com o teste manual, o emprego do teste automatizado é uma boa prática para diminuição dos custos envolvidos nas atividades de teste, pois as estratégias tradicionais de teste costumam ser custosas e exigir esforços consideráveis dos testadores (OLIVEIRA, 2012). Neste trabalho o teste automatizado funcional é explorado a nível de UI na elaboração de casos de teste. Esse teste é um tipo de teste caixa-preta e tem por objetivo verificar as funcionalidades do software, observando se as entradas e saídas geradas estão de acordo com suas especificações (BARBOSA et al., 2004; MYERS, 2004).

1.1 MOTIVAÇÃO

Uma aplicação móvel pode ser distribuída para diferentes plataformas, sendo necessário garantir seu comportamento funcional independentemente da plataforma na qual é executada. Além disso, SOs móveis possuem diversas configurações que influenciam no comportamento da aplicação (GUDMUNDSSON et al., 2016). Essas aplicações são diferentes do software tradicional e requerem técnicas especializadas para teste (ZEIN et al., 2016; WASSERMAN, 2010). Os dispositivos móveis trazem consigo portabilidade e diversidade, características que geram algumas restrições, como: telas e autonomia de energia limitada, mecanismos de conectividade e capacidade de processamento reduzido (MUCCINI et al., 2012).

No cenário das aplicações multiplataforma, o uso de aplicações híbridas é considerado uma estratégia para evitar o problema da fragmentação da aplicação em diversas plataformas (MALAVOLTA et al., 2015b), visto que sua principal característica é a capacidade de execução em múltiplas plataformas. Elas são construídas com *frameworks* de desenvolvimento híbrido o qual utilizam tecnologias presentes no desenvolvimento de aplicação para Web. Uma outra estratégia para evitar tal problema é o uso de *frameworks* de desenvolvimento capazes de traduzir o código-fonte em linguagem específica da plataforma móvel, ou combinar o uso elementos de UI nativos, enquanto a lógica da aplicação é implementada com uma linguagem não-nativa. O produto final é uma aplicação nativa e multiplataforma ao mesmo tempo (XANTHOPOULOS; XINOGALOS, 2013; WILLOCX et al., 2016, 2015). Neste trabalho a aplicação gerada por esse segundo tipo de *framework* foi nomeada como aplicação nativa-multiplataforma. Além das duas estratégias citadas, as aplicações baseadas em navegador Web também são multiplataforma, pois têm como base de execução um navegador de Web,

ferramenta disponível em qualquer plataforma. No entanto, esse tipo de aplicação tem desempenho inferior comparado aos demais tipos e não acessam os recursos nativos do SO e do dispositivo (CORRAL et al., 2012). Devido a essas limitações não foram consideradas neste estudo; ademais o teste para aplicações Web também podem ser executados nesse tipo de aplicação móvel.

O teste de aplicações móveis multiplataforma é um desafio devido a variabilidade de dispositivos, configurações e SO disponíveis no mercado (GRONLI; GHINEA, 2016; BOUSHEHRINEJADMORADI et al., 2015; JOORABCHI et al., 2013). O termo múltiplas configurações pode ser definido como a variabilidade (características que variam em relação aos diversos produtos) de dispositivos móveis com configurações diferentes, tais como versão do SO, tamanho de tela, vários tipos de sensores e hardware. Como testar a aplicação em um único dispositivo não garante a operação correta em outros (NAGAPPAN; SHIHAB, 2016; JOORABCHI et al., 2015), cada dispositivo representa uma configuração que precisa ser verificada. Além dos defeitos comuns a qualquer software, inconsistências relacionadas a falta de compatibilidade da execução da mesma aplicação em plataformas diferentes podem surgir devido aos mecanismos utilizados pelos *frameworks* para garantir a execução multiplataforma da aplicação. Por exemplo, as aplicações híbridas utilizam mecanismos chamados de *WebViews* e *Plugins*, enquanto as nativas-multiplataforma utilizam-se da tradução do código original para código intermediário de plataforma e mapeamento de elementos de UI. Esses mecanismos possuem implementações diferentes para cada plataforma.

O teste automatizado é essencial para cobrir muitas configurações, mas atualmente os mecanismos de teste para aplicações móveis não são multiplataforma. Por exemplo, um teste automatizado de UI usando uma ferramenta como o Appium (APPIUM, 2017)[4] tem que ser escrito duas vezes dado que as representações XML da UI do Android e iOS são diferentes. Mesmo em uma mesma plataforma não há homogeneidade entre suas versões (FAZZINI et al., 2017; MENEGASSI; ENDO, 2016); isso implica que dois ou mais *scripts* de teste de UI podem ser necessários para diferentes plataformas e suas versões. Além disso, a manutenção do *script* é considerada uma tarefa de custo elevado (LEOTTA et al., 2014, 2015; LI et al., 2017) e aplicações multiplataforma agravam tal problema, podendo exigir duas ou mais versões do mesmo *script* de teste. Os atuais *frameworks* e ferramentas de teste não fornecem o mesmo nível de suporte para diferentes plataformas (JOORABCHI et al., 2015) e as pesquisas sobre testes automatizados se concentraram em plataformas específicas, principalmente Android (ZEIN et al., 2016). Assim, há uma falta de abordagens para o teste automatizado de aplicações

[4]O Appium é um *framework open source* para automatizar teste em aplicações móveis nativas, Web ou híbridas nas plataformas iOS e Android. Será detalhado na Seção 2.3.2.

multiplataforma.

1.2 OBJETIVOS

O processo de desenvolvimento de aplicações móveis requer técnicas de teste especializadas que promovam a qualidade do produto final. No entanto, no cenário das aplicações multiplataforma a garantia da qualidade adicionalmente envolve o teste em diversos dispositivos móveis, variando o SO, tamanho de tela, sensores, etc. Cada configuração exige mecanismos de testes apropriados e específicos, e que não são compartilhados entre elas.

Esta dissertação tem como objetivo introduzir uma abordagem para gerar *scripts* de teste de UI para aplicações móveis multiplataforma em múltiplas configurações. A abordagem é baseada na execução da aplicação em um dispositivo de referência para cada SO, no caso, um Android e um iOS. Durante a execução da aplicação dados da UI são extraídos para definição de um modelo de teste funcional. Como ela se concentra em teste funcional em nível de sistema (caixa-preta), estratégias para localizar elementos da interface do usuário também são propostas e investigadas.

1.3 ORGANIZAÇÃO DO TEXTO

Esta dissertação de mestrado está organizada da seguinte forma: no Capítulo 2 são apresentados os principais conceitos sobre aplicações móveis e, especificamente, sobre aplicações híbridas e nativas-multiplataforma. Os principais *frameworks*, processo de construção e compilação dessas aplicações são evidenciados. Além disso, são abordados os principais conceitos de teste de software, principalmente no contexto das aplicações móveis, bem como seus desafios e particularidades. Os trabalhos relacionados também são apresentados.

O Capítulo 3 apresenta a abordagem proposta para o teste em aplicações multiplataforma e o embasamento referente a UI das aplicações utilizados em sua definição. Além disso, são apresentadas estratégias de localização de elementos de UI. A abordagem deu origem a uma ferramenta que também é apresentada.

No Capítulo 4, uma avaliação da abordagem, das estratégias de localização de elementos de UI e da ferramenta de suporte ao teste automatizado é apresentada.

Finalmente no Capítulo 5, as contribuições desta dissertação, limitações e trabalhos futuros são apresentados.

2 REVISÃO BIBLIOGRÁFICA

2.1 CONSIDERAÇÕES INICIAIS

Neste capítulo são apresentadas a fundamentação teórica e a contextualização sobre aplicações móveis e teste de software, necessárias para o entendimento do trabalho. A Seção 2.2 apresenta os principais conceitos sobre aplicações móveis contextualizando com a computação móvel e seu estado atual. São apresentadas definições sobre aplicações multiplataforma e um breve detalhamento sobre os seus *frameworks* de desenvolvimento. Na Seção 2.3 são apresentadas uma contextualização teórica sobre teste de software e uma breve introdução ao Teste Baseado em Modelo (TBM). Os desafios em testar aplicações móveis também são relacionados e algumas ferramentas específicas e suas características para o teste de aplicações móveis são identificadas. Ademais, os defeitos inerentes das aplicações móveis multiplataforma são discutidos. Por fim, a Seção 2.4 apresenta os trabalhos relacionados à esta pesquisa.

2.2 COMPUTAÇÃO MÓVEL

Segundo COULOURIS et al. (2012), a computação móvel surgiu como um paradigma no qual usuários poderiam carregar seus computadores pessoais e conectar-se com outros dispositivos por meio de linhas telefônicas e um *modem*. Os primeiros dispositivos "leves" surgiram por volta de 1980. A evolução tecnológica miniaturizou os dispositivos, cada um com várias formas de conectividade sem fio, incluindo redes de celulares, Wi-Fi e *Bluetooth*. Mark Weiser em sua publicação *The Computer for the 21st Century* (WEISER, 1991) cunhou o termo computação ubíqua, e faz referência à onipresença dos dispositivos móveis por meio da integração crescente da computação (visível ou invisível) incorporada ao ambiente humano (SILVA et al., 2015; KURKOVSKY, 2007). Normalmente esses dispositivos são gerenciados por SOs específicos capacitados a controlar seu hardware.

Um SO móvel pode ser definido como um sistema que deve ser portátil, utilizável em situações de mobilidade, em todo lugar, em qualquer lugar e a qualquer momento (PASTORE,

2013). Esses modernos SOs servem como plataforma para execução de uma ampla variedade de software denominados aplicações móveis. Portanto, é uma base sobre a qual as aplicações do usuário são executadas. De acordo com DEI; SEN (2015), um SO móvel fornece diversas interfaces de comunicação entre partes da aplicação e o hardware do dispositivo por meio de uma API. O SO executa e gerencia tarefas básicas, tais como o reconhecimento de entradas *(inputs)* para dispositivo e geração de saída *(output)* para o visor, e tarefas mais complexas como o acesso aos sensores, além de gerenciar a execução ao mesmo tempo de diferentes aplicações garantindo o seu isolamento em nível de memória.

Dentre os SOs móveis disponíveis no mercado, pesquisas (IDC, 2017; GARTNER, 2017; STATISTA, 2017c, 2016; OKEDIRAN et al., 2014) destacam que nos últimos anos a plataforma Android e iOS foram as mais utilizadas pelos usuários. Além disso possuem o maior número de aplicações em suas lojas de distribuição.

O Android é um SO móvel livre e de código aberto *(open source)* e está disponível para diversos tipos de dispositivos, como *smartphones*, *tablets*, TV, relógios, óculos e automóveis (DEITEL et al., 2016; LECHETA, 2015). Ele é baseado no núcleo do Linux, no qual é responsável por gerenciar a memória, os processos, *threads*, segurança dos arquivos e pastas, além de redes e *drivers*. O Android é o resultado de uma aliança chamada *Open Handset Alliance* (OHA), um grupo formado por gigantes do mercado de telefonia de celulares liderados pela Google. O objetivo do grupo é definir uma plataforma única e aberta para celulares, moderna e flexível para o desenvolvimento de aplicações corporativas (LECHETA, 2015; ABLESON et al., 2012). As aplicações Android são desenvolvidas em linguagem Java e/ou Kotlin[1]. Após compilação, o código é convertido para o formato *Dalvik Executable (.dex)*[2], o qual representa a aplicação compilada. Os arquivos *.dex* e outros recursos como imagens, são compactados em um único arquivo com a extensão *.apk (Android Package File)*, que representa a aplicação final, pronta para ser distribuída e instalada (TANENBAUM; BOS, 2016; LECHETA, 2015). O principal ponto de distribuição das aplicações Android é a loja Google Play[3].

O iOS é o SO dos dispositivos móveis da Apple e sua primeira aparição foi na primeira versão do iPhone lançada em 2007. No ano seguinte, foi lançado o SDK oficial para desenvolvimento de aplicações juntamente com loja de distribuição de aplicações AppStore[4]

[1] https://kotlinlang.org/

[2] O Android não é uma plataforma de linguagem Java tradicional. O código Java da aplicação é fornecido no formato bytecode do Dalvik, que implementa o ambiente Java no Android, sendo responsável por executar aplicações. É uma espécie de *Java Virtual Machine* (JVM) customizada para dispositivos móveis (TANENBAUM; BOS, 2016).

[3] https://play.google.com/store

[4] https://www.appstore.com

(LECHETA, 2015; ALASDAIR, 2013). Além do iPhone, outros dispositivos da empresa também são controlados por esse SO, tais como iPod, iPad e a Apple TV. Sua execução é restrita ao hardware construído pela Apple; portanto, somente os dispositivos da própria autora são os que executam o iOS (MILANI, 2014). As linguagens de programação oficiais para o iOS são *Objective-C* e *Swift*, sendo essa última uma linguagem com sintaxe simples e moderna (LECHETA, 2016). O XCode é uma IDE desenvolvida pela Apple que permite o desenvolvimento de projetos para seus dispositivos móveis e não há nenhum custo no processo para obtê-lo e começar a criar aplicações, assumindo a existência de um computador com o SO da Apple (MAC OS) (LECHETA, 2015; MILANI, 2014). Para disponibilizar aplicações na AppStore, é necessário adquirir uma licença da Apple como desenvolvedor.

2.2.1 APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

PASTORE (2013) define as aplicações móveis como um tipo de software que executa tarefas específicas nos dispositivos móveis dos usuários. As aplicações mais famosas nasceram no contexto social, tais como as versões para dispositivos móveis do Facebook[5] ou Twitter[6]. Elas estão se tornando frequentes em diversas outras categorias (por exemplo, negócios e educação) devido ao uso de elementos gráficos e a facilidade de acesso com um simples toque na tela.

Vários requisitos exclusivos distinguem as aplicações móveis das convencionais. As aplicações móveis são oferecidas para um conjunto de dispositivos e plataformas diferentes (SOs diferentes), com características diversas, 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 (GAO et al., 2014). Os desenvolvedores devem considerar que os dispositivos móveis têm algumas limitações de processamento, memória e armazenamento em relação ao computador tradicional. Tais características não podem ser ignoradas, caso contrário a experiência do usuário ao interagir com a aplicação será negativa. Além disso, pode ser explorada a presença específica de sensores (por exemplo, GPS e câmera) no desenvolvimento das aplicações (PASTORE, 2013).

O desenvolvimento de aplicações móveis pode ser classificado em três grupos, aplicações nativas, aplicações baseadas no navegador Web e aplicações híbridas (LATIF et al., 2016; IBM, 2012).

As aplicações nativas possuem arquivos binários executáveis instalados e armazenados

[5]http://www.facebook.com
[6]http://www.twitter.com

diretamente no dispositivo. Ela é codificada em um SDK, normalmente fornecido pela organização proprietária do SO móvel. A aplicação tem acesso direto às funções do SO, sem qualquer recurso intermediário, sendo livre para acessar todas as APIs que são disponibilizadas (IBM, 2012). Uma das suas principais vantagens envolve seu desempenho superior comparado com os demais tipos, além de acessar diretamente o hardware do dispositivo (XANTHOPOULOS; XINOGALOS, 2013). A Tabela 1 relaciona as diferentes ferramentas, linguagens de programação e o formato de empacotamento das aplicações nativas.

Tabela 1: Sistemas Operacionais e detalhes de desenvolvimento (IBM, 2012; LATIF et al., 2016).

	iOS	Android
Linguagem	Objective-C, C, C++ e swift	Java e Kotlin (em alguns casos C e C++)
Ferramentas	XCode	Android SDK e Android Studio
Formato de empacotamento	.app	.apk

Uma desvantagem desse tipo de aplicação é a incompatibilidade com as diversas plataformas móveis existentes. Por exemplo, uma aplicação escrita para Apple iOS não é executável em outro SO, sendo necessária uma nova implementação em um outro ambiente de desenvolvimento e linguagem de programação. No entanto, alguns *frameworks* possibilitam o desenvolvimento de aplicações móveis nativas e multiplataforma usando linguagens como C# e Javascript (XAMARIN, 2017a; NATIVESCRIPT, 2017; REACTNATIVE, 2017; REACTXP, 2017b; WEEX, 2017). Tais *frameworks* utilizam elementos de UI nativos e combinados com mecanismos de execução de código intermediário originado a partir do código-fonte da aplicação (LATIF et al., 2016; WILLOCX et al., 2015, 2016; XANTHOPOULOS; XINOGALOS, 2013). Neste trabalho, o termo adotado é aplicações nativas-multiplataforma.

As aplicações baseadas no navegador Web são acessadas e executadas sob modernos navegadores Web, com suporte aos novos recursos do HTML5 (W3C, 2017b), CSS3 (W3C, 2017b) e Javascript (W3C, 2017b). A especificação do HTML5 inclui recursos de geolocalização, armazenamento local *off-line*, formatos de mídias (áudio e vídeo), além de avançados componentes de UI (IBM, 2012). Normalmente estão disponíveis em um servidor Web. Uma nova abordagem para esse tipo de aplicação é a utilização das *Progressive Web Apps* (PWA) que fornecem uma melhor experiência de usuário semelhante às aplicações nativas (DEVELOPERS, 2017).

As aplicações Web têm como desvantagem a ausência de acesso aos vários recursos oferecidos pelo SO móvel, limitando o tipo de aplicação que pode ser desenvolvido. Outra desvantagem é que essas aplicações não estão disponíveis nas lojas de distribuição, sendo

necessário a utilização de outros meios para disponibilizá-las. Como vantagem é possível citar a disponibilização da mesma aplicação para todas as plataformas móveis *(cross-platform)*, como também no clássico *Desktop*, devido às características de portabilidade oferecidas pelas tecnologias Web utilizadas e aliadas aos navegadores Web.

As aplicações híbridas combinam características das aplicações nativas e das aplicações baseadas no navegador Web. Essas aplicações são desenvolvidas usando os recursos comuns das aplicações Web como HTML5, CSS3 e Javascript, e com suporte direto às APIs nativas do SO 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 (LATIF et al., 2016; IBM, 2012). A aplicação é dividida em duas partes, uma nativa e outra conhecida como *WebView*, responsável por executar o HTML, CSS e Javascript (XANTHOPOULOS; XINOGALOS, 2013).

No contexto das aplicações móveis híbridas, são utilizados *frameworks* de desenvolvimento híbridos como PhoneGap[7]/Cordova[8] e Sencha Touch[9]. O desenvolvedor especifica a lógica da aplicação e UI usando componentes do desenvolvimento Web. No entanto, esses componentes não possuem suporte nativo a todos os recursos disponibilizados pelo dispositivo e seu SO, assim o *framework* provê suporte a esses recursos nativos (BOUSHEHRINEJADMORADI et al., 2015). A vantagem desse tipo de aplicação está no resultado final, uma aplicação que pode ser disponibilizada nas lojas de distribuição das várias plataformas móveis. A porção Web das aplicações híbridas pode ser uma página Web armazenada em um servidor ou arquivos HTML, CSS e Javascript empacotados e armazenados localmente dentro da aplicação. Uma desvantagem está no desempenho da aplicação. Estudos indicam um desempenho inferior comparado com aplicações nativas (CORRAL et al., 2012).

A Tabela 2 relaciona as principais características apresentadas sobre os tipos de aplicações móveis.

2.2.2 APLICAÇÕES MÓVEIS MULTIPLATAFORMA

Segundo BLOM et al. (2008), o *slogan* "escreva uma vez, execute em qualquer lugar" foi introduzido pela Sun Microsystems com a linguagem de programação Java, em específico para o *framework* Java Micro Edition (Java ME). A ideia é implementar a aplicação apenas uma vez e então executá-la em qualquer plataforma, independentemente do seu SO. Com o mesmo

[7] http://phonegap.com
[8] https://cordova.apache.org
[9] https://www.sencha.com

Tabela 2: Comparação entre as características dos tipos de aplicações móveis.

Características	Nativa	Web	Híbridas
Loja de distribuição	Sim	Não	Sim
Instalação	Sim	Não	Sim
Desempenho	Alto	*Inferior	*Inferior
Multiplataforma	Não	Sim	Sim
Acesso a API do dispositivo	Sim	Sim (limitado)	Sim (limitado)
**O estudo conduzido por CORRAL; SILLITTI; SUCCI (2012) indica um desempenho inferior comparado com aplicações nativas.*			

objetivo a Microsoft introduziu o .NET Compact Framework, uma versão compactada do seu .NET Framework, mas especificamente para dispositivos móveis. O uso de tecnologias como HTML, CSS e Javascript também é possível o desenvolvimento de aplicações "escreva uma vez, execute em qualquer lugar". Essas são aplicações baseadas no navegador Web, uma ferramenta comum em qualquer plataforma computacional.

No contexto das aplicações móveis, projetos híbridos e nativos-multiplataforma são opções para a construção de aplicações multiplataforma, além de favorecerem o desenvolvimento de um projeto único evitando redundâncias de código, diminuindo custo e tempo inerentes à construção e manutenção de projetos específicos para cada plataforma (LATIF et al., 2016). O termo multiplataforma está relacionado ao conceito de codificar uma vez e executar em várias plataformas. Essa característica é explorada por *frameworks* comerciais e *open source*. A seguir alguns *frameworks* para construção de aplicações híbridas e nativas-multiplataforma são brevemente apresentados.

a) Frameworks de desenvolvimento de Aplicações Híbridas

O **Cordova** (CORDOVA, 2016) é um *framework open source* para desenvolvimento móvel para Android, Blackberry, iOS, Windows Phone, Ubuntu, Windows (8.1/10) e OS X. O acesso aos recursos nativos do SO é baseado em *plugins* que são invocados por código Javascript. Posteriormente, o Cordova será descrito em detalhes na Seção 2.2.3.

O **PhoneGap** (PHONEGAP, 2016) é uma distribuição *open source* do projeto Cordova fornecida pela Adobe[10]. O processo de compilação da aplicação ocorre em nuvem (PhoneGapBuild[11]). A licença não-comercial possui algumas limitações referente a compilação do código-fonte em nuvem.

O **Sencha Touch** (SENCHA, 2017) é um *framework* comercial que fornece suporte para criar aplicações que executam no Android, iOS, BlackBerry, Windows Phone e Tizen.

[10]http://www.adobe.com/
[11]https://build.phonegap.com/

O Cordova é utilizado pelo *framework* para acesso nativo aos recursos do dispositivo. Uma coleção abrangente de componentes de UI são pré-integrados ao *framework*.

O **IONIC** (IONIC, 2017) é um *framework open source* focado principalmente na aparência e interação com a UI da aplicação. É capaz de gerar aplicações para Android, iOS e Windows 10 Universal App[12]. O acesso aos recursos nativos do dispositivo é realizado como o uso de *plugins* do Cordova/PhoneGap.

O **Intel XDK** (XDK, 2017) é um *framework* não-comercial fornecido pela Intel[13] e suporta o desenvolvimento de aplicações híbridas para Android, iOS e Windows 10 Universal App. O suporte nativo da aplicação aos recursos do dispositivo é fornecido pelo Cordova/PhoneGap. Um ambiente de desenvolvimento próprio dá suporte ao seu uso.

O **AppBuilder** (TELERIK, 2016) é uma plataforma da Telerik para desenvolvimento de aplicações híbridas para o Android, iOS e Windows Phone. O ambiente de desenvolvimento é baseado em uma plataforma de nuvem. O acesso nativo da aplicação aos recursos do dispositivo é realizado com *plugins* do Cordova/PhoneGap. Possui licença comercial.

Ao analisar todos os *frameworks* anteriormente relacionados, foi possível perceber que o Apache Cordova é a base de todos para realizar o acesso aos recursos nativo do dispositivo. Ele será apresentado em mais detalhes na Seção 2.2.3.

b) Frameworks de desenvolvimento de Aplicações nativas-multiplataforma

O **Xamarin** (XAMARIN, 2017a) é um conjunto de produtos mantidos pela Microsoft para desenvolvimento de aplicações móveis usando a linguagem de programação C#. Os elementos de UI são nativos da plataforma móvel. Ao final, uma aplicação nativa é gerada para cada plataforma. A Seção 2.2.4 apresentará mais detalhes sobre o Xamarin.

O **React Native** (REACTNATIVE, 2017) é um *framework* mantido pelo Facebook para construção de aplicações móveis usando Javascript e/ou JSX. Ele oferece diversos componentes de UI que são convertidos para os elementos de UI de cada plataforma móvel. Na Seção 2.2.5 esse *framework* será detalhado.

O **NativeScript** (NATIVESCRIPT, 2017) é um *framework* que utiliza Javascript, AngularJS e TypeScript para construção de aplicações móveis nativas e multiplataforma para iOS e Android, com acesso completo aos recursos do SO. A UI é construída com elementos nativos oferecidos pelo SO.

[12]Windows 10 Universal App é uma plataforma de desenvolvimento de aplicações destinadas a uma ampla variedade de dispositivos, incluindo móveis, área de trabalho, HoloLens, Surface Hub e Xbox (MICROSOFT, 2016b)

[13]https://software.intel.com/pt-br

O **ReactXP** (REACTXP, 2017b) é um *framework* mantido pela Microsoft para o desenvolvimento de aplicações móveis baseada no React Native. Além do iOS e Android, o *framework* suporta o desenvolvimento de aplicações Web e Windows, e expõe alguns componentes e APIs dos SOs que não são implementados no React Native.

O **Weex** (WEEX, 2017) é um *framework* para desenvolvimento de aplicações móveis mantido pela *Apache Software Foundation* (ASF). A aplicação é desenvolvida usando Javascript em conjunto com a biblioteca Vue.js[14] e oferece uma rica extensão de elemento de UI nativos, além de suporte à APIs do SO.

Ao analisar esses *frameworks* foram identificadas duas semelhanças em suas abordagens para construção de aplicações nativas-multiplataforma. Primeira, não há emulação de elementos de UI. A aplicação é renderizada utilizando os elementos de UI nativos da plataforma móvel; nesse caso, uma vantagem para o usuário que está adaptado ao estilo de UI da plataforma do seu dispositivo. A outra semelhança refere-se à implementação de um mecanismo de "ponte" (em inglês, *bridge*) capaz de invocar os métodos nativos do Objective-C/Swift (para iOS) ou Java (para Android). Atualmente o Xamarin e React Native dentre os *frameworks* nativo-multiplataforma têm o maior número de desenvolvedores e participação de mercado, fato observado ao consultar os repositórios de código-fonte disponíveis no GITHUB[15] [16], o site de perguntas e respostas Stackoverflow[17] e as tendências de busca no Google registradas pelo Google Trends[18]. Esses dois *frameworks* serão detalhados nas Seções 2.2.4 e 2.2.5.

2.2.3 APACHE CORDOVA

Atualmente o Apache Cordova (CORDOVA, 2016) é o *framework* mais comum para construção de aplicações multiplataforma (LOPES, 2016; STATISTA, 2017b). Criado pela empresa Nitobi em 2008, seu nome original era PhoneGap. Em 2011, a Adobe anunciou a aquisição da Nitobi e forneceu o projeto para a ASF, sendo rebatizado como Apache Cordova e disponibilizado como um projeto *open source*. A arquitetura de uma aplicação construída com o Cordova consiste nos seguintes componentes (WARGO, 2013):

- Um projeto nativo Figura (1-a) para cada SO suportado, contendo um componente nativo chamado *WebView* (Figura 1-b) usado para processar HTML, CSS e Javascript;

[14] https://vuejs.org
[15] https://github.com/search?utf8=%E2%9C%93&q=xamarin&type=Repositories
[16] https://github.com/search?utf8=%E2%9C%93&q=reactnative&type=Repositories
[17] https://insights.stackoverflow.com/survey/2017#technology-frameworks-libraries-and-other-technologies
[18] https://trends.google.com.br/trends/explore?cat=31&q=xamarin,react%20native

- Um conjunto de *plugins* (Figura 1-c) para fornecer acesso aos recursos nativos do SO por meio do *WebView* (Figura 1-b). Um *plugin* é implementado em duas partes, a primeira parte (Figura 1-c) é uma biblioteca Javascript e a segunda parte (1-d) corresponde a sua implementação nativa para execução diretamente no SO, expondo os recursos nativos. O código-fonte da aplicação escrito em linguagem Javascript instancia a porção Javascript do *plugin*, e o *WebView* e por sua vez fornece uma interface entre as partes (biblioteca Javascript (Figura 1-c) e nativa (Figura 1-d), assim tornando possível o uso dos recursos nativos (Figura 1-e).

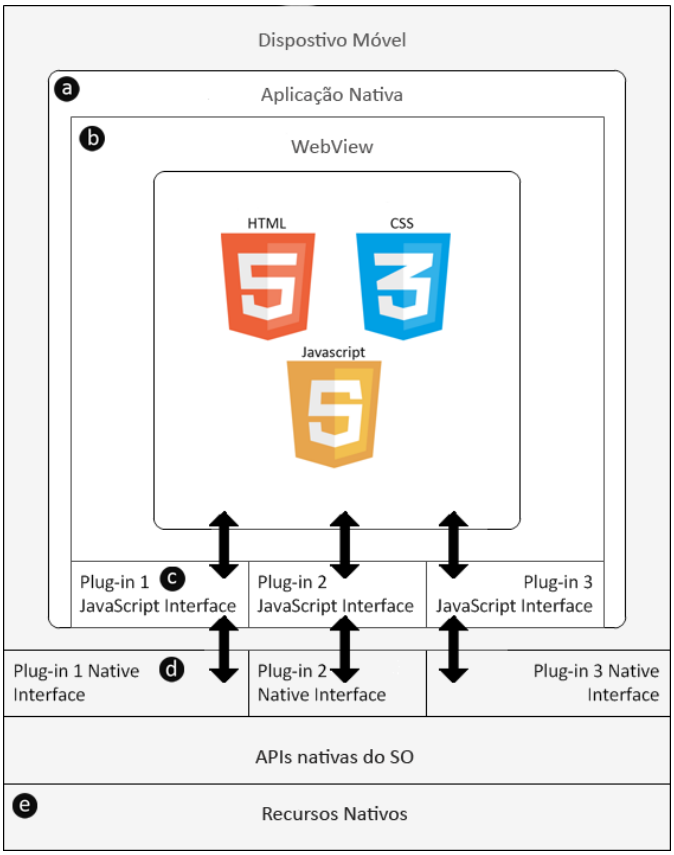

Figura 1: Componentes da arquitetura de uma aplicação híbrida (adaptado de WARGO (2013)).

Além dos *plugins* oficiais criados pela ASF, novos *plugins* podem ser criados por terceiros e disponibilizados para comunidade de desenvolvedores. O site do *framework*[19] disponibiliza um repositório de *plugins* para pesquisa e download.

Os elementos HTML da aplicação, como <*body*> e <*p*> são organizados em tempo de execução em uma estrutura hierárquica de árvore chamada *Document Object Model* (DOM) (W3C, 2017c), a qual oferece uma API para representar e manipular o conteúdo da aplicação

[19]https://cordova.apache.org/plugins

(FLANAGAN, 2013). O código Javascript é utilizado para interagir com esta estrutura: criar, modificar e remover elementos dinamicamente (DEITEL; DEITEL, 2008).

Conforme indicado no site do *framework*, os SOs móveis suportados são: Android, Blackberry, iOS e Windows Phone. Para compilar um projeto de aplicação construído com o Cordova é necessário a instalação do SDK fornecido pela organização proprietária do SO móvel (Android SDK para compilação para o Android e o XCode para o iOS).

Para exemplificar as propriedades e configurações do *framework* foi desenvolvida uma aplicação exemplo no estilo "agenda de contatos". As telas da aplicação podem ser visualizadas na Figura 2. A aplicação é capaz de receber dados de um contato (nome, telefone e e-mail), capturar uma foto ou usar uma foto da galeria de imagens do usuário, e armazenar os dados cadastrados em formato JSON[20] diretamente no sistema de arquivos local. Outras funcionalidades também estão disponíveis, tais como pesquisa de contatos, alteração e exclusão dos dados. A interface de usuário foi construída com HTML e CSS, e com apoio da biblioteca Material Design[21] do Google, que fornece padrões de ícones, cores, animações, tipografia, além de controles de interface apropriados para aplicações móveis. O código Javascript foi empregado no mapeamento das interações do usuário na UI e nas regras de negócio, além da manipulação de *plugins* do Cordova necessários para acesso aos recursos nativos do SO, tais como o sistema de arquivos e a câmera do dispositivo.

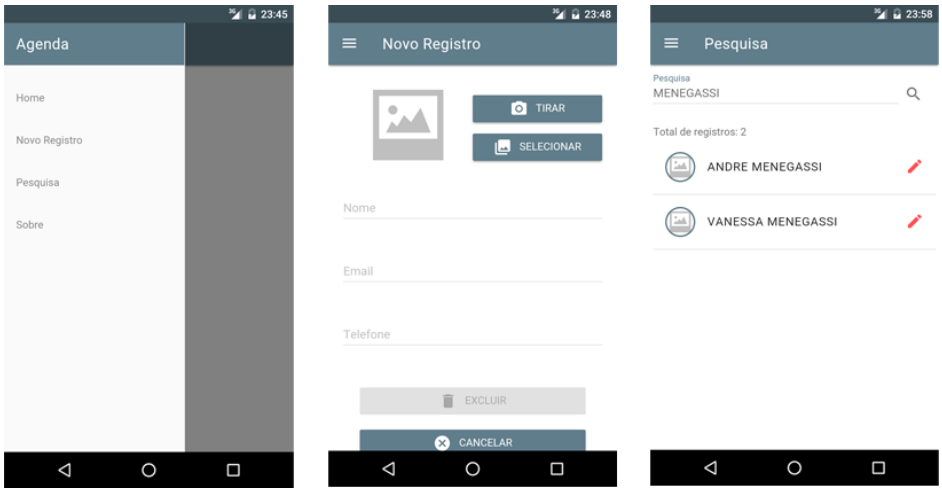

Figura 2: Aplicação de Exemplo (fonte própria).

Um projeto Cordova é organizado em diretórios, seguindo uma estrutura padrão concebida para separar diferentes tipos de arquivos, como descrito a seguir:

[20] *Javascript Notation Object* (JSON) é uma estrutura textual para estruturação de dados e usualmente comum para realizar troca de dados entre ambientes computacionais diferentes.

[21] https://material.google.com/

- **hooks**: Contém *scripts* usados para personalizar os comandos da ferramenta de interface de linha de comando do Cordova (Cordova-Cli).
- **platforms:** Contém o código-fonte nativo das plataformas adicionadas ao projeto. Por exemplo, o código nativo para gerar o *WebView* da plataforma Android e iOS.
- **plugins:** Os *plugins* utilizados no projeto são armazenados nesse diretório.
- **www:** Contém os artefatos da programação Web do projeto, tais como arquivos HTML, CSS e Javascript.

O arquivo *config.xml* (Figura 3) é um arquivo no formato XML baseado na especificação do W3C para empacotamento de aplicações Web (CAMDEN, 2016). Contém dados referente a configuração da aplicação, tais como metadados do projeto (nome e descrição), imagens para ícones e tela de abertura *(splashscreen)*, lista de preferências da aplicação como a orientação padrão (*portrait* ou *landscape*), dados específicos para cada SO móvel, e a lista de *plugins* utilizados na implementação do projeto da aplicação. Essas configurações são utilizadas pelo Cordova no momento da geração do projeto nativo para cada SO móvel.

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="app.agenda" version="1.0.0" xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="
http://cordova.apache.org/ns/1.0">
    <name>Agenda</name>
    <description>
        Agenda
    </description>
    <content src="index.html" />
    <preference name="permissions" value="none" />
    <preference name="orientation" value="portrait" />
    ...
    <platform name="android">
        <allow-intent href="market:*" />
    </platform>
    <platform name="ios">
        <allow-intent href="itms:*" />
        <allow-intent href="itms-apps:*" />
    </platform>
    <icon src="www/icon.png" />
    <splash src="www/splash.png" />
    ...
    <plugin name="cordova-plugin-camera" spec="~2.3.0">
        <variable name="CAMERA_USAGE_DESCRIPTION" value=" " />
        <variable name="PHOTOLIBRARY_USAGE_DESCRIPTION" value=" " />
    </plugin>
</widget>
```

Figura 3: Arquivo de configuração do projeto Cordova (fonte própria).

A Figura 4 é um recorte do código-fonte do exemplo e representa a função manipuladora *getFoto* responsável por tratar a interação do usuário ao informar que deseja selecionar uma foto para um contato da agenda. A Linha 12 demonstra a implementação em Javascript para uso do *plugin* de acesso a câmera do dispositivo. Esse *plugin* define globalmente um objeto *navigator.camera*, o qual fornece uma API para tirar fotos e para escolher imagens

da galeria de imagens do SO. A função *getPicture* recebe três parâmetros: (i) uma função *callback* para manipular a imagem selecionada, (ii) função de *callback* para tratar uma possível ocorrência de erro durante o processo de seleção, e (iii) um objeto literal (definido na Linha 2) com as configurações do *plugin*.

```
1  getFoto: function(sourceType) {
2      var config = {
3          allowEdit: true,
4          quality: 70,
5          encodingType: Camera.EncodingType.PNG,
6          destinationType: Camera.DestinationType.DATA_URL,
7          sourceType: sourceType,
8          targetWidth: 250,
9          targetHeight: 500,
10         mediaType: Camera.MediaType.PICTURE
11     };
12     navigator.camera.getPicture(function(imageData) {
13         imageData = "" + imageData;
14         document.querySelector("#imgCadastro").src = imageData;
15         }, function() {
16             alert("Imagem não foi selecionada");
17         }, config);
18 }
```

Figura 4: Demonstração em nível de código-fonte de um *plugin* para acesso nativo ao recurso câmera (fonte própria).

2.2.4 XAMARIN

O Mono[22] é a implementação *open source* e multiplataforma do Microsoft .NET Framework[23]. Seu projeto inclui um *Common Language Runtime* (CLR)[24] compatível com o .NET Framework, um conjunto de compiladores para linguagem C#, e uma implementação de bibliotecas para execução das aplicações (em inglês, *runtime libraries*). É um *framework* projetado para permitir que os desenvolvedores criem, compilem e executem aplicações (em geral) multiplataforma codificadas em C# (REYNOLDS, 2014). Dessa iniciativa, nasceu o Xamarin, que viabiliza aos desenvolvedores a criação de aplicações móveis multiplataforma com acesso à recursos específicos das plataformas Android, iOS e WindowsPhone (BILGIN, 2016). A Figura 5 oferece uma visão geral da arquitetura do Xamarin. Contida em sua estrutura, o Xamarin.iOS (Figura 5-a) é o MonoTouch (Mono para iOS) e o Xamarin.Android (Figura 5-b) é o Mono para Android. Esses são vinculadores (em inglês, *bindings*) responsáveis por realizar o mapeamento às APIs das plataformas Android e iOS para o desenvolvimento de aplicações móvel usando recursos nativo: sistema de notificações, gráficos, animação e recursos

[22] http://www.mono-project.com/

[23] https://www.microsoft.com/net

[24] É um ambiente gerenciado de tempo de execução *(run-time)* que executa o código e fornece serviços que facilitam o processo de criação de componentes e aplicações cujos objetos interagem entre linguagens (MICROSOFT, 2016a).

de localização, câmera - todos usando C#. Xamarin.Forms (Figura 5-c) é um conjunto de ferramentas totalmente multiplataforma fornecendo um único pacote de elemento de UI, *layouts* e páginas que mapeiam os elementos nas plataformas específicas (HERMES, 2016).

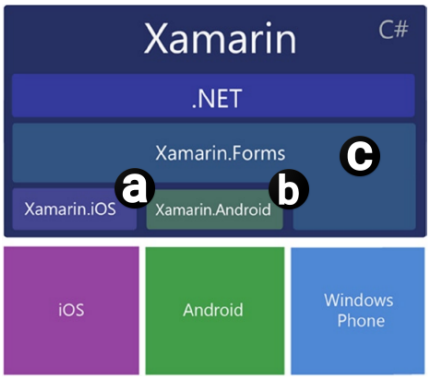

Figura 5: Arquitetura do Xamarin (adaptado de HERMES (2016)).

A UI da aplicação pode ser escrita em C# ou XAML[25]. A Figura 6 demonstra o código C# ou XAML necessário para implementar a UI ilustrada na Figura 7 (campos para entrada de texto, senha e um botão). O resultado final são aplicações com interfaces adequadas aos seus SOs devido ao mecanismo que mapeia os elementos disponibilizado pelo *framework* para elementos nativos da plataforma.

```
var profilePage = new ContentPage {
    Title = "Profile",
    Icon = "Profile.png",
    Content = new StackLayout {
        Spacing = 20, Padding = 50,
        VerticalOptions = LayoutOptions.Center,
        Children = {
            new Entry { Placeholder = "Username" },
            new Entry { Placeholder = "Password", IsPassword = true },
            new Button {
                Text = "Login",
                TextColor = Color.White,
                BackgroundColor = Color.FromHex("77D065") }}}
};
var settingsPage = new ContentPage {
    Title = "Settings",
    Icon = "Settings.png",
    (...)
};
var mainPage = new TabbedPage { Children = { profilePage,
                                             settingsPage } };
```

C#

```
<?xml version="1.0" encoding="UTF-8"?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="MyApp.MainPage">
    <TabbedPage.Children>
        <ContentPage Title="Profile" Icon="Profile.png">
            <StackLayout Spacing="20" Padding="20"
                         VerticalOptions="Center">
                <Entry Placeholder="Username"
                       Text="{Binding Username}"/>
                <Entry Placeholder="Password"
                       Text="{Binding Password}"
                       IsPassword="true"/>
                <Button Text="Login" TextColor="White"
                        BackgroundColor="#77D065"
                        Command="{Binding LoginCommand}"/>
            </StackLayout>
        </ContentPage>
        <ContentPage Title="Settings" Icon="Settings.png">
            <!-- Settings -->
        </ContentPage>
    </TabbedPage.Children>
</TabbedPage>
```

XAML

Figura 6: Codificação da UI da aplicação no Xamarin (adaptado de XAMARIN (2017a)).

Um projeto Xamarin utiliza mecanismos adequados de compilação do código C# para cada plataforma (BILGIN, 2016; XAMARIN, 2017a):

Android O código C# é compilado para uma linguagem intermediária executada pelo Mono *runtime*. Enquanto dentro do dispositivo Android, uma camada chamada *Managed*

[25] O XAML *(eXtensible Application Markup Language)* é uma linguagem de marcação declarativa que simplifica a criação de UIs. É utilizada nos ambientes de desenvolvimento da Microsoft (MICROSOFT, 2016c).

Callable Wrappers (MCW) é responsável por tratar a comunicação entre o *Mono runtime* e o *Android runtime*.

iOS O código C# é compilado para uma linguagem intermediária chamada *ARM assembly language* e então é usado um mecanismo chamado *Mono Touch Ahead-Of-Time compiler—mtouch* que compila o código ARM para o formato binário suportado pela plataforma. Isso significa que as aplicações desenvolvidas com Xamarin são completamente aplicações nativas nessa plataforma.

No desenvolvimento de aplicações nativas-multiplataforma é requerido a configuração do ambiente de desenvolvimento para cada plataforma alvo. Para o Android é requerido a instalação dos SDKs do Java e Android. Elas fornecem um compilador, emulador e outras ferramentas necessárias para geração da aplicação no formato oficial de distribuição. No caso para compilação de aplicações para iOS (ipa/app), é requerido um computador executando o MAC OS e com o XCode instalado.

2.2.5 REACT NATIVE

O React Native é um *framework* mantido pelo Facebook para construir aplicações móveis multiplataforma usando Javascript. Similar ao ReactJS[26] (para aplicações Web), as aplicações são codificadas em uma mistura de Javascript e XML, conhecida como JSX. Também é possível o uso de código Javascript puro. A Figura 8 ilustra um trecho de código JSX e define

[26]https://facebook.github.io/react/

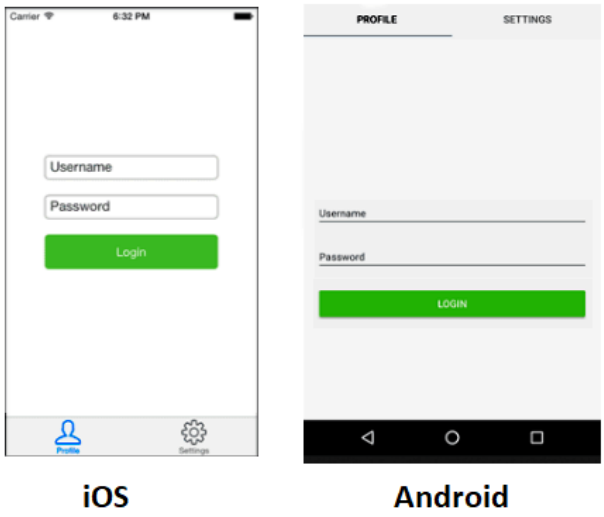

Figura 7: Uma aplicação construída com o Xamarin sob execução nas plataformas iOS e Android (adaptado de XAMARIN (2017a)).

dois elementos de UI: o primeiro exibe um texto estático e o segundo um campo para entrada de texto. A Figura 9 ilustra a UI gerada por esse código. O *framework* utiliza um mecanismo de "ponte" o qual fornece suporte para acessar os recursos nativos do SO e do dispositivo por meio da invocação de APIs em Objective-C (para iOS) ou Java (para Android). Além disso, a renderização da UI é nativa, portanto, utiliza elementos reais da plataforma e não *WebViews* (EISENMAN, 2016).

```
export default class App extends Component {
  state = {
    inputValue: "Type here!"
  };
  _handleTextChange = inputValue => {
    this.setState({ inputValue });
  };
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.paragraph}>
          Hello, world!
        </Text>
        <TextInput
          value={this.state.inputValue}
          onChangeText={this._handleTextChange}
          style={{ width: 200, height: 44, margin: 80,
                   padding: 10, backgroundColor: '#fff',
                   orderWidth: 1, borderColor: '#000' }}
        />
      </View>
    );

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#ecf0f1',
  },
  paragraph: {
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
    color: '#34495e',
  },
});
```

Figura 8: Codificação da UI de uma aplicação usando JSX do React Native (fonte própria).

A transformação de um projeto React Native (Figura 10-a) para uma aplicação nativa, consiste na compilação do código JSX/Javascript (Figura 10-b) através de mecanismos como o JavaScriptCore[27] (quando a aplicação é executada em emuladores ou dispositivos reais) ou V8[28] (quando a aplicação é executada dentro do navegador Google Chrome para sua depuração). O site do *framework* afirma e sem detalhes que ambos mecanismos de compilação são muito semelhantes, mas pode haver algumas inconsistências referente a falta de compatibilidade (REACTNATIVE, 2017). Para execução da aplicação, o SO deve dispor de um ambiente de execução do código Javascript, chamado de Javascript *runtime* (Figura 10-c). O código

[27] http://trac.webkit.org/wiki/JavaScriptCore
[28] https://developers.google.com/v8/

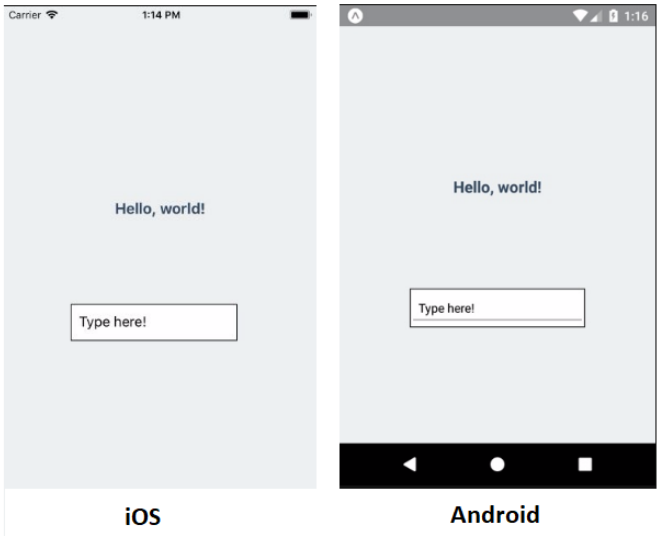

Figura 9: Uma aplicação construída com o React Native sob execução nas plataformas iOS e Android (fonte própria).

compilado é executado nesse ambiente, no qual invoca (ponte) APIs da plataforma (Figura 10-d) para acesso a recursos e renderização de elementos de UI do React para elementos de plataforma (MASIELLO; FRIEDMAN, 2017).

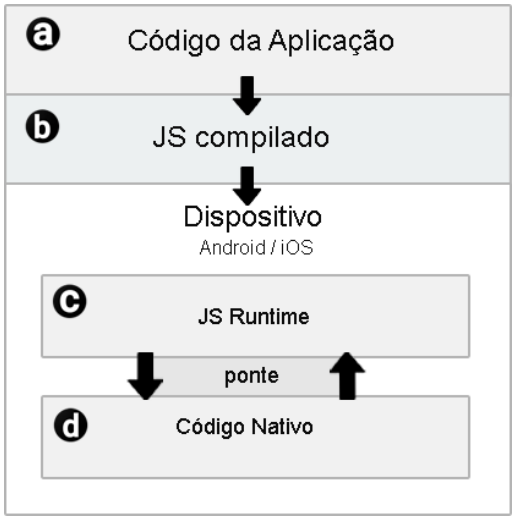

Figura 10: Arquiterua da aplicação construída com o React Native sob execução em plataformas diferentes (adaptado de MASIELLO; FRIEDMAN (2017) e HEARD (2017)).

Tal como o Xamarin, o React Native requer a instalação dos SDKs para compilação do projeto para ambas plataformas.

2.3 FUNDAMENTOS DO TESTE DE SOFTWARE

As aplicações móveis precisam passar por atividades de teste para garantir seu correto funcionamento. Na literatura, a definição para Teste de Software apresentada por MYERS (2004) é utilizada amplamente e define teste como um processo de execução de um programa com a intenção de encontrar defeitos. As atividades de teste ocorrem em três fases (BARBOSA et al., 2004): (i) O teste de unidade é aplicável na menor unidade do projeto de software, com o objetivo de identificar defeitos de lógica e de implementação em cada módulo do software; (ii) o teste de integração objetiva encontrar defeitos durante a integração das partes/módulos do software; e (iii) o teste de sistema é aplicável integralmente ao software objetivando identificar inconsistência no produto desenvolvido de acordo com a especificação.

O teste de software inicia com a definição de um conjunto de casos de teste. Um caso de teste consiste em uma especificação de valores de entrada de teste, condições de execução e resultados esperados, elaborados para atingir um objetivo específico, tal como identificar um defeito, forçar um caminho a ser percorrido dentro do software ou garantir o cumprimento dos requisitos de software (IEEE, 1990). Para MYERS (2004), um caso de teste efetivo é aquele que tem uma alta probabilidade de detecção de um defeito ainda não descoberto.

Na especificação do caso de teste, para cada conjunto de entradas identificadas é necessário especificar as saídas esperadas. Durante a sua execução, o mecanismo capaz de comparar a saída gerada pelo software sob teste com a saída esperada é chamado de oráculo de teste. Portanto, o oráculo de teste é quem determina se o caso de teste passou ou falhou. Em um processo de execução manual do caso de teste, o testador enfrenta a difícil e custosa tarefa de verificar manualmente o comportamento do sistema para todos os casos de teste, fazendo o papel do oráculo, determinando se o caso de teste passou ou falhou (BARR et al., 2015).

Ao contrário do teste manual, o teste automatizado é realizado por um programa, responsável por executar casos de teste, uma ou "n" vezes e avaliar seus resultados identificando se o teste passou ou falhou (HOFFMAN, 2001). O emprego de testes automatizados é uma boa prática para diminuição dos custos envolvidos nas atividades de teste, pois as estratégias tradicionais de teste costumam ser custosas e exigir esforços consideráveis dos testadores. Outro benefício está relacionado ao aumento da produtividade nas atividades de teste (mais testes sendo realizados), além de serem reproduzidos diversas vezes com possibilidade de avaliação

dos resultados gerados e minimizar os erros humanos (OLIVEIRA, 2012).

Os critérios de teste fornecem ao desenvolvedor uma abordagem sistemática e teoricamente fundamentada, utilizados para auxiliar na seleção ou adequação de um conjunto de casos de teste. São estabelecidos, basicamente, a partir de três técnicas: a funcional, a estrutural e a baseada em defeitos, sendo diferenciadas de acordo com a origem das informações utilizadas para estabelecer os requisitos de teste (BARBOSA et al., 2004; MYERS, 2004).

O *teste funcional* é conhecido como teste caixa-preta e tem por objetivo verificar as funções do software, observando somente as entradas e saídas geradas pelo software. Os detalhes da implementação em nível de código não são observados no teste funcional, concentrando-se apenas na busca de circunstâncias em que o programa não se comporta de acordo com suas especificações (BARBOSA et al., 2004; MYERS, 2004).

O *teste estrutural* é conhecido como teste caixa-branca e tem por objetivo verificar a estrutura interna do software para derivar casos de teste (BARBOSA et al., 2004; MYERS, 2004). Os caminhos lógicos do software são testados, gerando casos de teste que verificam o algoritmo por meio de um conjunto de condições de desvios do código, laços de repetição como também as definições e uso de variáveis (DELAMARO et al., 2007). Estão categorizadas com base na complexidade, no fluxo de controle e no fluxo de dados.

O *teste baseado em defeitos* utiliza informações sobre os tipos de defeitos mais frequentes no processo de desenvolvimento de software para derivar os requisitos de testes. A ênfase está nos erros que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência. A Análise de Mutantes e a Semeadura de Erros são abordagem típicas dessa técnica (BARBOSA et al., 2004; MYERS, 2004).

Existe um empenho da comunidade em desenvolver abordagens que automatizem a geração de casos de teste, tornando o processo de teste sistemático e formal; nesse contexto, o Teste Baseado em Modelo (TBM) vem sendo pesquisado (ENDO, 2013). No TBM, os casos de teste são derivados baseando-se em modelagens de um componente ou sistema que descrevem seu comportamento esperado, por exemplo, testes de UI (ISTQB, 2014). Para a construção do modelo de teste algumas técnicas de modelagem são utilizadas para expressar os requisitos e funcionalidades do software sob teste. Espera-se que a técnica de modelagem adotada para o TBM seja bem definida sintática e semanticamente, resultando em testes mais eficientes e efetivos (HIERONS et al., 2009). Exemplos dessas técnicas são o *Event Sequence Graph* (ESG) e o diagrama de Máquina de Estados Finitos *(do inglês, Finite State Machine - FSM)* da *Unified Modeling Language* (UML) (ENDO, 2013).

2.3.1 DESAFIOS NO TESTE DE APLICAÇÕES MÓVEIS

As aplicações móveis são diferentes do software tradicional e requerem técnicas especializadas para teste. Os dispositivos móveis trazem consigo portabilidade e diversidade, características que geram algumas restrições (MUCCINI et al., 2012). Muitas características dos dispositivos móveis influenciam tanto no desenvolvimento como também nas estratégias de testes das aplicações móveis. Duas características principais são a heterogeneidade das configurações de hardware de dispositivos e a variabilidade das suas condições de execução. Tais dispositivos vêm equipados com variados sensores (GPS, bússola, acelerômetro, etc.), vários tamanho e tipo de telas, além de capacidade de processamento diferente (AMALFITANO et al., 2013). Essas características geram desafios na condução dos testes em aplicações móveis. Para MUCCINI et al. (2012), os novos SOs, a diversidade de dispositivos, os recursos limitados, as telas sensíveis ao toque e a sensibilidade ao contexto (ambiente em que o dispositivo está inserido) são peculiaridades das aplicações móveis que implicam nos testes.

Segundo NAGAPPAN; SHIHAB (2016), uma vasta gama de estudos apresenta técnicas para ajudar os desenvolvedores a melhorar em teste em aplicações móveis, em particular, tentando melhorar o teste de UI e de cobertura de sistema. Um dos maiores desafios nessa linha é a incapacidade em atingir uma alta cobertura de código. SCHWEIGHOFER; HERIČKO (2013) sugerem que a diversidade de plataformas apresenta ainda um grande desafio em termos de projetar a melhor utilização do espaço na tela. A UI é apresentada/visualizada diferente com base na resolução de tela do dispositivo e suas dimensões. Os autores recomendam testar a UI no máximo de dispositivos móveis possíveis.

Outro desafio está relacionado a execução dos testes em um único dispositivo e/ou um simulador. No entanto, com o sucesso das múltiplas plataformas, uma grande quantidade de aplicações multiplataforma foram disponibilizadas nas lojas de distribuição. Além disso, as aplicações precisam executar em diferentes hardware e versões do SO. O teste em um único dispositivo não garante o funcionamento correto em outro (NAGAPPAN; SHIHAB, 2016). Os atuais *frameworks* e ferramentas não fornecem o mesmo nível de suporte em diferentes plataformas para testar funcionalidades envolvendo mobilidade, serviços de localização, sensores, diferentes tipos de gestos e entradas de dados (JOORABCHI et al., 2013).

2.3.2 FERRAMENTAS DE TESTE AUTOMATIZADO

No contexto do teste automatizado, as ferramentas de teste são exploradas como forma de auxiliar a padronização e formalização de código para abordagens automatizadas de teste

(OLIVEIRA, 2012). Alguns *frameworks* e suas respectivas características ao suporte a teste em aplicações móveis são identificados e relacionados na Tabela 3. Observa-se que o Appium[29] e Calabash[30] dentre os *frameworks* relacionados, são os únicos *open source* com suporte a teste de aplicações multiplataforma.

O Appium é um *framework open source* para automatizar teste em aplicações móveis nativas, Web ou híbridas. Além disso, é multiplataforma e possibilita automatizar testes para as plataformas iOS e Android, usando uma API do Selenium chamada WebDriver[31]. O uso da API possibilita a reutilização de *script* de teste entre as diversas plataformas. No entanto, o Appium não oferece suporte para gerar esse *script*; isso é uma tarefa do testador que pode codificá-los em diversas linguagens de programação. O Appium é na sua essência um servidor Web que utiliza a arquitetura cliente/servidor para expor uma API REST. Ele recebe conexões de um cliente contendo comandos de automatização da UI e executa-os em um dispositivo móvel. A resposta é uma mensagem HTTP que representa o resultado da execução do comando, como por exemplo o XML da UI contendo o resultado após execução de uma ação. Essa abordagem dá o poder de codificar casos de teste em qualquer linguagem de programação com suporte a clientes HTTP (APPIUM, 2017).

Para condução desta pesquisa e definição de uma abordagem de teste a ser apresentada adiante, o Appium foi selecionado como *framework* de suporte ao teste automatizado. Sua integração com o Selenium[32] foi quesito fundamental nessa decisão. A Selenium tem ampla disseminação na comunidade técnica, com diversos exemplos disponíveis em sites especializados e suporte à geração de *scripts* de teste em vários ambientes de desenvolvimento. Além disso, uma pesquisa online (TESTPROJECT, 2016) sobre ferramentas de automação de teste e tendências para 2016 identificou a ferramenta como a mais utilizada (42% entre os 644 participantes).

2.3.3 DEFEITOS EM APLICAÇÕES MÓVEIS MULTIPLATAFORMA

A falta de compatibilidade da execução da aplicação em configurações diferentes pode gerar defeitos que impactam em seu funcionamento. A aplicação pode funcionar corretamente na plataforma X e falhar na plataforma Y. Os testes precisam ser executados nessas diferentes configurações em função dos defeitos apresentados a seguir.

Em específico para o Cordova, um *plugin* é implementado em duas partes, a primeira

[29] http://appium.io
[30] http://calaba.sh
[31] http://www.seleniumhq.org/projects/webdriver/
[32] http://www.selenium.org

Tabela 3: Ferramentas de teste x Características (adaptado de STEVEN (2016)).

	Tipo de Aplicação			SO				Linguagem do script					Licença		
Características x Ferramentas	*Nativa*	*Web*	*Híbrida*	*Android*	*iOS*	*Universal Windows App*	*Outros*	*Java*	*C#*	*ObjectiveC/Swift*	*Javascript*	*Outras*	*Open Source*	*Comercial*	*Outra*
Calabash	✔		✔	✔	✔							✔	✔		
XCTest UI	✔				✔					✔					✔
Ranorex	✔	✔	✔	✔	✔	✔						✔		✔	
EggPlant	✔			✔	✔	✔	✔					✔			
Robotium	✔	✔	✔	✔				✔					✔		
Monkey Talk	✔	✔	✔	✔	✔							✔	✔		
Monkey Test	✔	✔	✔	✔								✔	✔		
Selendroid	✔	✔	✔	✔				✔			✔	✔	✔		
Appium	✔	✔	✔	✔	✔	✔		✔	✔			✔	✔		
SmartBear	✔	✔	✔	✔	✔			✔	✔	✔	✔	✔		✔	
UI Automator	✔	✔	✔	✔				✔							✔
Robolectric	✔							✔					✔		
Espresso	✔		✔	✔				✔							✔

parte em Javascript e a segunda parte corresponde a sua implementação nativa específica para cada plataforma. Ao entender que tais *plugins* funcionam como uma espécie de componente necessário para implementação da aplicação, erros podem ser introduzidos pelo seu uso, de tal forma a afetar outros *plugins*, além da aplicação no geral. No estudo conduzido por MOHAMED; ZULKERNINE (2010) é observado que a maioria das técnicas de avaliação da confiabilidade de um software não considera as falhas inseridas por componentes.

Outro problema das aplicações híbridas está relacionado a incompatibilidade entre as versões da mesma aplicação em múltiplas configurações. Alguns registros evidenciam tais inconsistências e podem ser encontrados no *BugTracker*[33] da aplicação híbrida Moodle Mobile (MOODLE, 2017):

[33] https://tracker.moodle.org/projects/mobile

- A barra de *scroll* em determinada função relacionada ao conteúdo da aplicação não é visualizada no iOS;
- O botão para apagar uma mensagem no iOS é visualizado em um posicionamento elevado na tela;
- Uma borda não esperada é visualizada no Android.

Em comum com as aplicações de Internet, tais como sites, intranets, portais de *e-commerce*, as aplicações híbridas utilizam código Javascript, dessa forma compartilhando questões relacionadas ao teste de software. A natureza altamente dinâmica e orientada a eventos da linguagem Javascript, bem como sua interação de tempo de execução com o DOM, torna complexa a realização de testes nessas aplicações (MIRSHOKRAIE et al., 2015). Além disso, o interpretador HTML pode gerar resultados diferentes nas diversas plataformas, provocando inconsistências de UI.

No caso das aplicações nativas-multiplataforma, falhas estruturais estão relacionadas ao mecanismo de tradução implementado pelo *framework* para conversão do código original para código intermediário da plataforma de destino. Quando um desenvolvedor usa *frameworks* desse tipo, espera-se que as aplicações se comportem consistentemente durante a execução nas múltiplas plataformas. Infelizmente, essa tradução é uma tarefa complexa e envolve o mapeamento correto das semânticas dos SDKs das plataformas de destinos (BOUSHEHRINEJADMORADI et al., 2015). Complementar a isso, a própria documentação do *framework* React Native sugere que embora os mecanismos de compilação para código específico de plataforma sejam muito semelhantes, pode haver algumas inconsistências. Consultando questões relacionadas ao termo *"cross-platform"* no *BugTracker*[34] do Xamarin, é possível notar evidências dessas falhas (foram encontrados aproximadamente 450 relatos).

2.4 TRABALHOS RELACIONADOS

TAO; GAO (2014) conduziram uma pesquisa no o qual foco está relacionado à modelagem de ambientes de teste e não especificamente ao teste de aplicações móveis. Os autores relatam que o atual processo de teste de aplicações móveis é custoso e tedioso devido a diversidade de ferramentas e tecnologias disponíveis. Nesse sentido, fornecem uma abordagem para modelagem de ambientes de teste de aplicações móveis e discutem a complexidade dos métodos de avaliação desses ambientes. Para demonstrar e analisar os modelos de testes propostos, um estudo de caso utilizou aplicações nativas e híbridas.

[34] https://bugzilla.xamarin.com/query.cgi?format=advanced

ZEIN et al. (2016) conduziram um Mapeamento Sistemático com propósito de identificar pesquisas existentes em técnicas de teste para aplicações móveis, sendo que 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. (2014), no qual faz 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 SO móvel constitui o ambiente de teste, enquanto nas aplicações Web o navegador Web é o ambiente de teste. Os autores levantam questões, tais como a falta de padronização na infraestrutura de teste de aplicações móveis, ferramentas para gerar *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 automatizado que suporta simultâneos testes e em larga escala.

MALAVOLTA et al. (2015a) 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) dentre as mais de 11 mil aplicações selecionadas para a pesquisa, 445 aplicações móveis são híbridas (3,73%), (ii) observaram 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) revelaram o Apache Cordova como o *framework* mais utilizado na construção desse tipo de aplicações. Os pesquisadores estenderam esse trabalho em um novo estudo: MALAVOLTA et al. (2015b), no qual foram coletados na loja do Google 3.041.315 comentários de usuários das 11 mil aplicações selecionadas no primeiro estudo. Enquanto no primeiro estudo o foco foi nas aplicações híbridas, considerando principalmente o ponto de vista dos desenvolvedores, o novo estudo objetivou observar as aplicações sob a perspectiva do usuário final, questões relacionadas a desempenho, presença de defeitos e classificação *(rating)* na loja. Os pesquisadores destacam que o usuário não tem conhecimento para distinguir uma aplicação nativa de uma híbrida, ele somente espera o funcionamento correto da aplicação em seu dispositivo (por exemplo, sem atrasos *(delays)*, com poucas falhas, com uma experiência de usuário natural), independentemente da tecnologia empregada na construção da aplicação. As principais conclusões desse estudo são os seguintes: (i) as aplicações híbridas foram melhores quando tratavam de troca de dados intensos, e piores quando acessavam recursos de baixo nível do dispositivo; (ii) a classificação *(rating)* das aplicativas híbridas e nativas foram similares; (iii) o desempenho das aplicações variou entre as categorias da loja, mas no geral o desempenho das aplicações nativas foi melhor na percepção do usuário.

LEE et al. (2016) desenvolveram um *framework* para análise estática do código das

aplicações híbridas específicas da plataforma Android. Batizado de HybriDroid, a proposta do *framework* é analisar a comunicação entre a linguagem Java e Javascript (ambiente nativo e Web) utilizada nas aplicações híbridas. A análise foi realizada explorando as classes Java extraída de aplicações. Para validação, foram coletadas 88 aplicações híbridas do mundo real baixadas diretamente da loja Google Play. No geral, 14 aplicações foram reportadas com 31 defeitos identificados pelo HybriDroid, sendo observados 24 como verdadeiros e 7 com falsos positivos. A maioria dos defeitos foram classificados como *MethodNotFound*, isso é, quando uma invocação a um método Java a partir do Javascript não é encontrada. Outros defeitos encontrados em menor incidência foram: *MethodNotExecuted* - quando o retorno de um método Java não é compatível com tipo de dados do Javascript, *TypeOverloadedBridgeMethod* - quando o mecanismo de mapeamento não sabe qual sobrecarga de um método Java deve utilizar e *IncompatibleTypeConversion* - quando um método Java tem tipos de argumentos não suportados pelo Javascript.

LI et al. (2017) propuseram uma abordagem e ferramenta chamada ATOM para automaticamente manter atualizado os *scripts* de teste baseados em UI para o teste de regressão da aplicação. A ferramenta usa um *Event Sequence Models* (ESMs) para descrever o comportamento da UI da aplicação, bem como um Delta ESM para capturar as alterações introduzidas por uma nova versão. O Delta ajuda a compreender o impacto das alterações e a fazer ajustes nos *scripts* de teste. O ATOM foi avaliado para manter *scripts* de teste de 11 aplicações Android, usando 22 versões diferentes (duas versões de cada aplicação). A manutenção de *scripts* de teste é importante para aplicações com atualizações constantes. No futuro planeja-se investigar como os *scripts* gerados pela ferramenta a ser apresentada nesta dissertação executam em tal cenário.

JOORABCHI et al. (2015) propuseram uma ferramenta chamada *Checking Compatibility Across Mobile Platforms (CHECKCAMP)*, capaz de detectar e visualizar inconsistências entre as versões para Android e iOS de uma mesma aplicação. A ferramenta gera um modelo (extraído da UI) para cada versão e realiza comparações procurando por inconsistências. Em sua avaliação, 14 pares de aplicações (aplicações contidas nas duas plataformas) mostraram que a abordagem baseada em modelo de UI fornece uma solução eficaz. O CHECKCAMP inferiu corretamente os modelos com uma alta taxa de precisão. Além disso, a ferramenta foi capaz de detectar 32 inconsistências válidas durante a comparação dos modelos das versões das aplicações nas diferentes plataformas. As inconsistências foram classificadas em funcionais e de dados. Inconsistências funcionais estão relacionadas às diferenças entre os modelos da aplicação para cada plataforma, enquanto as inconsistências de dados estão relacionadas às diferenças nos valores armazenados em componentes de UI.

BOUSHEHRINEJADMORADI et al. (2015) realizaram testes em um *framework* de desenvolvimento de aplicações móveis multiplataforma. O estudo sugere que há duas classificações para *frameworks* de desenvolvimento de aplicações multiplataforma: *Web-based framework* e *Native framework*. Essa última classificação utiliza-se de uma plataforma mãe/principal na construção da aplicação, por exemplo, iOS, Android ou Windows Phone - utilizando recursos nativos, e posteriormente o código "mãe" é traduzido e compilado para outra plataforma (plataforma de destino). Exemplos desses *framewoks* são o Xamarin e o Apportable. Especificamente para o Xamarin, foi desenvolvido uma ferramenta chamada X-Checker que utiliza o sequenciamento dos métodos das classes da aplicação na definição dos casos de teste. O objetivo do X-Checker é descobrir inconsistências do *framework* comparando a execução das compilações diferentes (uma para Android e outra iOS) por meio de teste diferencial[35]. Durante a validação da ferramenta, a mesma gerou 22.465 casos de teste, invocando 4.758 métodos implementados em 354 classes em 24 *Dynamic Link Library* (DLLs) do Xamarin. No geral, foram encontradas 47 inconsistências no código do Xamarin.

SONG et al. (2011) objetivaram desenvolver um *framework* de automação de teste com suporte a plataformas heterogêneas. O testador mapeia os eventos sob teste e extraí alguns dados, como nome identificador do elemento, tipo do elemento, posição na tela ou texto. Esses dados fornecem subsídios para gerar casos de teste automatizados *(scripts)* em uma linguagem independente compatível de tradução para os *frameworks* de teste Robotium (Android) e FoneMonkey (iOS). A reutilização do caso de teste gerado é o ponto chave da abordagem, reduzindo drasticamente o tempo de teste em outras plataformas. Para a avaliação do *framework*, foi comparado o teste manual com o teste automatizado pelo *framework*. Um projeto de aplicação móvel foi utilizado e 22 casos de teste foram gerados. Na comparação o teste automatizado alcançou aproximadamente 80% de economia de tempo de execução.

ZHANG et al. (2015) investigam o uso do teste de compatibilidade de aplicações móveis entre dispositivos e plataformas. O teste de compatibilidade é um tipo de teste de sistema que visa validar a dependência entre o software e seus diferentes ambientes de execução (múltiplas configurações). No cenário das aplicações móveis, compatibilidade de plataforma refere-se a validar se a aplicação pode funcionar corretamente em diferentes plataformas móveis e suas várias versões. Nesse contexto é proposto uma estratégia para reduzir os custos e melhorar a eficiência da execução do teste de compatibilidade baseando-se em um algoritmo na qual seleciona os dispositivos móveis, diversas plataformas e configurações. A ideia é

[35]Teste diferencial requer a disponibilização dois ou mais sistemas comparáveis disponíveis para realização de testes. Os sistemas são expostos a uma séria exaustiva de casos de teste. Os resultados das execuções são comparados e podem expor problemas de interesse para a equipe de desenvolvimento (MCKEEMAN, 1998).

selecionar um conjunto de dispositivos móveis com características similares, dessa forma, diminuindo o número de dispositivos a serem usados nos testes das aplicações. Além disso, é proposto um serviço em nuvem para realização dos testes de compatibilidade para as aplicações. Para avaliar o serviço foram selecionados 20 dispositivos móveis populares e duas aplicações. Em um outro estudo conduzido por CHENG et al. (2015), também é abordado questões referente ao teste de compatibilidade, mas nesse caso foi utilizado algoritmos genéticos na seleção dos dispositivos. Os dois estudos sugerem que esse de tipo de teste ajuda na definição de um conjunto de dispositivos móveis adequado para realização do teste em uma aplicação móvel.

FAZZINI et al. (2017) implementaram uma técnica para teste em aplicações móveis que consiste em três fases principais: (i) gravação da interação do usuário com a aplicação, visando testar sua funcionalidade. A técnica oferece uma interface conveniente para definir oráculos de teste baseados em asserções; (ii) geração de casos de teste baseados na fase de gravação da interação do usuário com a aplicação; (iii) a técnica sugere a execução dos casos de teste gerados em vários dispositivos e resume os resultados em um relatório. Em detalhes, a fase de geração de casos de teste produz como saída um caso de teste que reproduz fielmente as ações executadas pelo usuário durante a fase de gravação. Tal caso de teste é baseado no *framework* de teste Android chamado Espresso (GOOGLE, 2017). Ao final da execução, a técnica sugere um relatório de execução de casos de teste que contém o resultado da execução em cada dispositivo, contendo o tempo de execução e informações sobre ocorrências de erros ou falhas. Os oráculos são gerados com base nas propriedades dos elementos selecionados na gravação do caso de teste. A técnica foi implementada em um *framework* chamado Barista, que foi avaliado em um estudo empírico envolvendo 15 sujeitos humanos e 15 aplicações Android reais. No total, 206 casos de teste foram registrados e executados em sete dispositivos reais. No geral, a taxa de compatibilidade média em todas as aplicações e dispositivos foi de 99,2%. Dois casos de teste falharam devido a um espaço adicional inserido abaixo de um elemento do tipo *TableLayout*. Na comparação da execução dos casos de teste entre o Barista e a ferramenta TestTroid Record (TESTDROID, 2017), o Barista foi superior.

WEI et al. (2016) conduziram um estudo empírico para entender e caracterizar questões referente a fragmentação da plataforma Android. As numerosas combinações de diferentes modelos de dispositivos e versões do SO tornam impossível para os desenvolvedores Android testar exaustivamente as aplicações móveis. Os pesquisadores realizaram um estudo empírico baseado na análise estática do código-fonte das aplicações Android. O objetivo foi entender e caracterizar os problemas relacionados a fragmentação das aplicações. O estudo deu origem a uma técnica chamada FicFinder para detectar automaticamente problemas de

compatibilidade. Para validar FicFinder, os autores investigaram 191 problemas reais de compatibilidade coletados a partir de cinco aplicações Android (populares e *open source*) para entender suas causas, sintomas e estratégias de correção. As causas de problemas de compatibilidade mais comuns estão relacionadas a evolução da API da plataforma e problemas com o *driver* de hardware.

Appium Studio (EXPERITEST, 2017) é um Ambiente Integrado de Desenvolvimento (em inglês, *Integrated Development Environment*) (IDE) projetado para auxiliar na automação de teste de aplicações móveis utilizando o *framework* Appium (APPIUM, 2017) (detalhado na Secão 2.3.2) e a API do Selenium WebDriver. Ele suporta escrever, gravar e executar testes para aplicações (nativas, Web e híbridas) Android e iOS. A ferramenta elimina a maioria das rígidas dependências e pré-requisitos para o uso do Appium, tais como a necessidade de desenvolver testes para iOS em máquinas com o MAC OS e a incapacidade de executar testes em paralelo em dispositivos reais e simulados. Como seu principal objetivo é ser um *front-end* para o Appium, ele não lida com testes automatizados de aplicações multiplataforma tal como proposto nesta dissertação. No entanto, um *plugin* que incorpora a proposta a ser apresentada adiante e estratégias de localização de elementos poderiam trazer resultados promissores ao IDE.

A Tabela 4 reapresenta os trabalhos relacionados agrupando-os por características comuns identificadas entre eles. É observado que a maioria dos trabalhos abordam aplicações nativas (principalmente para plataforma Android) e exploram a implementação e avaliação de ferramentas para suporte ao teste. Apenas alguns trabalhos exploraram o teste em multiplataforma. É possível comparar esses trabalhos com a abordagem a ser definida adiante nesta dissertação. A seguir são elencadas algumas comparações:

- A abordagem para teste apresentada nesta dissertação pode ser utilizada em ambientes de teste em nuvem, tal como proposto por TAO; GAO (2014), no qual fornecem uma abordagem para modelar ambientes de teste para aplicações móveis;
- Esta pesquisa difere dos trabalhos de MALAVOLTA et al. (2015a e 2015b) ao ampliar o estudo para aplicações nativas-multiplataforma, além das híbridas;
- Diferentemente da abordagem definida desta dissertação, o trabalho relacionado condizido por LEE et al. (2016) não abrange a execução da aplicação e a realização de testes automatizados. Além disso, somente aplicações para a plataforma Android foram analisadas;
- A abordagem proposta por LI et al. (2017) é capaz de automaticamente manter atualizado os *scripts* de teste baseados em elementos de UI para o teste de regressão da aplicação.

No futuro, essa abordagem pode complementar a abordagem proposta nesta dissertação;

- Esta dissertação apresenta uma abordagem similar em relação a adotada por JOORABCHI et al. (2015), mas sem a necessidade de instrumentação do código (abordagem *black-box*), o testador rastreará e gravará os eventos da aplicação, de tal forma a criar seus próprios testes para aplicações multiplataforma;

- Esta pesquisa difere do trabalho conduzido por BOUSHEHRINEJADMORADI et al. (2015) ao se concentrar em testes de aplicações móveis multiplataforma, e não no teste do *framework* de desenvolvimento das aplicações;

- Diferentemente da abordagem proposta nesta dissertação que se concentra em apenas um *framework* de teste, o trabalho de SONG et al. (2011) apresenta a geração de um *script* independente de *framework*, com a possibilidade de sua tradução para outros *frameworks*.

- O trabalho de ZHANG et al. (2015) trás uma estratégia para melhorar a seleção de dispositivos móveis para o teste de compatibilidade de aplicações entre as diversas plataformas. Além apresentar uma proposta de seleção de dispositivos para o teste multiplataforma, a abordagem apresentada nesta dissertação inclui a geração do mecanismo único de teste entre as plataformas/dispositivos;

- A abordagem proposta por FAZZINI et al. (2017) possibilita a definição de oráculos de teste baseados em assertivas, além da conversão dos casos de teste para um *framework* Android. A abordagem proposta nesta dissertação vai além ao gerar *script* para teste de aplicação multiplataforma;

- A abordagem definida nesta dissertação é mais ampla ao compará-la com a apresentada por WEI et al. (2016), devido a execução do código da aplicação e a definição de mecanismos comuns de compatibilidade entre diferentes ecossistemas. No futuro é pretendido identificar falhas de uma perspectiva da execução do teste;

- A abordagem apresentada nesta dissertação complementa o Appium Studio (EXPERITEST, 2017) para criar *scripts* de teste adaptados para execução em vários dispositivos de configurações diferentes. Um *plugin* pode ser desenvolvido como trabalho futuro.

Tabela 4: Resumo dos Trabalhos Relacionados.

Trabalhos Relacionados	Tipo de Aplicação	Plataforma	Análise Estática e Teste	Ferramenta de Suporte	Multiplataforma
MALAVOLTA et al. (2015a)	Híbrida	Android	Estática		
MALAVOLTA et al. (2015b)	Nativa e Híbrida	Android	Estática		
LEE et al. (2016)	Híbrida	Android	Estática	Sim	
JOORABCHI et al. (2015)	Nativa	Android e iOS		Sim	Sim
BOUSHEHRINEJADMORADI et al. (2015)	Nativa	Android e iOS	Estática	Sim	Sim
SONG et al. (2011)	Nativa	Android e iOS		Sim	Sim
ZHANG et al. (2015)	-	Android e iOS			Sim
FAZZINI et al. (2017)	Nativa	Android		Sim	
WEI et al. (2016)	Nativa	Android	Estática	Sim	
EXPERITEST (2017)	Nativa e Híbrida	Android e iOS		Sim	Sim
TAO; GAO (2014)	Nativa e Híbrida	Não mencionado			
LI et al. (2017)	Nativa	Android	Estática	Sim	

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo foram introduzidos os principais conceitos e características sobre a computação móvel, SOs móveis, teste de software em aplicações móveis (e seus diversos tipos), aplicações multiplataforma e *frameworks* para desenvolvimento dessas aplicações também foram descritos. Os *frameworks* Apache Cordova, Xamarin e React Native foram detalhados. Sobre teste de software foram evidenciados os principais conceitos, com foco maior em teste de aplicações móveis. De modo geral, os desafios encontrados no teste de aplicações móveis são elencados, ferramentas de teste automatizado, além dos trabalhos relacionados com esta dissertação.

No próximo capítulo, é apresentada uma abordagem para teste de software para aplicações móveis multiplataforma, principal proposta desta dissertação, a qual tem como principal objetivo contribuir para as atividades de teste desse tipo aplicação.

3 UMA ABORDAGEM PARA O TESTE DE APLICAÇÕES MÓVEIS MULTIPLATAFORMA

3.1 CONSIDERAÇÕES INICIAIS

Este capítulo apresenta e detalha a abordagem proposta para o teste de aplicações móveis multiplataforma. A Seção 3.2 apresenta a descrição do problema referente a UI que impacta no teste de aplicações. A Seção 3.3 introduz a abordagem proposta e suas etapas para selecionar dispositivos móveis, definir um modelo de teste, casos de teste, estratégias para localização de elemento de UI e formalizar um mecanismo adequado para o teste automatizado de UI em múltiplas configurações. E por fim, a Seção 3.4 apresenta uma ferramenta de apoio ao teste multiplataforma chamada `x-PATeSCO`, desenvolvida a partir da abordagem proposta.

3.2 DESCRIÇÃO DO PROBLEMA

As plataformas móveis organizam a UI das aplicações em estruturas XML que possibilitam a localização de seus elementos de interface para o teste automatizado. Uma estrutura XML pode conter um ou mais nós (ou elementos), e são usados para especificar algum tipo de informação, como por exemplo os elementos de UI da aplicação. Esses nós são complementados com dados que dão melhor semântica aos elementos nos quais estão contidos, chamados de atributos (W3C, 2017a). A Figura 11-c ilustra um trecho de uma estrutura XML, no caso um elemento do tipo *android.view.View* e seus vários atributos, por exemplo, *index="2"* (2 é seu respectivo valor).

Em aplicações híbridas, a UI é construída com elementos HTML, mas durante sua execução são transformados em elementos específicos da plataforma organizados em XML. A Figura 11-a apresenta a UI de uma aplicação híbrida e ilustra o elemento HTML <*a*> (Figura 11-b) transformado em um elemento nativo Android (Figura 11-c) e iOS (Figura 11-d). É possível perceber que essa estrutura difere entre as plataformas Android e iOS. A diferença também ocorre entre versões da mesma plataforma (MENEGASSI; ENDO, 2016). A Tabela 5 apresenta um breve mapeamento entre os elementos de UI HTML e XML gerados pelas

plataformas Android e iOS.

Figura 11: Representação de um elemento HTML mapeado para a UI do Android e iOS; *screenshot* da aplicação Fresh Food Finder (TRICE, 2017).

Tabela 5: Mapeamento de elementos HTML para elementos nativos de plataforma específica (MENEGASSI; ENDO, 2016).

Tipo de Elemento HTML	Tipo de Elemento Android	Tipo de Element iOS
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

No caso dos *frameworks* de desenvolvimento de aplicações nativas-multiplataforma, tal mapeamento ocorre diretamente para XML. Esse tipo de *framework* possibilita o uso de elementos de UI nativos das plataformas alvo (XAMARIN, 2017a; REACTNATIVE, 2017). Portanto, a aplicação final (uma para cada plataforma) utiliza elementos nativos da sua plataforma para representar a UI. A Figura 12 apresenta a estrutura XML de um elemento de UI em uma aplicação (Figura 12-a) desse tipo sob execução no Android (Figura 12-b) e iOS (Figura 12-c). Idem as aplicações híbridas, o XML também é diferente entre o Android e iOS, devido aos mecanismos próprios desses SOs para representar a UI das aplicações.

Conforme apresentado, os SOs móveis provêm uma estrutura XML para representação da UI (Figuras 11 e 12). Nesse contexto, seletores são utilizados para localizar elementos de UI nessa estrutura. Os seletores são "padrões" ou "modelos" que fornecem mecanismos para

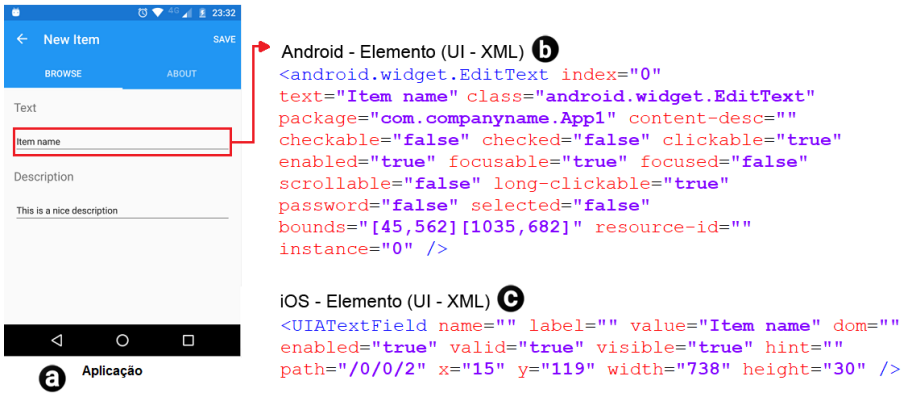

Figura 12: XML representando um elemento de UI da aplicação nativa-multiplataforma Tasky (XAMARIN, 2017b).

localizar elementos (ou nós) em estruturas computacionais, tais como XML ou HTML (W3C, 2011). XPath é uma linguagem de consulta para escrever expressões capazes de selecionar elementos (nós) em estruturas computacionais chamadas de árvore tais como documentos XML (W3C, 2017d). Seu uso pode ser aplicado ao contexto do teste automatizado de UI das aplicações móveis ao viabilizar a localização de elementos. A Figura 13 ilustra na prática o uso do seletor necessário para localizar o elemento destacado na Figura 11-a e simular a sua ação de clique no teste automatizado usando a API WebDriver do Selenium.

```
1  ...
2  AppiumDriver<IWebElement> _driver= new AndroidDriver<IWebElement>(defaultUri, _capabilities);
3  ...
4  IWebElement e = _driver.FindElementByXPath("//*/android.view.View[@content-desc='Search For a Market']");
5  e.Click();
```

Figura 13: Seleção de um elemento e seu uso no teste automatizado.

Os nós da estrutura XML podem ser compostos por atributos que ajudam a definir a sua semântica. Nesta pesquisa várias estruturas de UI foram analisadas e observou-se que alguns atributos contêm dados descritivos sobre o elemento de UI e normalmente estão disponíveis na interface para visualização pelo usuário, dado que seus valores integram a representação visual do elemento. Para a plataforma Android, os atributos identificados foram *"content-desc"* e *"text"*, enquanto para a plataforma iOS os atributos foram *"label"* e *"value"*. Esses atributos foram aqui nomeados como "atributos chave". Também foi verificado que os atributos *"resource-id"* (para Android) e *"name"* (para iOS) mantêm informações sobre a identificação do elemento, portanto, nomeados como "atributos identificadores".

As Figuras 11 e 12 sugerem que os fabricantes das plataformas não definem uma estrutura XML padrão, e tal falta de compatibilidade reflete negativamente nas atividades de teste de UI para aplicações multiplataforma. A aplicação é desenvolvida com recursos que

permitem sua execução multiplataforma, mas os mecanismos para testá-la não são. Dessa forma, diferentes *scripts* de teste são necessários para automatizar os testes da mesma UI, cada qual com seletores adequados para atender as especificidades de plataforma e suas versões. Para ilustrar, o seletor XPath necessário para localizar o elemento destacado na Figura 11-a (botão "Search For a market") adequado para Android 6.0.3 é ***"//*/android.view.View[@content-desc='Search For a Market']"*** e para iOS 9.3 é ***"//*/UIAStaticText[@label='Search For a Market']"***. Essas XPaths usam o tipo do elemento e atributos chave específicos de plataforma. Apenas o valor do atributo manteve-se igual entre as duas plataformas.

3.3 ABORDAGEM PROPOSTA

O objetivo da abordagem proposta é a definição de um mecanismo para automatizar o teste de aplicações móveis multiplataforma, mediante a definição de um caso de teste a partir da inspeção da UI da aplicação (teste funcional) e sua conversão em um *script* de teste capaz de testar uma aplicação em múltiplas configurações. A Figura 14 ilustra a abordagem em três etapas principais: *a)* Um dispositivo de referência é escolhido para cada plataforma, um Android e outro iOS, então chamados de C_{Ref1} e C_{Ref2} (Figura 14-a, Seção 3.3.1). *b)* Define-se um modelo orientado a eventos para representar os casos de teste e expressões XPath individuais para localizar elementos de UI em ambas plataformas (Figura 14-b, Seção 3.3.2). *c)* Propõe-se a geração de um *script* de teste adequado para o teste em múltiplas configurações e define-se duas novas estratégias que combinam as expressões individuais (Figura 14-c, Seção 3.3.3). As três etapas são detalhadas nas subseções a seguir.

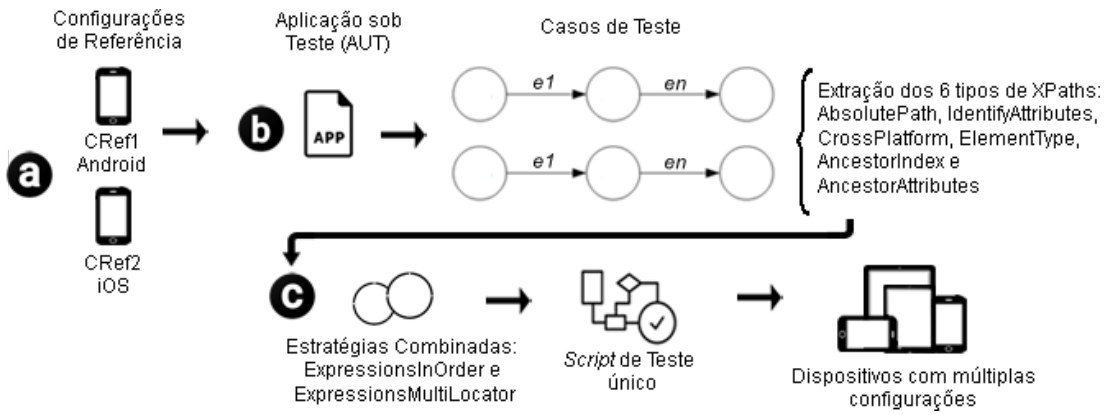

Figura 14: Visão geral da abordagem.

3.3.1 SELEÇÃO DE DISPOSITIVOS

Essa etapa consiste na seleção de dispositivos móveis de referência, uma para cada plataforma. Neste estudo, um dispositivo Android e um iOS foram selecionados com base na popularidade entre os usuários da plataforma e nomeados como configurações de referência: C_{Ref1} e C_{Ref2} (Figura 14-a). Tais dispositivos serão a base para definição do mecanismo único de teste. Alguns estudos (VILKOMIR et al., 2015; VILKOMIR; AMSTUTZ, 2014) propõem a escolha de dispositivos móveis para teste de aplicações baseado em sua popularidade de uso. Em aplicações específicas para uma organização empresarial, na qual seus usuários e a variedade de dispositivos são controlados, a seleção das referências pode vir a partir de uma demanda existente.

3.3.2 SELEÇÃO DE ELEMENTO DE UI E DEFINIÇÃO DE CASOS DE TESTE

A segunda etapa consiste na definição de um modelo orientado a eventos para expressar os casos de teste funcionais para uma aplicação móvel multiplataforma sob teste *(em inglês, App Under Test - AUT)*, instalada nos dispositivos de referência. É necessário um modelo de teste para representar a sequência de eventos de interações do usuário. Dessa forma, um ESG (BELLI et al., 2006) foi utilizado para modelar os casos de teste como uma sequência de eventos de UI (nós) conectados por arestas. Para cada evento, um elemento de UI é selecionado e alguns dados são fornecidos, tais como o nome identificador do evento e o tipo de ação a ser executada (por exemplo, clique ou digitação de texto). Essa etapa é reproduzida em ambos dispositivos de referência selecionados, C_{Ref1} e C_{Ref2}, e dois ESG compatíveis são gerados (Figura 15). A compatibilidade dos ESGs está relacionada à igualdade de número de eventos e nos dados fornecidos para os elementos contidos nos eventos das referências.

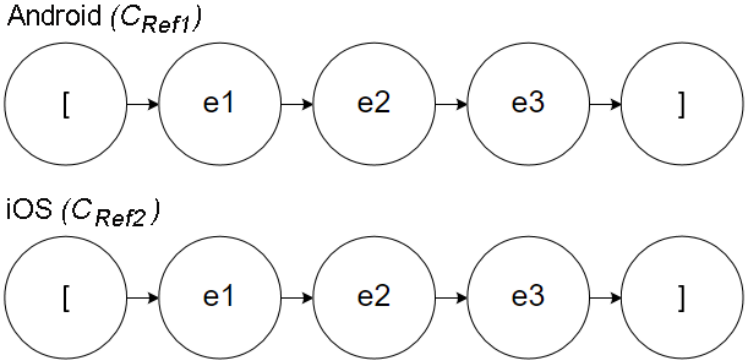

Figura 15: Eventos sob teste mapeados em um ESG.

O processo de seleção de elementos para o caso de teste é baseado no critério de teste funcional (caixa-preta), no qual o testador inspeciona a UI da aplicação em execução

para gerar o caso de teste. Nesse processo é definido o modelo de teste e a estrutura XML da UI da aplicação é extraída das C_{Ref1} e C_{Ref2}. Baseada nessa estrutura, cada elemento de UI tem seu tipo identificado (caixa texto, botão, âncora, etc), bem como seus atributos chave e identificadores, e seus respectivos valores são armazenados (Seção 3.2). Esses dados fornecem subsídios para a construção de diferentes tipos de expressões de consulta XPath capazes de localizar o elemento para o evento sob teste. Este estudo investiga seis tipos de expressões de consulta individuais, três baseadas em outros trabalhos (RAO; PACHUNOORI, 2013; LEOTTA et al., 2013a, 2013b), além de sugerir outras três. Elas foram batizadas com nomes que representam sua estratégia para localizar um elemento e descritas juntamente com exemplos referente ao elemento de UI destacado na Figura 11-a. Os exemplos são apresentados a seguir.

AbsolutePath: É uma expressão específica da plataforma, baseada no caminho absoluto a partir do nó inicial da estrutura XML. Em alguns casos, o índice[1] do elemento é requerido para identificar sua posição dentro dessa estrutura. Essa expressão tem sido empregada em testes de aplicações Web para localizar elementos dentro da estrutura DOM (LEOTTA et al., 2013a). É uma alternativa bem conhecida quando o elemento não tem um atributo de identificação ou atributos chave, e foi adotada nesse trabalho. Um exemplo é mostrado a seguir:

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: `UIAApplication/UIAWindow/UIAScrollView/UIAWebView/UIALink[2]`

IdentifyAttributes: É uma expressão baseada nos valores de atributos que identificam o elemento, tais como, *resource-id* para plataforma Android e *name* para plataforma iOS. O uso de tais expressões é bem conhecido para teste em aplicações Web (LEOTTA et al., 2013a, 2013b). O atributo identificador *(id)* é uma das primeiras estratégias para localizar elementos HTML. No exemplo a seguir, as expressões procuram a partir da raiz (//) da estrutura XML um elemento de qualquer tipo (*) que contenha o atributo (@) *resource-id* (para Android) ou *name* (para iOS) e seus respectivos valores especificados:

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

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

CrossPlatform: É uma expressão proposta neste estudo e define uma expressão única para plataformas diferentes. Tal expressão é preparada para selecionar um determinado elemento da UI da aplicação independente da sua plataforma de execução. Ela combina os atributos chave (Android: *content-desc* ou *text* e iOS: *label* ou *value*) dos elementos e seus respectivos valores

[1] As expressões XPath tem o índice iniciado na posição 1 (W3C, 2017d).

em ambas plataformas, discutidos na Seção 3.2. Nessa abordagem, foi combinado os atributos das duas plataformas em uma expressão única. A expressão do exemplo a seguir procura a partir da raiz da estrutura XML (//) um elemento de qualquer tipo (*) que contenha o atributo (@) *content-desc* (para Android) ou (or) *label* (para iOS) e seus respectivos valores especificados:

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

ElementType: É uma expressão preparada para localizar um elemento baseado na combinação de seu tipo e seus respectivos atributos chave (Android: *content-desc* ou *text* e iOS: *label* ou *value*) e atributos identificadores de plataforma específica (Android: *resource-id* ou iOS: *name*). Rao e Pachunoori (2013) sugerem a combinação do tipo de elemento nas expressões XPath. Nesse trabalho esse tipo de expressão foi adaptada e prioriza o tipo do elemento em combinação com os atributos chave e identificador. A prioridade são os atributos chave em relação ao atributo identificador. Essa combinação pode assegurar uma localização singular, isso é, evitar que um outro elemento seja localizado erroneamente devido a possuir os mesmos valores de atributos. No exemplo a seguir, as expressões procuram por elementos a partir da raiz da estrutura (//), dentro de qualquer ascendente (*/) um elemento de determinado tipo e atributo (@) com valor específico:

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

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

AncestorIndex: É uma expressão proposta neste estudo e baseia-se no índice do elemento desejado contido em seu elemento ascendente. O índice define a exata posição do elemento, nesse caso, dentro do elemento ascendente. Essa expressão é híbrida ao considerar que o elemento ascendente é localizado por uma expressão de consulta relativa (baseada nos atributos chave e atributos identificadores) e o elemento interno é localizado pelo índice (posicionamento absoluto). Esse tipo de expressão pode ajudar a localizar o elemento quando as expressões baseadas em atributos falharem devido às mudanças dinâmicas; cada nova execução produz diferentes valores de atributos. Além disso, o uso do elemento ascendente assegura uma localização singular, pois delimita o escopo de localização do elemento alvo. As expressões do exemplo ilustrado a seguir procuram um elemento desejado de qualquer tipo (*) por meio do seu índice ([?]) dentro de outro elemento ascendente localizado a partir da raiz da estrutura (//) por meio dos valores dos atributos chave ou identificador (o que aparecer primeiro):

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

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

AncestorAttributes: Semelhante a expressão anterior, mas a localização por índice do elemento desejado é substituída por uma localização baseada nos valores dos atributos chave da plataforma específica. A seguir as expressões de exemplo procuram um elemento desejado de qualquer tipo (*) por meio de atributos chave (@) dentro de outro elemento ascendente localizado a partir da raiz da estrutura (//) mediante os valores dos atributos chave ou identificador (o que aparecer primeiro):

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

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

Como cada plataforma pode apresentar uma estrutura XML e valores de atributos diferentes, um tipo de expressão nem sempre é aplicável em todas as plataformas e suas versões. Essa desigualdade implica na seleção dos elementos. Com exceção do tipo *CrossPlatform*, cada expressão possui duas versões, uma para cada plataforma, geradas a partir dos dados extraídos das configurações de referência.

3.3.3 MECANISMO ÚNICO DE TESTE

Essa etapa estabelece um mecanismo adequado para teste automatizado de UI em múltiplas configurações. Nesse caso, um *script* de teste comum entre diferentes plataformas. O caso de teste representado pelo ESG (Figura 15) é base para sua construção, uma vez que cada evento sob teste contém dados sobre um elemento de UI, assim como as expressões de consulta capazes de selecioná-lo.

O Algoritmo 1 apresenta um pseudocódigo do *script* para a execução das seis expressões individuais. O método *Exec* (Linha 2) recebe por parâmetro as expressões de consulta XPath, uma para cada plataforma, além da plataforma alvo da execução corrente. O método localiza um elemento na estrutura XML da UI da aplicação utilizando as expressões (Linha 8). Ao final, o elemento encontrado é executado de acordo com a ação indicada pelo testador (Linha 12). Quando o elemento não é encontrado uma exceção é lançada para falhar a execução do caso de teste (Linhas 9-10). Estruturas lógicas condicionais conduzem o fluxo do *script* para atender a cada plataforma (Linhas 3-7).

Como as expressões individuais anteriormente mencionadas podem ser limitadas em alguns contextos, são propostas duas novas estratégias de localização que combinam as seis expressões, *ExpressionsInOrder* e *ExpressionsMultiLocator*, apresentadas a seguir.

ExpressionsInOrder: Para selecionar o elemento vinculado ao evento, as expressões são ordenadas por seu tipo e executadas sequencialmente. Na falha da primeira expressão, a

Algoritmo 1 Pseudocódigo - Execução das expressões individuais.

```
1: Input
   xPathSelectorAndroid - Seletor de elemento para Android
   xPathSelectorIOS - Seletor de elemento para iOS
   platform - Nome da plataforma sob teste
2: procedure EXEC(xPathSelectorAndroid, xPathSelectorIOS, platform)

3:     if platform = "Android" then
4:         xPathSelector = xPathSelectorAndroid;
5:     else if platform = "iOS" then
6:         xPathSelector = xPathSelectorIOS;
7:     end if

8:     e = FindElementByXPath(xPathSelector);

9:     if e == null then
10:        throw new exception("Element not found");
11:    else
12:        e.action();
13:    end if
14: end procedure
```

próxima é executada, e assim sucessivamente. O objetivo 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 *CrossPlatform* devido sua adequação para selecionar elementos de UI no Android e no iOS. Alguns estudos (LEOTTA et al., 2015, 2014; RAO; PACHUNOORI, 2013) sugerem fragilidades na localização de elementos por expressões absolutas, sendo assim, estão *(AncestorIndex e AbsolutePath)* ao final da lista. A sequência definida para execução foi:

1° *CrossPlatform*;
2° *ElementType*;
3° *IdentifyAttributes*;
4° *AncestorAttributes*;
5° *AncestorIndex*;
6° *AbsolutePath*.

O Algoritmo 2 apresenta um pseudocódigo para a estratégia proposta. O método *ExecInOrder* (Linha 2) recebe como parâmetro um conjunto de seis expressões XPath ordenadas, e localiza um elemento dentro da estrutura XML da UI da aplicação usando as expressões (Linhas 9-14). Ao final, o elemento encontrado (por alguma das expressões) é executado de acordo com a ação indicada pelo testador (Linha 18). Quando um elemento não é encontrado, uma exceção é lançada para falhar o caso de teste (Linhas 15-16). Estruturas condicionais direcionam o fluxo do evento para atender a cada plataforma.

Algoritmo 2 Pseudocódigo - Execução ExpressionsInOrder

```
1: Input
   xPathCrossPlatform - Seletor multiplataforma de elemento (CrossPlatform)
   xPathsAndroid[] - Seletores de Elemento para Android (AbsolutePath, IdentityAttributes, CrossPlatform, ElementType, AncestorIndex e
   AncestorAttributes)
   xPathsiOS[] - Seletores de Elemento para iOS (AbsolutePath, IdentityAttributes, CrossPlatform, ElementType, AncestorIndex e
   AncestorAttributes)
   platform - Nome da plataforma sob teste
2: procedure EXECINORDER(xPathCrossPlatform, xPathsAndroid[], xPathsiOS[], platform)

3:     xPathSelectors[] = xPathCrossPlatform;

4:     if platform = "Android" then
5:         xPathSelectors[] += xPathsAndroid;
6:     else if platform = "iOS" then
7:         xPathSelectors[] += xPathsiOS;
8:     end if

9:     for each selector in xPathSelectors[] do
10:        e = FindElementByXPath(selector);
11:        if e != null then
12:            break;
13:        end if
14:    end for

15:    if e == null then
16:        throw new exception("Element not found");
17:    else
18:        e.action();
19:    end if
20: end procedure
```

ExpressionsMultiLocator: Nessa estratégia, 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. (2015), a qual é empregada no teste de aplicações Web. A seguir são discutidos e apresentados os critérios de definição de pesos. A Tabela 6 foi adaptada a partir do estudo citado anteriormente e relaciona os pesos normalizados entre 0 e 1 definidos para cada expressão.

- As expressões de consulta baseadas em valores de atributos chave (visualizados na UI) possuem peso alto, devido a constância em manter seus valores e não dependem de índices e/ou caminho absoluto para sua localização. Exemplos desses atributos são o *content-desc*, *text*, *label* e *value*. As expressões contidas nesse grupo são: *ElementType*, *CrossPlatform* e *AncestorAttributes*;

- As 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 aplicações, os valores dos atributos identificadores de elementos sofrem alterações durante as diferentes execuções, dessa forma a confiabilidade na seleção é comprometida. Um exemplo desse tipo de expressão é a *IdentifyAttributes*;

- As 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 da aplicação (LEOTTA et al., 2015, 2014; RAO; PACHUNOORI, 2013) ou quando o conteúdo é dinâmico. Exemplos dessas expressões são: *AbsolutePath* e *AncestorIndex*.

Tabela 6: Tipos de expressões e pesos.

Tipos de Expressão	Confiabilidade	Peso
CrossPlatform	Alta	0,25
ElementType	Alta	0,25
AncestorAttributes	Alta	0,25
IdentifyAttributes	Média	0,15
AbsolutePath	Baixa	0,05
AncestorIndex	Baixa	0,05

O Algoritmo 3 apresenta um pseudocódigo para a estratégia *ExpressionsMultiLocator*. Tal como na estratégia combinada anterior, o método *ExecMultiLocator* (Linha 2) recebe como parâmetro um conjunto de seis expressões de consulta. Mas nesse caso, todas as expressões são executadas e para cada elemento encontrado seu peso é extraído de acordo com o tipo da expressão corrente. Um mesmo elemento retornado por diferentes expressões tem seu peso acumulado. Ao final, o elemento com maior votação (somatório dos pesos) é usado na execução do caso de teste (Linhas 24-25). Quando um elemento não é encontrado uma exceção é lançada para falhar a execução do caso de teste (Linhas 21-22). Nessa primeira versão da estratégia não foi implementado critérios de desempate quando elementos diferentes obtêm o mesmo peso, sendo assim, o primeiro elemento contido na estrutura de dados é selecionado para o teste.

3.4 IMPLEMENTAÇÃO DA FERRAMENTA

A abordagem proposta foi implementada em uma ferramenta chamada `x-PATeSCO` *(cross-Platform App Test Script reCOrder)* desenvolvida com o Microsoft .NET *Framework*, linguagem C# e Microsoft VisualStudio 2017. O projeto da ferramenta possui 5.850 linhas de código (LOC) e sua arquitetura está baseada em quatro classes principais:

- *DeviceConfig*: Representa um dispositivo de referência e mantém dados como sua plataforma, versão e parâmetros necessários para estabelecer a conexão com o Appium;
- *MyNode*: Representa um evento sob teste. Mantém os dados dos eventos e elementos selecionados, por exemplo, os dados fornecidos pelos testadores e o XML que representa o elemento na UI na aplicação;

Algoritmo 3 Pseudocódigo - Execução ExpressionsMultiLocator

```
1: Input
   xPathCrossPlatform - Seletor multiplataforma de elemento (CrossPlatform)
   xPathsAndroid[] - Seletores de Elemento para Android (AbsolutePath, IdentityAttributes, CrossPlatform, ElementType, AncestorIndex e
   AncestorAttributes)
   xPathsiOS[] - Seletores de Elemento para iOS (AbsolutePath, IdentityAttributes, CrossPlatform, ElementType, AncestorIndex e
   AncestorAttributes)
   platform - Nome da plataforma sob teste
2: procedure EXECMULTILOCATOR(xPathSelectorsCrossPlatform[], xPathSelectorsAndroid[], xPathSelectorsiOS[], platform)

3:     xPathSelectors[] = xPathSelectorsCrossPlatform;

4:     if platform = "Android" then
5:         xPathSelectors[] += xPathSelectorsAndroid;
6:     else if platform = "iOS" then
7:         xPathSelectors[] += xPathSelectorsiOS;
8:     end if

9:     elements[];
10:    voting[];

11:    for each selector in xPathSelectors[] do
12:        e = FindElementByXPath(selector);
13:        if e != null then
14:            if !elements.Contains(e) then
15:                elements.Add(e);
16:                voting[e] = 0;
17:            end if
18:            voting[e] = voting[e] + ExtractWeight(selector);
19:        end if
20:    end for
21:    if elements.length == 0 then
22:        throw new Exception("Element not found")
23:    else
24:        e = MaxElement(voting);
25:        e.action();
26:    end if
27: end procedure
```

- *XPathSelector*: Responsável por manipular o XML extraído da UI para gerar as seis expressões individuais e duas estratégias combinadas propostas pela abordagem;
- *ScriptGenerate*: Responsável por gerar o projeto de teste contendo o *script* de teste único. O projeto pode ser opcionalmente gerado instrumentado para avaliação experimental da abordagem.

A `x-PATeSCO` foi baseada no *framework* Appium (APPIUM, 2017) descrito na Seção 2.3.2 e sua arquitetura (Figura 16) utiliza o WebDriver do Selenium para gerar comandos de automatização da UI da aplicação, os quais são encaminhados para o Appium que conecta-se aos dispositivos e processa os comandos. A UI da aplicação é recuperada pela ferramenta e permite ao testador interagir com seus eventos e elementos para elaborar e gravar o caso de teste. Os casos de teste podem ser gravados em arquivos no formato JSON e posteriormente reabertos para edição. No arquivo também é gravado o XML da UI que dará origem as expressões. Ao final, o testador pode gerar o projeto de teste e executá-lo em múltiplas configurações. Em sua versão de avaliação, o projeto é instrumentado para coletar e persistir os dados sobre a execução

dos *scripts* em arquivos no formato JSON, posteriormente usados na avaliação da abordagem.

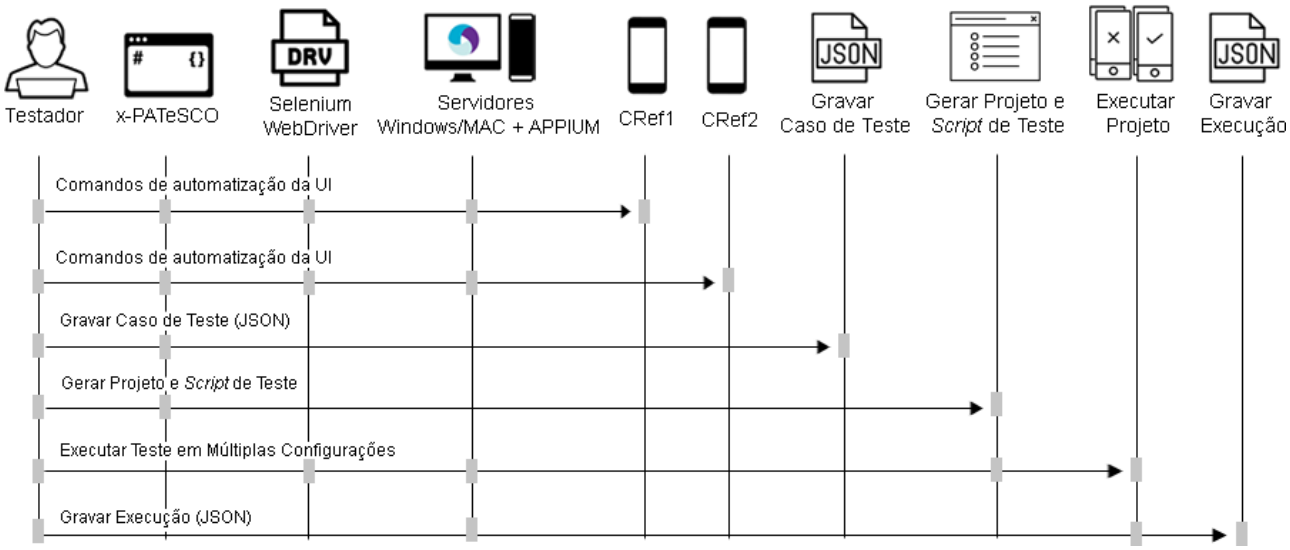

Figura 16: Arquitetura da ferramenta x-PATeSCO.

A Figura 17 foi extraída da ferramenta e apresenta a funcionalidade para realizar a conexão ao Appium, selecionar elementos de UI (para o teste) e configurar os eventos (clique ou entrada de texto). No entanto, a ferramenta não oferece suporte para complexas interações de usuário, como multi-toque e gestos (por exemplo, pinça). Uma última funcionalidade é indicar para o testador a aplicabilidade das seis expressões. Essa informação possibilita identificar se o elemento contém a estrutura XML necessária para a geração das expressões. A aplicabilidade pode indicar se realmente uma aplicação pode ser testada em outras plataformas, visto que quanto menor o número de estratégias aplicáveis, menor pode ser o sucesso de execução do *script* de teste nas múltiplas configurações.

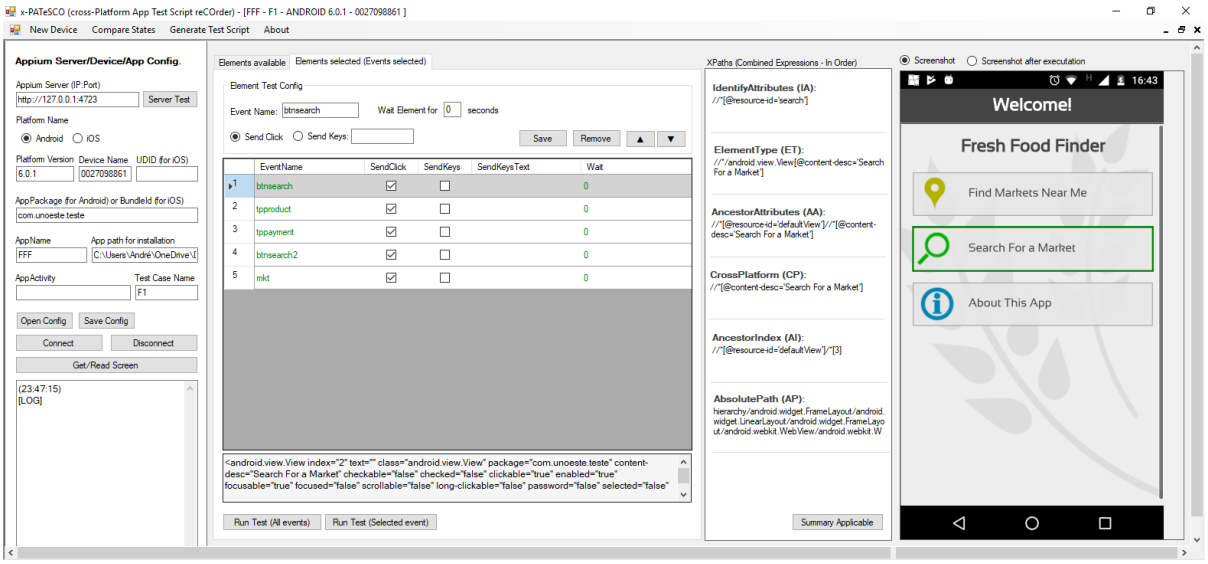

Figura 17: Ferramenta x-PATeSCO.

Com base nos dados extraídos das referências, a ferramenta produz um projeto de teste contendo o *script* codificado em C# para o Microsoft Visual Studio com suporte ao Unit Testing Framework (MICROSOFT, 2017). Também é possível somente gerar a classe de teste que pode ser importada em projetos existentes. Cada classe do projeto representa um caso de teste da aplicação. A Figura 18 mostra a interface para gerar o projeto de teste e qual das oito estratégias de localização (Locator Strategies) será usada pelo *script*. Entretanto, considerando ampliar seu uso, a ferramenta foi adaptada para dar suporte ao teste em apenas uma plataforma e suas versões. Por conseguinte, aplicações de plataforma única podem aproveitar os recursos da `x-PATeSCO` para gerar o *script* de execução do caso de teste.

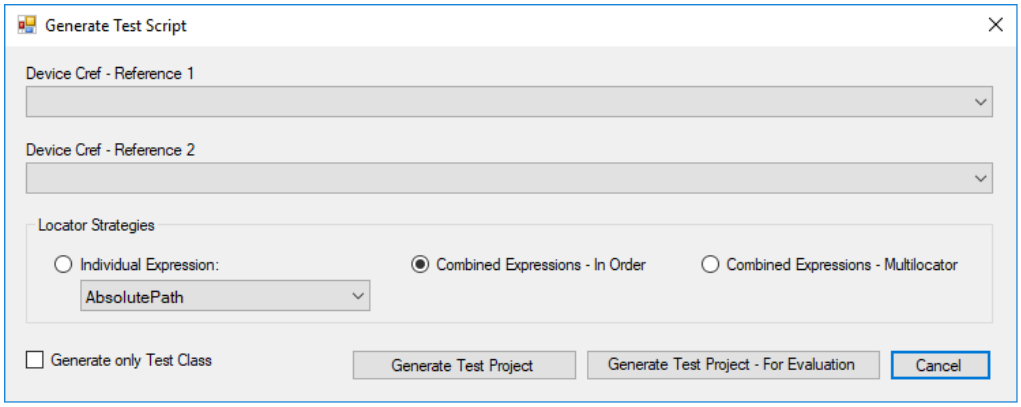

Figura 18: Ferramenta x-PATeSCO - Funcionalidade para gerar o *script* de teste.

A Figura 19 ilustra a janela *Solution Explorer* do Microsoft VisualStudio contendo a organização do projeto de teste gerado pela `x-PATeSCO`. Nessa ilustração três casos de teste

(F1, F2 e F3) foram automatizados para a aplicação Fresh Food Finder (TRICE, 2017). Para cada caso de teste, oito classes de teste foram geradas, cada qual implementa uma estratégia de localização de elementos de UI. No final do projeto, a classe *ProjectConfig* (Figura 20) contém os dados sobre a configuração do projeto, tais como parâmetros para o Appium, plataforma e um conjunto de dispositivos disponíveis para o teste (Figura 20 - Linhas 18-32). A classe *LocatorStrategy* implementa os métodos de localização dos elementos usados pelas classes dos casos de teste.

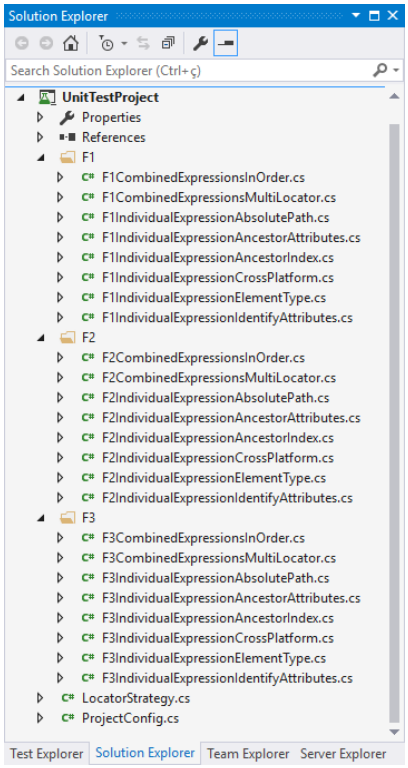

Figura 19: Organização do projeto de teste gerado pela Ferramenta x-PATeSCO.

A Figura 21 ilustra um trecho do *script* de teste único para múltiplas configurações gerado pela ferramenta. Para cada evento selecionado, um método é criado no *script* (Linhas 32-36) contendo as expressões XPath adequadas para as plataformas (Linhas 44-51), necessárias para localizar o elemento sob teste. O método utiliza uma das oito estratégias, conforme a indicação do testador (Figura 18-Locator Strategies). O exemplo demonstra a estratégia individual *ElementType*. Ao final (Linhas 55-57), a expressão é executada, e a ação gravada é disparada. Além disso, o *script* é devidamente configurado para conectar-se ao Appium (Linhas 5-28) de acordo com a plataforma corrente usada no teste, bastando ao testador apenas indicar as assertivas do teste (Linha 59). O *script* de teste pode ser usado em outros dispositivos de múltiplas configurações *(Cn)*, garantindo o teste da aplicação em uma maior abrangência de dispositivos. Nesse caso, uma opção é sua utilização em ambientes de teste em nuvem, na qual

```
1   public static class ProjectConfig
2   {
3       public static string OutputPath { get; private set; }
4       public static string OutputDeviceID { get; private set; }
5       public static string PlataformName { get; private set; }
6       public static string PlatformVersion { get; private set; }
7       public static string DeviceName { get; private set; }
8       public static string AppPackage { get; private set; }
9       public static string AppPath { get; private set; }
10      public static string AppiumServer { get; private set; }
11      public static string Uuid { get; private set; }
12
13      public static int IndexDeviceUnderTest { get; private set; }
14
15      static ProjectConfig()
16      {
17          /*Configure your device here */
18
19          var configs = new[]{
20                            new { OutputDeviceID = "Android6.0.1-MotoG4", PlataformName = "Android",
21                                  PlatformVersion = "6.0.1", DeviceName = "", AppPackage = "", Uuid = "", AppPath = @""},
22                            new { OutputDeviceID = "Android5.1-MotoG1", PlataformName = "Android",
23                                  PlatformVersion = "5.1", DeviceName = "", AppPackage = "", Uuid = "", AppPath = @""},
24                            new { OutputDeviceID = "IOS9.3-iPad", PlataformName = "iOS",
25                                  PlatformVersion = "9.3", DeviceName = "", AppPackage = "", Uuid = "", AppPath = @""},
26                            new { OutputDeviceID = "IOS7.1.2-iPhone", PlataformName = "iOS",
27                                  PlatformVersion = "7.1.2", DeviceName = "", AppPackage = "", Uuid = "", AppPath = @""},
28                            new { OutputDeviceID = "IOS10.2-iPad4", PlataformName = "iOS",
29                                  PlatformVersion = "10.2", DeviceName = "", AppPackage = "", Uuid = "", AppPath = @""},
30                            new { OutputDeviceID = "Android4.4-TableSamsung", PlataformName = "Android",
31                                  PlatformVersion = "4.4", DeviceName = "", AppPackage = "", Uuid = "", AppPath = @""}
32          };
33
34          OutputPath = @"c:\temp";
35          int indexDeviceUnderTest = 5;
36
37          IndexDeviceUnderTest = indexDeviceUnderTest;
38
39          OutputDeviceID = configs[indexDeviceUnderTest].OutputDeviceID;
40          PlataformName = configs[indexDeviceUnderTest].PlataformName;
41          PlatformVersion = configs[indexDeviceUnderTest].PlatformVersion;
42          DeviceName = configs[indexDeviceUnderTest].DeviceName;
43          AppPackage = configs[indexDeviceUnderTest].AppPackage;
44          AppPath = configs[indexDeviceUnderTest].AppPath;
45
46          if (PlataformName == "Android")
47          {
48              AppiumServer = "http://127.0.0.1:4723/wd/hub";
49          }
50          else if (PlataformName == "iOS")
51          {
52              Uuid = configs[indexDeviceUnderTest].Uuid;
53              AppiumServer = "http://192.168.159.129:4723/wd/hub";
54          }
55      }
56
57  }
```

Figura 20: Classe de configuração do projeto gerada pela ferramenta.

oferecem um grande número de dispositivos.

Ao propor o mecanismo definido pela abordagem e implementar a ferramenta com base no Appium, é esperado construir uma infraestrutura de teste que funcione em ambientes para o teste de aplicações em múltiplas configurações como Amazon Device Farm (AMAZON, 2017), Bitbar (BITBAR, 2017) e TestObject (TESTOBJECT, 2017). Esses serviços oferecem um grande número de dispositivos reais que podem ser conectados e usados em testes de aplicações móveis multiplataforma. O BitBar e TestObject suportam acesso advindos de clientes Appium, portanto poucos ajustes no mecanismo para gerar o *script* de teste para incluir os parâmetros necessários à conexão aos serviços, tornariam a ferramenta compatível com os serviços. No caso do Amazon Device Farm, não há suporte a clientes Appium, mas é possível

```
1   ...
2   [TestMethod]
3   public void TestMethodMain()
4   {
5           /*APPIUM config*/
6           _capabilities.SetCapability("platformName", ProjectConfig.PlataformName);
7           _capabilities.SetCapability("platformVersion", ProjectConfig.PlatformVersion);
8           _capabilities.SetCapability("deviceName", ProjectConfig.DeviceName);
9           _capabilities.SetCapability("appPackage", ProjectConfig.AppPackage);
10          _capabilities.SetCapability("newCommandTimeout", "3000");
11          _capabilities.SetCapability("sessionOverride", "true");
12
13          Uri defaultUri = new Uri(ProjectConfig.AppiumServer);
14
15          if (ProjectConfig.PlataformName == "Android")
16          {
17                  _capabilities.SetCapability("app", ProjectConfig.AppPath);
18                  _capabilities.SetCapability("appActivity", ProjectConfig.AppActivity);
19                  _driver = new AndroidDriver<IWebElement>(defaultUri, _capabilities, TimeSpan.FromSeconds(3000));
20          }
21          else if (ProjectConfig.PlataformName == "iOS")
22          {
23                  _capabilities.SetCapability("automationName", "XCUITest");
24                  _capabilities.SetCapability("app", ProjectConfig.AppPath);
25                  _capabilities.SetCapability("bundleId", ProjectConfig.AppPackage);
26                  _capabilities.SetCapability("udid", ProjectConfig.Uuid);
27                  _driver = new IOSDriver<IWebElement>(defaultUri, _capabilities, TimeSpan.FromSeconds(3000));
28          }
29
30          /*initialing test*/
31
32          btnsearchSendClick_Test(); //btnsearch
33          tpproductSendClick_Test(); //tpproduct
34          tppaymentSendClick_Test(); //tppayment
35          btnsearch2SendClick_Test(); //btnsearch2
36          mktSendClick_Test(); //mkt
37  }
38
39  public void btnsearchSendClick_Test()
40  {
41          ForceUpdateScreen();
42          string[] selectors = new string[0];
43
44          if (ProjectConfig.PlataformName == "Android")
45          {
46                  selectors = new string[] {@"//*/android.view.View[@content-desc='Search For a Market']"};
47          }
48          else if (ProjectConfig.PlataformName == "iOS")
49          {
50                  selectors = new string[] {@"//*/UIAStaticText[@label='Search For a Market']"};
51          }
52
53          string[] selectorsType = new string[] {@"ElementType"};
54
55          IWebElement e = _locator.FindElementByXPath(selectors[0], selectorsType[0]);
56
57          e.Click();
58
59          /*Insert your assert here*/
60  }
61  ...
62
```

Figura 21: *Script* de teste gerado pela ferramenta.

enviar ao serviço um *script* codificado em Java, o que seria um passo pequeno em uma nova versão da ferramenta (gerar *scripts* de teste em Java).

Estão disponíveis livremente no GITHUB um repositório contendo o projeto, código-fonte e os artefatos necessários para compilação da ferramenta `x-PATeSCO` (MENEGASSI; ENDO, 2018).

3.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a abordagem e a ferramenta de apoio ao teste em aplicações multiplataforma, a qual é composta por três etapas, seleção de dispositivos, definição de casos de teste e seleção de elementos de UI para um modelo de teste, e a formalização de um mecanismo para criar *scripts* de teste capazes de testar a mesma aplicação em diversos dispositivos. A abordagem é implementada pela ferramenta, qual é capaz de gerar projetos de teste em C# para o Microsoft VisualStudio com suporte ao Unit Testing Framework.

O próximo capítulo apresenta uma avaliação dessa abordagem e da ferramenta de apoio. A avaliação objetiva investigar a efetividade e desempenho das expressões individuais e estratégias combinadas propostas, bem como verificar a capacidade da abordagem em contribuir para o teste de aplicações móveis em múltiplas configurações.

4 AVALIAÇÃO DA ABORDAGEM PROPOSTA

4.1 CONSIDERAÇÕES INICIAIS

Este capítulo apresenta o estudo experimental conduzido para avaliar o mecanismo proposto pela abordagem e implementado em uma ferramenta para gerar *scripts* de teste para aplicações móveis multiplataforma em múltiplas configurações. As descrições das etapas conduzidas neste estudo experimental são apresentadas a seguir. Na Seção 4.2 são apresentadas as Questões de Pesquisa (QPs) a serem respondidas e suas métricas para avaliação da abordagem. A Seção 4.3 apresenta o processo de execução do experimento e os recursos utilizados. Na Seção 4.4 é apresentada a análise dos resultados obtidos. Na Seção 4.5, é apresentada a discussão dos resultados. E na Seção 4.6, são evidenciadas as possíveis ameaças à validade desta pesquisa.

4.2 DEFINIÇÃO DO EXPERIMENTO

Para avaliar o mecanismo de teste proposto pela abordagem, é necessário entender o desempenho das estratégias de localização (expressões individuais e combinadas), analisar o comportamento com diferentes aplicações móveis, e verificar os resultados obtidos em várias configurações. Portanto, foi conduzido uma avaliação experimental para comparar as oito estratégias de localização: seis expressões individuais (Seção 3.3.2) e duas estratégias combinadas resultante da combinação das expressões individuais (Seção 3.3.3).

A seguir, duas Questões de Pesquisa (QPs) foram investigadas:

- **QP1:** *Qual é a efetividade das estratégias de localização para testar aplicações móveis multiplataforma em várias configurações?*
- **QP2:** *Como as estratégias de localização se comportam em relação ao tempo de execução?*

A **QP1** objetiva comparar a efetividade das estratégias de localização por meio da

análise do quão aplicável são suas expressões e se o teste baseado nelas pode ser executado com sucesso em diferentes configurações. Primeiramente, foi medido a *aplicabilidade* para observar em quantos eventos cada estratégia pode ser usada para selecionar elementos da UI. A *executabilidade* foi verificada para compreender o quão bem-sucedido (para selecionar um elemento em tempo de execução) uma estratégia pode ser. Em seguida, foi analisado a executabilidade em dois níveis, eventos e caso de teste.

Para a **QP2**, foi objetivado analisar como a estratégia de localização pode influenciar o tempo de execução dos casos de teste. Então, foi medido o tempo de CPU em segundos dos eventos executados com sucesso para cada uma das estratégias.

As questões QP1 e QP2 foram analisadas sob três perspectivas: geral, *framework* de desenvolvimento e plataforma móvel.

4.3 PROCEDIMENTO DO EXPERIMENTO

Para responder às QPs algumas etapas foram definidas. Foi selecionado um conjunto de nove aplicações móveis capazes de serem executadas nas plataformas Android e iOS, e construídas por *frameworks* de desenvolvimento multiplataforma. Neste trabalho foi considerado três *frameworks*, Apache Cordova (Seção 2.2.3), Xamarim (Seção 2.2.4) e React Native (Seção 2.2.5). No total, duas aplicações industriais e sete aplicações *open source* foram utilizadas no experimento. Empresas de TI parceiras forneceram os projetos das aplicações industriais contendo os ativos necessários para sua compilação no Android e iOS. Os projetos das outras aplicações foram obtidos em livros sobre desenvolvimento de aplicações híbridas e multiplataforma, e em repositórios no GITHUB. A Tabela 7 lista as aplicações selecionadas e algumas características de seus respectivos projetos: tipo da aplicação, número de linhas de código (LOC), principal linguagem de programação, tipo de aplicação multiplataforma e *framework* de desenvolvimento.

O ambiente de desenvolvimento específico para o Android e iOS foi configurado para compilação dos projetos das aplicações selecionadas. Para a compilação dos projetos iOS, foi necessário assinar a aplicação com um certificado de desenvolvedor fornecido pela Apple. Para executar o teste automatizado, uma aplicação assinada e dispositivos móveis registrados são requeridos para a plataforma iOS. Isso restringe o experimento somente para aplicações que dispunham do código-fonte. O resultado da compilação foi o binário para a instalação das aplicações em seis dispositivos reais.

A Tabela 8 apresenta os seis dispositivos reais utilizados no experimento, três com

Tabela 7: Aplicações sob teste.

Aplicação	Tipo	LOC	Principal LP	Tipo de aplicação Multiplataforma	Framework de Desenvolvimento
Fresh Food Finder	Open Source	13.824	Javascript	Híbrida	Cordova
Pedido App	Open Source	71.565	Javascript	Híbrida	Cordova
MemesPlay	Industrial	5.484	Javascript	Híbrida	Cordova
Agenda	Open Source	1.038	Javascript	Híbrida	Cordova
TodoListCordova	Open Source	9.304	Javascript	Híbrida	Cordova
MovieApp	Open Source	2.088	Javascript	Nativa Multiplataforma	ReactNative
TodoList	Open Source	405	Javascript	Nativa Multiplataforma	ReactNative
Tasky	Open Source	654	C#	Nativa Multiplataforma	Xamarin
Ofertados	Industrial	178.266	C#	Nativa Multiplataforma	Xamarin

Android e três com iOS. Os dispositivos com Android 6.0.1 e iOS 9.3 foram selecionados como configurações de referência. A abordagem sugere o uso de dispositivos populares entre seus usuários como configurações de referência (Seção 3.3). O site STATISTA (2017a, 2017d) indica a versão 6 do Android como a mais utilizada por usuários da plataforma até setembro de 2017, e versão 9 do iOS foi a mais utilizada em 2016. Portanto, as versões das configurações de referência do experimento estão entre as mais usadas: Android 6.0.1 e iOS 9.3.

Tabela 8: Dispositivos avaliados.

Dispositivo	SO	Tela (Polegadas)	CPU	RAM
Samsung Tab E	Android 4.4.4	7	Quad Core 1.3 GHz	1 GB
Motorola G4 *(CRef)*	Android 6.0.1	5.5	Octa Core 1.4 GHz	2 GB
Motorola G1	Android 5.1	5	Quad Core 1.2 GHz	1 GB
iPhone 4	iOS 7.1.2	3.5	Duo Core 1 GHz	512 MB
iPad 2 *(CRef)*	iOS 9.3	9.7	Duo Core 1 GHz	512 MB
iPad 4	iOS 10.2	9.7	Dual-Core 1.4 Ghz	1 GB

A ferramenta `x-PATeSCO` (Seção 3.4) foi utilizada para apoiar a avaliação experimental. Ela conecta-se ao Appium que, por sua vez, conecta-se aos dispositivos móveis de referência para extrair dados e gerar as expressões de consulta. Posteriormente, o Appium é usado pelo *script* gerado para executar o teste em múltiplas configurações. Específicas versões do Appium (de acordo com o SO móvel) foram instaladas e configuradas em dois servidores de teste para acesso via rede de computadores local. Os recursos de software usados no experimento foram os seguintes:

- XCode 7.3 (para compilação das aplicações para o iOS 7);

- XCode 8.2 (para compilação das aplicações para o iOS 9.3 e 10.2);
- Android SDK r24.4.1 (para compilação das aplicações para Android);
- VisualStudio 2017;
- Appium 1.4.3 (Windows OS) e 1.5/1.6.3 (MAC OS);
- Selenium WebDriver for C# 2.53.0;
- Appium WebDriver 1.5.0.1;
- Windows 10 OS;
- MAC OS X Sierra.

As nove aplicações (Tabela 7) foram divididas entre quatro participantes independentes com conhecimento em teste de software; foi solicitado que eles explorassem as aplicações e projetassem dois ou três casos de teste para cada. No total considerando todas as aplicações, os participantes selecionaram 118 eventos organizados em 26 casos de teste. Cada caso de teste tem uma sequência de eventos representando exatamente as interações do usuário, na qual foram mapeadas pelos participantes para um modelo ESG com auxílio da ferramenta FourMA (FARTO; ENDO, 2017) e posteriormente reconstruído na `x-PATeSCO` para gerar o projeto de teste contendo o *script* único. Cada evento contém um elemento de UI selecionado para o teste e alguns dados que fornecem subsídios para a construção das expressões de localização de elementos. Para cada aplicação essa etapa foi reproduzida em ambos dispositivos de referência, e ao final dois ESG compatíveis foram gerados. A ferramenta gerou um projeto de teste para cada aplicação no qual implementam as oito estratégias e coletam as métricas proposta nesta avaliação (Seção 4.2). A ferramenta `x-PATeSCO` e os objetos do experimento estão disponíveis como um projeto no GITHUB (MENEGASSI; ENDO, 2018).

A execução dos projetos de teste ocorreu em um computador com processador Intel Core i7 dual-core (2,9 GHz) e 8 GB de RAM, sem qualquer outro processamento ou carga de comunicação, a fim de evitar a saturação de CPU e memória.

4.4 ANÁLISE DOS RESULTADOS

Os projetos contendo os *scripts* de teste foram executados nas seis configurações e dados foram coletados, processados e analisados para responder as questões de pesquisas propostas. Esta seção apresenta e discute os resultados obtidos.

A Tabela 9 apresenta os dados sobre os testes executados para cada aplicação. Ela demonstra o número de casos de teste, quantos eventos foram testados, número de linha de código (LOC) do *script* de teste gerados pela ferramenta `x-PATeSCO` e a média de LOC por

caso de teste. Cada caso de teste deu origem a uma classe contendo o *script* de teste, além disso o projeto de teste implementa os mecanismos de testes definidos nos Algoritmos 1, 2 e 3 para execução das oito estratégias definidas pela abordagem.

Tabela 9: Dados sobre os testes.

Aplicação	Nº de Casos de Teste	Nº de Eventos sob Teste	Script de Teste LOC	Média de LOC por Caso de Teste
Fresh Food Finder	3	20	5.600	233
Pedido App	3	16	5.134	214
MemesPlay	3	12	3.849	160
Agenda	3	18	5.432	226
TodoListCordova	3	10	3.649	152
MovieApp	3	10	3.435	143
TodoList	3	13	4.136	172
Tasky	3	9	3.153	131
Ofertados	2	10	3.248	203

a) QP1 – Efetividade

Visão Geral. Os parágrafos a seguir apresentam os resultados obtidos para a QP1 sem agrupá-los, de forma que os *frameworks* de desenvolvimento e plataformas móveis não foram analisados particularmente.

Foi observado a aplicabilidade das estratégias de localização (expressões individuais e expressões combinadas). A *taxa de aplicabilidade* é definida como o número de eventos que dada estratégia pode ser usada para selecionar um elemento de UI divido pelo o número total de eventos. Conforme discutido na Seção 3.3, algumas estratégias podem não ser usadas em alguns contextos, por exemplo, quando o elemento de UI não tem nenhum dos atributos chave. Como essa informação é útil para o testador quando o caso de teste está sendo elaborado, a relação de aplicabilidade foi medida para as C_{Refs}. A aplicabilidade pode indicar se realmente uma aplicação pode ser testada em outras plataformas, uma vez que quanto menor o número de estratégias aplicáveis, menor pode ser o sucesso de execução do *script* de teste nas múltiplas configurações. A Figura 22 ilustra a taxa de aplicabilidade quando o caso de teste é gravado pela `x-PATeSCO` nas C_{Refs}.

As estratégias individuais baseadas em caminho absoluto (*AbsolutePath* e *AncestorIndex*) obtiveram 100% de aplicabilidade em todas as aplicações. Isso era esperado, uma vez que as expressões absolutas mapeiam o caminho exato do elemento na estrutura XML e não dependem de partes muitas vezes opcionais (por exemplos, atributos). Para as estratégias individuais baseadas em atributos (*CrossPlatform*, *AncestorAttributes* e *IdentifyAttributes*), essa aplicabilidade foi inferior. Dentre elas, a estratégia *IdentifyAttributes* apresentou a menor aplicabilidade, sendo útil em 60,1% dos eventos. As estratégias combinadas usam as seis expressões individuais. *ExpressionInOrder* e *ExpressionMultiLocator* tiveram uma taxa de aplicabilidade de 100%. Resultado também esperado ao considerar que as expressões absolutas

estão contidas no mecanismo desse tipo de estratégia.

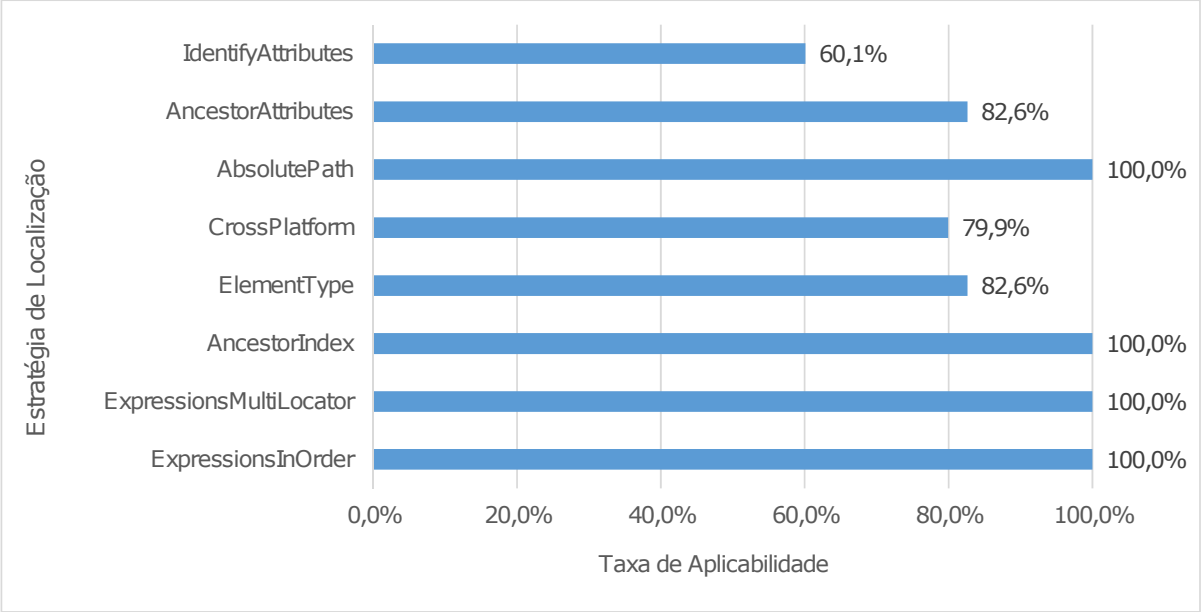

Figura 22: Taxa de aplicabilidade nas C_{Refs}.

Enquanto a aplicabilidade mostra quando dada estratégia pode ser usada, é importante analisar o quanto essa estratégia é bem-sucedida para selecionar um elemento de UI na execução do *script* de teste. A *taxa de executabilidade* é definida como a capacidade para selecionar um elemento de UI com êxito quando um evento é testado. Após a execução de todos os *scripts*, estratégias e eventos nas seis configurações, a taxa de executabilidade das estratégias foi calculada e analisada sob duas óticas, (i) apenas eventos aplicáveis (com base na taxa de aplicabilidade, Figura 22) e (ii) todos os eventos disponíveis. A Tabela 10 resume essa taxa e mostra o resultado para cada configuração. A Coluna "nº de Eventos Aplicáveis" apresenta o número de eventos aplicáveis para cada estratégia nas duas configurações de referência. Em específico para a configuração IOS7-1-2-IPhone, a Coluna "eventos ajustados" foi incluída para corrigir o número de eventos aplicáveis para três aplicações (MovieApp, Ofertados e TodoList) não compatíveis com essa versão do iOS. Para cada configuração foi mostrado o número de eventos executados com sucesso ("EES"), a taxa de executabilidade do evento relativo ao número de eventos aplicáveis ("% Executabilidade"), e a taxa de executabilidade do evento relativo ao número total de eventos ("% Total"). As duas últimas Colunas resumem a executabilidade em todas as configurações, enquanto a duas últimas linhas mostram, por configuração, o número total de eventos e a média da taxa de executabilidade dos eventos nas configurações, respectivamente.

Para cada configuração, 118 eventos estão disponíveis para execução, exceto para a configuração IOS7-1-2-IPhone com 85 eventos. As configurações com versões mais antigas em

Tabela 10: Taxa de executabilidade dos eventos.

	Nº de Eventos Aplicáveis		Android4-4-TabletSamsung			Android5-1-MotoG1			Android6-0-1-MotoG4			IOS7-1-2-iPhone				IOS9-3-iPad			IOS10-2-iPad4			RESUMO	
Estratégias de Localização	CRef1	CRef2	EES*	% Executa-bilidade	% Total	EES*	% Executa-bilidade	% Total	EES*	% Executa-bilidade	% Total	EES*	**Eventos Ajustados	% Executa-bilidade	% Total	EES*	% Executa-bilidade	% Total	EES*	% Executa-bilidade	% Total	% Executa-bilidade	% Total
ExpressionsInOrder	118	118	29	24,6	24,6	99	83,9	83,9	107	90,7	90,7	39	85	45,9	45,9	118	100,0	100,0	82	69,5	69,5	70,2	70,2
ExpressionsMultiLocator	118	118	29	24,6	24,6	99	83,9	83,9	107	90,7	90,7	39	85	45,9	45,9	118	100,0	100,0	82	69,5	69,5	70,2	70,2
AbsolutePath	118	118	0	0,0	0,0	76	64,4	64,4	117	99,2	99,2	14	85	16,5	16,5	118	100,0	100,0	30	25,4	25,4	52,6	52,6
AncestorIndex	118	118	31	26,3	26,3	108	91,5	91,5	117	99,2	99,2	6	85	7,1	7,1	106	89,8	89,8	35	29,7	29,7	59,7	59,7
ElementType	99	101	29	29,3	24,6	89	89,9	75,4	95	96,0	80,5	14	73	19,2	16,5	93	92,1	78,8	64	63,4	54,2	67,1	56,9
CrossPlatform	89	97	16	18,0	13,6	64	71,9	54,2	80	89,9	67,8	36	85	42,4	42,4	95	97,9	80,5	73	75,3	61,9	66,7	53,9
AncestorAttributes	99	101	17	17,2	14,4	81	81,8	68,6	86	86,9	72,9	18	73	24,7	21,2	97	96,0	82,2	53	52,5	44,9	61,5	52,1
IdentifyAttributes	63	86	12	19,0	10,2	47	74,6	39,8	47	74,6	39,8	29	63	46,0	34,1	80	93,0	67,8	45	52,3	38,1	61,3	38,5
Nª de Eventos disponíveis por Config.			118			118			118			85				118			118				
Média			19,9%			80,2%			90,9%			30,9%				96,1%			54,7%				

**EES: Eventos executados com sucesso* ***Eventos Ajustados: Nº de eventos aplicáveis para IOS7-1-2 iPhone*

relação a C_{Refs} (Android4-4-TabletSamsung e IOS7-1-2-IPhone) apresentaram a menor taxa de executabilidade. Essa é uma particularidade evidenciada por estratégias de caminho absoluto (*AbsolutePath* e *AncestorIndex*). Nesse caso é notado que a estrutura XML da UI das aplicações nessas configurações é muito diferente das referências.

Algumas características específicas de duas aplicações afetaram os resultados. A taxa de 99,2% para *AbsolutePath* no Android6-0-1-MotoG4, e não 100% como no IOS9-3-iPad, foi observada. Como o Android6-0-1-MotoG4 é uma das C_{Refs} e *AbsolutePath* é sempre aplicável, esse resultado era esperado. Tal questão ocorreu na aplicação MovieApp: o conteúdo da última tela sob teste é dinâmico, alterando a estrutura XML de referência da qual o caso de teste foi gravado. Isso quebrou a expressão de localização e nenhum elemento de UI foi selecionado. Isso também afetou a executabilidade das estratégias combinadas. Também no Android6-0-1-MotoG4, a taxa de 90,7% (abaixo de 100%) foi observado para as estratégias combinadas. Além da questão da MovieApp, a aplicação ToDoList apresentou um comportamento que também afetou a executabilidade. Um de seus elementos nativos (uma barra de menu) tem uma estrutura diferente para cada plataforma. No Android, os valores dos atributos do menu colidiram com valores de atributos de outro elemento dentro da mesma UI. Isso afetou algumas expressões utilizadas pelas estratégias, pois dois elementos eram retornados pela mesma expressão, dessa forma, interrompendo a execução do *script*. No iOS, essa colisão não ocorreu e as estratégias combinadas obtiveram 100% de executabilidade.

Em relação às estratégias individuais, *ElementType* teve o melhor resultado com 67,1% de taxa de executabilidade (para eventos aplicáveis), seguido pela *CrossPlatform* (66,7%), *AncestorAttributes* (61,5%), *IdentifyAttributes* (61,3%) e *AncestorIndex* (59,7%). A *AbsolutePath* apresentou a pior taxa de executabilidade de eventos com apenas 52,6%. A última coluna compara com o número geral de eventos; *AncestorIndex* foi a melhor (59,7%) pois é sempre aplicável e *IdentifyAttributes* foi a pior (38,5%) devido sua baixa taxa de aplicabilidade.

Em relação às estratégias combinadas, *ExpressionsInOrder* e *ExpressionsMultiLocator* foram melhores do que as individuais (ambas com 70,2%). Sua melhoria varia de 3,1% (em relação a *ElementType*) a 17,2% (em relação a *AbsolutePath*). Analisando a última coluna, sua executabilidade é ainda melhor com melhorias de 10,5% (em relação a *AncestorIndex*) a 31,7% (em relação a *IdentifyAttributes*).

A partir desse ponto a taxa de executabilidade é analisada da perspectiva do caso de teste. Um caso de teste foi considerado quando todos os seus eventos puderam ser executados usando uma determinada estratégia. A Tabela 11 resume a taxa de executabilidade do caso de teste e mostra os resultados para cada configuração. Semelhante à Tabela 10, a única coluna modificada é a primeira para cada configuração, chamada de casos de teste com sucesso (CTS). Ela mede quantos casos de teste foram executados com sucesso (todos os eventos executados com sucesso). Para cada configuração, 26 casos de teste estão disponíveis, exceto para configuração IOS7-1-2-iPhone com 18 casos de teste. A Coluna "número de casos de teste aplicáveis" resume quantos casos de teste são aplicáveis em relação a sua configuração de referência. Por exemplo, a estratégia *ElementType* teve 17 casos de teste para a C_{Ref1} (Android) e 15 casos de teste para a C_{Ref2} (iOS).

Tabela 11: Taxa executabilidade dos casos de teste.

	Nº de Casos de Teste Aplicáveis		Android4-4-TabletSamsung			Android5-1-MotoG1			Android6-0-1-MotoG4			IOS7-1-2-iPhone				IOS9-3-iPad			IOS10-2-iPad4			RESUMO	
Estratégias de Localização	CRef1	CRef2	CTS*	% aplicável	% Total	CTS*	% aplicável	% Total	CTS*	% aplicável	% Total	CTS*	**CTs ajustados	% aplicável	% Total	CTS*	% aplicável	% Total	CTS*	% aplicável	% Total	% aplicável	% Total
ExpressionsInOrder	26	26	7	26,9	26,9	21	80,8	80,8	22	84,6	84,6	6	18	33,3	33,3	26	100,0	100,0	15	57,7	57,7	65,5	65,5
ExpressionsMultiLocator	26	26	7	26,9	26,9	21	80,8	80,8	22	84,6	84,6	6	18	33,3	33,3	26	100,0	100,0	15	57,7	57,7	65,5	65,5
AbsolutePath	26	26	0	0,0	0,0	15	57,7	57,7	25	96,2	96,2	3	18	16,7	16,7	26	100,0	100,0	4	15,4	15,4	49,3	49,3
AncestorIndex	26	26	7	26,9	26,9	23	88,5	88,5	25	96,2	96,2	0	18	0,0	0,0	20	76,9	76,9	5	19,2	19,2	54,1	54,1
ElementType	17	15	7	41,2	26,9	13	76,5	50,0	13	76,5	50,0	3	18	16,7	16,7	13	86,7	50,0	9	60,0	34,6	58,6	39,2
CrossPlatform	10	16	3	30,0	11,5	7	70,0	26,9	7	70,0	26,9	5	12	41,7	27,8	12	75,0	46,2	9	56,3	34,6	58,1	29,1
AncestorAttributes	17	17	4	23,5	15,4	11	64,7	42,3	11	64,7	42,3	4	13	30,8	22,2	14	82,4	53,8	6	35,3	23,1	51,0	33,8
IdentifyAttributes	4	6	3	75,0	11,5	4	100,0	15,4	4	100,0	15,4	3	4	75,0	16,7	4	66,7	15,4	3	50,0	11,5	75,0	14,2
Nº de Casos de Teste disponíveis por Config.			26			26			26			18				26			26				

**CTS: Nº de Casos de Teste com Sucesso* ***CTs Ajustados: Nº de Casos Casos de Teste para IOS7-1-2 iPhone*

Dentre as estratégias individuais, a *IdentifyAttributes* teve o melhor resultado com 75% de taxa de executabilidade (para casos de teste aplicáveis), seguido de *ElementType* (58,6%), *CrossPlatform* (58,1%), *AncestorIndex* (54,1%) e *AncestorAttributes* (51%). A *AbsolutePath* teve a pior taxa de executabilidade para casos de teste com somente 49,3%. A última Coluna relaciona com o número total de casos de teste; *AncestorIndex* foi a melhor (54,1%) quando aplicável, e *IdentifyAttributes* foi a pior (14,2%) devido a sua baixa taxa de aplicabilidade. Relativo as estratégias combinadas, *ExpressionsInOrder* e *ExpressionsMultiLocator* (ambas com 65,5%) ficaram atrás da *IdentifyAttributes* (com 75%). No entanto, elas superaram as outras expressões individuais, variando de 6,9% (em relação a *ElementType*) a 16,2% (em relação a *AbsolutePath*). Ao analisar a última Coluna, a taxa de executabilidade do caso de teste foi

melhor, com melhorias variando de 11,4% (em relação a *AncestorIndex*) a 51,3% (em relação a *IdentifyAttributes*).

As estratégias baseadas em caminhos absolutos somente mantiveram-se altamente aplicáveis nas configurações de referência. Na análise geral, a taxa de executabilidade das estratégias individuais que usam expressões baseadas em atributos é melhor do que as baseadas em caminho absoluto. Entretanto, as estratégias combinadas foram melhores do que as individuais. Da perspectiva do caso de teste, as estratégias combinadas *ExpressionsInOrder* e *ExpressionsMultiLocator* ficaram apenas atrás da *IdentifyAttributes* quando aplicável.

Framework de Desenvolvimento. Os parágrafos a seguir agrupam os resultados obtidos sobre a executabilidade dos eventos e casos de teste na perspectiva do *framework* de desenvolvimento.

Foi verificado a influência do *framework* na executabilidade dos eventos e casos de teste. A Tabela 12 apresenta a taxa de executabilidade das estratégias agrupadas por *framework*. A primeira Coluna ("% EES") apresenta o percentual de sucesso por *framework* (execução de um evento); por exemplo em *ExpressionsInOrder*, 70,2% no *framework* Cordova. Enquanto a segunda Coluna ("% CTS") apresenta o percentual de sucesso referente aos casos de teste executados, 48,3%. Na média geral, o Xamarin obteve a maior taxa de executabilidade para eventos e casos teste, 65,6% e 42,6%, respectivamente, em seguida, o Cordova. O React Native obteve a menor taxa. Ao analisar as estratégias individuais, *AncestorIndex* obteve a melhor taxa de executabilidade para eventos no Cordova e React Native. No entanto, *ElementType* foi a melhor no Xamarin, variando de 4,7% (em relação a *AncestorIndex* - Cordova) a 8,3% (em relação a *AncestorIndex* - React Native). *IdentifyAttributes* foi a pior em executabilidade de eventos no Cordova e React Native, mas no Xamarin ela se manteve uma posição acima da pior (*AbsolutePath*). No ponto de vista dos casos de teste executados com sucesso, as melhores estratégias individuais são diferentes entre os três *frameworks*, porém a *IdentifyAttributes* foi a pior em todos. As estratégias combinadas atingiram as melhores taxas; no geral, a variação da executabilidade entre elas no mesmo *framework* foi baixa. Em sua análise deve ser considerado que as aplicações possuem diferentes implementações, as quais impactam na aplicabilidade das expressões, dessa forma não sendo possível uma análise comparativa mais extensa entre os *frameworks*.

Tabela 12: Taxa executabilidade relativa aos *frameworks*.

	Cordova		React Native		Xamarin	
Estratégias de Localização	% EES*	% CTS*	% EES*	% CTS*	% EES*	% CTS*
ExpressionsInOrder	70,2%	48,3%	49,6%	27,1%	93,3%	65,0%
ExpressionsMultiLocator	70,2%	48,3%	50,4%	29,2%	93,1%	62,5%
AbsolutePath	57,2%	40,8%	42,7%	22,9%	48,0%	32,5%
AncestorIndex	59,0%	39,2%	62,6%	37,5%	59,6%	37,5%
ElementType	54,6%	22,5%	58,3%	27,1%	67,3%	45,5%
CrossPlatform	57,2%	18,3%	40,9%	18,8%	53,8%	30,0%
AncestorAttributes	53,7%	21,7%	41,8%	18,8%	58,7%	37,5%
IdentifyAttributes	40,8%	5,8%	18,3%	4,2%	51,0%	30,0%
% Média	57,9%	30,6%	45,6%	23,2%	65,6%	42,6%

% EES: % Eventos Executados com Sucesso* *% CTS: % Casos de Teste Executados com Sucesso*

O Xamarin alcançou a maior taxa de executabilidade e o React Native a menor taxa. A estratégia *IdentifyAttributes* foi a pior em executabilidade no Cordova e React Native. No entanto, no Xamarin a pior foi *AbsolutePath*. As estratégias combinadas obtiveram taxas semelhantes entre elas no mesmo *framework*. Além disso, mantiveram as maiores taxas de executabilidade para eventos e casos de teste.

Plataforma Móvel. Nos parágrafos a seguir os resultados sobre a executabilidade de eventos e casos de teste são agrupados sob ponto de vista das plataformas móveis Android e iOS.

A Figura 23 ilustra a taxa de executabilidade entre plataformas. O Android teve a melhor taxa de executabilidade em ambos níveis, eventos (55,9%) e casos de teste (45,2%). A diferença entre as plataformas para taxa de executabilidade de eventos foi 3,3% enquanto para casos de teste, 5,1%. Esses dados sugerem uma pequena e melhor compatibilidade da plataforma Android em manter o padrão da estrutura XML da UI em suas versões. Ao considerar que a efetividade do *script* de teste está relacionada a execução de casos de testes com sucesso em ambas plataformas, a efetividade média do *script* gerado pela `x-PATeSCO` foi de 42,6%.

Em comparação com a plataforma iOS, o Android obteve uma pequena e superior diferença na taxa de executabilidade em nível de eventos e casos de teste.

b) QP2 – Tempo de Execução

Analisou-se o custo de cada estratégia de localização ao verificar o tempo de execução da CPU. Para isso, não foi considerado os eventos nos quais as estratégias falharam ao localizar o elemento de UI.

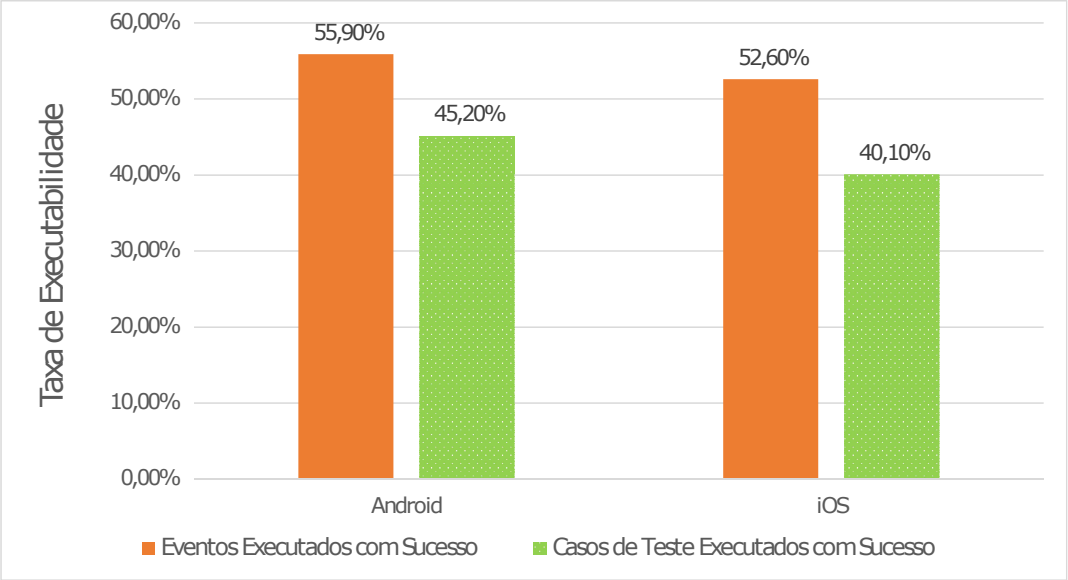

Figura 23: Taxa de executabilidade nas plataformas Android e iOS.

Visão Geral. A seguir os resultados extraídos da execução do experimento para responder a QP2 são apresentados de maneira geral, sem agrupar os demais pontos de vista.

A Tabela 13 apresenta a média de CPU (em segundos) dos eventos executados com sucesso por estratégia e plataforma. Em relação as estratégias individuais, a *IdentifyAttributes* foi a mais rápida (13,2s), seguida pela *CrossPlatform* (14,9s), *ElementType* (16,4s) e *AncestorAttributes* (16,4s). Estratégias baseadas em caminho absoluto (*AncestorIndex* e *AbsolutePath*) foram as mais lentas (24,4s e 25,1s, respetivamente). Quanto às estratégias combinadas, *ExpressionsInOrder* e *ExpressionsMultiLocator* foram mais lentas que as estratégias individuais com 26,9s e 166,7s, respectivamente. Enquanto *ExpressionsInOrder* está perto das estratégias individuais mais lentas (entre 24-25s), o tempo de execução da *ExpressionsMultiLocator* é aproximadamente seis vezes mais lento. Isto era esperado, pois seu uso exige a execução das seis expressões individuais mais o cálculo de votação.

Tabela 13: Tempo médio em segundos da execução dos eventos com sucesso.

Estratégias de Localização	Android4-4-TabletSamsung	Android5-1-MotoG1	Android6-0-1-MotoG4	IOS7-1-2-iPhone	IOS9-3-iPad	IOS10-2-iPad4	MÉDIA GERAL	MÉDIA POR PLATAFORMA	
								Android	iOS
ExpressionsInOrder	10,93	14,27	13,60	14,67	59,94	47,92	26,9	38,80	122,53
ExpressionsMultiLocator	14,49	29,90	34,67	151,63	495,28	274,13	166,7	79,06	921,04
AbsolutePath	0,00	15,48	15,34	12,15	50,83	56,62	25,1	30,81	119,60
AncestorAttributes	5,30	8,74	7,58	15,61	33,40	27,68	16,4	21,62	76,69
AncestorIndex	15,33	16,97	15,83	7,89	52,18	38,25	24,4	48,13	98,32
CrossPlatform	3,22	6,86	6,09	13,30	30,64	29,34	14,9	16,17	73,28
ElementType	10,03	9,95	9,05	12,12	30,80	26,34	16,4	29,03	69,25
IdentifyAttributes	7,24	10,67	10,86	11,88	20,49	18,22	13,2	28,78	50,60
						Total em Segundos:		292,41	1531,32
						Total em Minutos:		4,87	25,52

A Figura 24 sumariza graficamente a distribuição do tempo de execução das estratégias em gráficos *BoxPlot*. No geral, os dados analisados são assimétricos e não apresentam valores discrepantes *(outliers)* em sua distribuição. As estratégias individuais compartilham o mesmo mecanismo de execução das expressões; sua respectiva expressão XPath é executada individualmente sem interferência das demais; dentre elas as estratégias baseadas em atributos chave *AncestorAttributes, CrossPlatform, ElementType e IdentifyAttributes* apresentaram a menor variabilidade do tempo de execução dos eventos. A variabilidade entre as individuais baseadas em caminho absoluto é mais alta *(AbsolutePath e AncestorIndex)*. Nas estratégias combinadas a variabilidade do tempo de execução é alta, principalmente com a *ExpressionsMultiLocator*. A estratégia *ExpressionsInOrder* apresentou uma variabilidade próxima da estratégia individual *AbsolutePath*. Na maioria das expressões o quartil superior está próximo da linha do máximo valor devido as execuções com tempo médio mais alto oriundas dos dispositivos iOS. Por exemplo, *AbsolutePath* é aproximadamente quatro vezes mais lenta no iOS.

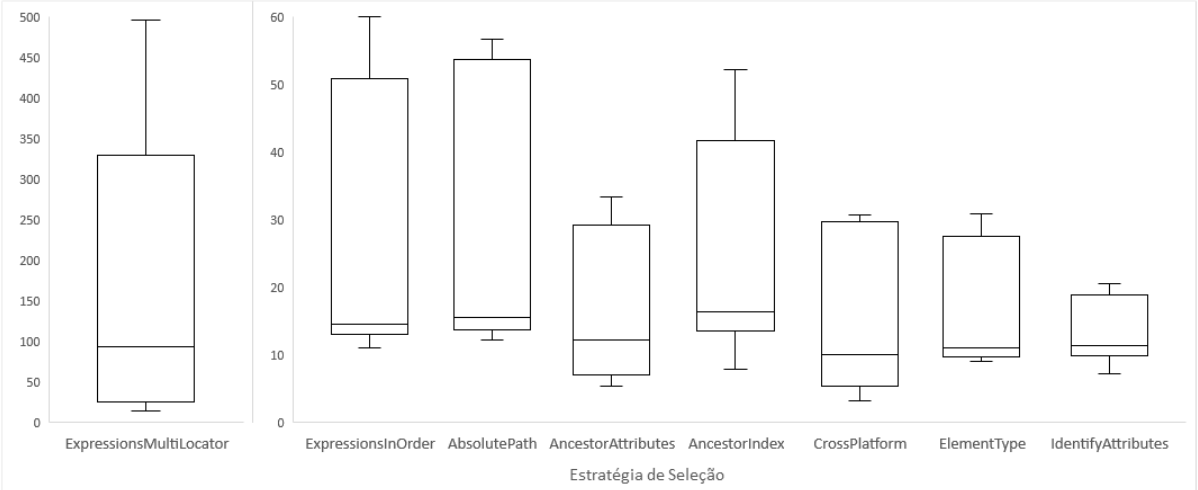

Figura 24: BoxPlots representando a variabilidade do tempo de execução em segundos das estratégias.

As estratégias baseadas em atributos são mais rápidas, seguida das estratégias baseadas em caminho absoluto. As estratégias combinadas têm o pior tempo de execução: *ExpressionsInOrder* está próximo da mais lenta estratégia individual e seis vezes mais rápida do que *ExpressionsMultiLocator*.

Framework de Desenvolvimento. Os parágrafos a seguir apresentam a análise do tempo de execução das estratégias a partir da perspectiva dos *frameworks* de desenvolvimento.

A Tabela 14 apresenta o tempo médio em segundos da execução com sucesso

dos eventos e casos de teste em diferentes aplicações, com diferentes implementações e funcionalidades. Nesse caso é considerado o *framework* de desenvolvimento móvel. O Xamarin obteve o menor tempo de execução para eventos (8,19s) e casos de teste (28,85s), seguido do React Native. O Cordova apresentou o maior tempo para execução dos eventos e casos de teste. Os resultados gerais apresentados anteriormente também se mantiveram na perspectiva do *framework*; a estratégia individual *IdentifyAttributes* é a mais rápida e a *AbsolutePath* é a mais lenta. A estratégia combinada *ExpressionsInOrder* manteve-se como a mais rápida em todos os *frameworks* em comparação com *ExpressionMultiLocator*.

Tabela 14: Tempo médio em segundos da execução dos eventos e casos de teste com sucesso em relação aos *frameworks*.

	Cordova		React Native		Xamarin	
Estratégias de Localização	Evento	Caso de Teste	Evento	Caso de Teste	Evento	Caso de Teste
ExpressionsInOrder	8,50	41,66	9,04	34,81	6,35	22,98
ExpressionsMultiLocator	56,66	265,63	51,18	192,98	30,73	111,65
AbsolutePath	6,98	34,20	7,84	30,53	6,11	22,07
AncestorIndex	6,81	32,17	6,90	26,86	5,60	21,75
ElementType	5,67	30,29	5,76	13,72	4,52	14,69
CrossPlatform	6,05	26,77	6,09	17,50	4,12	14,56
AncestorAttributes	5,96	33,88	6,21	17,43	4,08	13,40
IdentifyAttributes	5,86	21,92	7,57	21,69	4,02	9,66
Média	12,81	60,81	12,58	44,44	8,19	28,85

O Xamarin foi o *framework* que obteve o menor tempo de execução para eventos e casos de teste, seguido do React Native, enquanto o Cordova foi o mais lento.

Plataforma Móvel. Os parágrafos a seguir relatam os resultados para o tempo de execução das estratégias sob o ponto de vista das plataformas móveis.

As duas últimas colunas da Tabela 13 resumem o tempo médio de execução dos eventos por plataforma. A plataforma Android foi aproximadamente 5 vezes mais rápida (4,87 minutos) comparada à iOS (25,52 minutos). Na plataforma Android a estratégia individual *CrossPlatform* foi a mais rápida e *AncestorIndex* a mais lenta. No iOS a mais rápida foi *IdentifyAttributes* e *AbsolutePath* a mais lenta. Entre as combinadas a estratégia *ExpressionsMultiLocator* foi a mais lenta em ambas plataformas.

Os eventos executados na plataforma Android foram aproximadamente 5 vezes mais rápidos em comparação com iOS. Destaque para as estratégias *CrossPlatform* no Android e *IdentifyAttributes* no iOS.

4.5 DISCUSSÃO DOS RESULTADOS

A avaliação experimental relatada avaliou a abordagem e a ferramenta propostas para definição de um mecanismo único de teste para múltiplas configurações. Os resultados obtidos fornecem evidências acerca da viabilidade, aplicabilidade e efetividade da abordagem e ferramenta apresentadas.

No teste de aplicações multiplataforma é essencial cobrir o maior número de dispositivos existentes no mercado. Entretanto, manter um *script* de teste para cada configuração disponível pode elevar o custo do projeto. Isso motiva serviços em nuvem *(cloud services)*, como Amazon Device Farm e BitBar, a oferecer recursos de teste em centenas de dispositivos. Infelizmente, cada plataforma móvel tem sua própria representação de UI, que também varia entre diferentes versões do mesmo SO, o que dificulta a definição de um *script* único para teste em vários dispositivos. A abordagem proposta visa lançar alguma luz sobre como automatizar tais testes em múltiplas configurações. Consequentemente, investigou e comparou estratégias para localizar elementos de UI. A seguir, os resultados obtidos são discutidos.

Cada versão de plataforma apresenta mudança na organização da estrutura XML da UI, o que pode impactar na manutenibilidade das expressões absolutas. Um elemento de UI pode ser representado diferente nas múltiplas configurações. Os atributos dos elementos possibilitam a criação de expressões baseadas em seus valores, o que ajuda a garantir uma melhor executabilidade no teste de múltiplas configurações devido a preservação dos valores entre as diferentes configurações. Portanto expressões baseadas em atributos são mais robustas. O Xamarin e React Native são *frameworks* que trabalham com mecanismos de mapeamento de elemento de UI nativos. O desenvolvedor utiliza um elemento de UI disponível pelo *framework* e na execução da aplicação na plataforma alvo esse elemento é mapeado para um elemento nativo. Foi identificado que tal mapeamento difere na aparência visual do elemento (como o usuário da aplicação vê o elemento). O *framework* pode inserir o elemento na tela de acordo com o guia de recomendações de UI da plataforma. Por exemplo, uma barra de menu no Android posicionada ao topo da tela é no iOS posicionada na parte inferior da tela. Essa diferença na representação visual do elemento também reflete na estrutura XML da UI, pois os elementos apresentam tipos, atributos e valores de atributos diferentes, o que impacta na construção das XPath compatibilizadas. Essa situação foi observada para duas aplicações nativa-multiplataforma, uma construída com o Xamarin e a outra com o React Native.

A melhor executabilidade das estratégias combinadas é devido ao seu mecanismo

interno que pode executar as seis expressões individuais quando necessário. Entre as estratégias individuais que usam atributos, o uso do atributo identificador é sempre recomendado por possibilitar localizar um elemento com exatidão e menor tempo dentro do XML. No entanto, ele nem sempre está disponível. Na visão geral, *IdentifyAttributes* foi a pior estratégia devido a falta do atributo nos elementos selecionados para o teste.

Como já discutido, o mecanismo de localização de elementos utilizado pelas estratégias combinadas pode executar até todas as seis expressões individuais, por isso são mais lentas. Na prática o tempo é um fator importante. Testes que executam em menor tempo garantem a descoberta de defeitos e/ou liberação da aplicação para os usuários mais rapidamente. Além disso, minimiza os custos quando os testes são realizados por serviços em nuvem, tal como o TestObject que limita o uso de dispositivos por minutos. Na perspectiva da plataforma móvel, os testes executados no Android foram aproximadamente cincos vezes mais rápidos comparados aos testes na plataforma iOS. A média de execução de eventos na iOS foi 25,52 minutos. No cenário do experimento foi utilizado um outro dispositivo com o MAC OS para executar o Appium como servidor e realizar a comunicação com os dispositivos da Apple, requisito para o teste na plataforma iOS. Essa condição pode ter influenciado no tempo de execução do teste na plataforma iOS devido a necessidade de dois computadores; um computador executando o projeto de teste sob o Windows e outro executando o Appium no MAC OS para conexão aos dispositivos móveis. A comunicação entre os computadores ocorreu via rede local. No entanto, relatos na comunidade do Appium apontam questões referente à lentidão nessa plataforma e corroboram os resultados obtidos.

A expressão combinada *ExpressionsInOrder* apresentou entre todas outras avaliadas a melhor taxa de executabilidade com tempo de execução próximo das estratégias individuais, embora ainda há espaço para melhorias. Com base nos resultados apresentados e com o objetivo de aumentar a sua cobertura da executabilidade de eventos e casos de teste, novas ordens de execução das expressões individuais podem ser propostas e avaliadas. Além disso, recomenda-se que os programadores devam priorizar sempre que possível o uso de atributos identificadores durante a implementação da UI da aplicação, melhorando assim o desempenho em nível de tempo de execução do teste e aplicabilidade em localizar um elemento usando a estratégia *ExpressionsInOrder*. No entanto, é conhecido que alguns *frameworks* podem mudar dinamicamente o valor desses atributos, falhando a expressão. Assim, sugere-se na sequência de expressões interna usadas por essa estratégia o uso das expressões baseadas nos atributos chave. Portanto, recomenda-se que *ExpressionsInOrder* deve ser usada como uma estratégia para localizar elementos de UI em teste de aplicações multiplataforma.

Na perspectiva do *script* de teste gerado pela x-PATeSCO, sua efetividade média para executar um caso de teste com sucesso nas múltiplas configurações foi de aproximadamente 42%. Sua executabilidade pode ser melhorada se durante o desenvolvimento da aplicação móvel o uso de atributos nos elementos de UI forem constantes, assim possibilitando a criação de expressões robustas para localização do elemento usado no teste. Acredita-se que a aplicação deva ser construída com subsídios necessários para que o teste automatizado de múltiplas configurações possa ser aplicado.

Na perspectiva dos *frameworks* o Xamarin apresentou os melhores resultados para executabilidade (Tabela 12) e tempo médio de execução (Tabela 14). A arquitetura do React Native pode ser menos efetivo devido a limitações de desempenho do Javascript e à sobrecarga causada pelo mecanismo que invoca (ponte) as APIs da plataforma (REACTXP, 2017a). Ao incluir o Cordova no comparativo, ele apresentou os piores resultados para tempo de execução. Esse fato pode ser justificado ao considerar que a UI da aplicação é construída com tecnologias da Web (HTML e CSS), e que em sua execução ela é transformada em XML pela plataforma, impactando no tempo de execução. Além disso, uma aplicação híbrida é executada em uma camada adicional chamada *WebView*. No caso dos *frameworks* nativo-multiplataforma, como o Xamarin e React Native utilizam a UI nativa da plataforma móvel alvo o que assegura maior rapidez na execução da aplicação. Contudo, como cada aplicação implementa funcionalidades diferentes, quais podem consumir serviços externos ou internos, por exemplo acesso a Internet ou processamento local. Portanto, comparar o tempo de execução é apenas um indício e não se pode concluir muito sobre a influência do *framework* no tempo de execução da aplicação.

Espera-se que a abordagem, x-PATeSCO, e os resultados obtidos contribuam para futuros avanços no teste de aplicações móveis.

4.6 LIMITAÇÕES E AMEAÇAS A VALIDADE

Esta seção introduz possíveis ameaças à validade desta pesquisa.

Primeiro, a adoção de um número limitado de configurações (dispositivos) e aplicações reduzem a generalização dos resultados obtidos. Quanto ao número de configurações, existe uma enorme quantidade de dispositivos no mercado e o custo de uma investigação abrangente seria proibitivo. Optou-se por variar as versões do SO, misturando *smartphones* e *tablets* com diversas configurações. Quanto às aplicações, a necessidade do código-fonte do projeto foi uma restrição para a plataforma iOS; isso limitou as opções. Para reduzir essa ameaça, foi selecionado não só projetos de código aberto, mas também dois projetos industriais. Além

disso, aplicações desenvolvidas com diferentes *frameworks* de desenvolvimento também foram selecionadas.

Outra ameaça é a seleção das configurações de referência. Como as estratégias de localização são projetadas com base nelas, outros dispositivos de referência podem produzir resultados diferentes. Acredita-se que essa seleção seria normalmente baseada na popularidade; assim, a decisão foi baseada em estatísticas conhecidas sobre o uso dos usuários de versões da plataforma.

Para reduzir o viés nos tipos e na estrutura dos testes, quatro participantes independentes com experiência em computação móvel projetaram os casos de teste. Juntamente com a diversidade do domínio das aplicações, tamanho do projeto e *framework* de desenvolvimento, visou-se cobrir as principais características que influenciam o teste automatizado para aplicações móveis multiplataforma.

É reconhecido que a maioria das ameaças listadas podem ser superadas no futuro com replicações. Assim, a ferramenta e os objetos do experimento estão disponíveis como um pacote experimental de código aberto disponível em um repositório no GITHUB (MENEGASSI; ENDO, 2018).

4.7 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado um estudo experimental para avaliar a abordagem e a ferramenta propostas para gerar *scripts* de teste único para aplicações móveis multiplataforma. Os objetos do experimento foram nove aplicações móveis construídas com *frameworks* multiplataforma (Cordova, Xamarin e React Native) obtidos com empresas parceiras e em repositórios no GITHUB. Quatro participantes independentes participaram do experimento projetando os casos de teste das aplicações. Os projetos de teste das aplicações foram gerados a partir da ferramenta originada da abordagem. Durante a execução dos projetos, dados foram coletados atendendo às métricas das duas questões de pesquisa respondidas.

Os dados coletados na avaliação demonstram que a abordagem e a ferramenta `x-PATeSCO` podem contribuir para o teste em múltiplas configurações. Para uma das estratégias de seleção proposta, foi identificado efetividade e desempenho superior ao compará-la com as demais estratégias.

O capítulo a seguir conclui esta dissertação, sumarizando as principais contribuições, as limitações são revisitadas e os trabalhos futuros são mencionados.

5 CONCLUSÃO

A grande variedade de dispositivos e SO disponíveis no mercado tornam o teste de aplicações móveis multiplataforma um desafio (GRONLI; GHINEA, 2016; BOUSHEHRINEJADMORADI et al., 2015; JOORABCHI et al., 2013). Testar a aplicação em somente um dispositivo não garante o correto funcionamento em outro (NAGAPPAN; SHIHAB, 2016; JOORABCHI et al., 2015), assim cada dispositivo representa uma configuração que precisa ser verificada. Embora o teste automatizado seja essencial para cobrir muitas configurações, os mecanismos atuais de teste não são multiplataforma. Versões diferentes da mesma plataforma apresentam variações na representação da UI (FAZZINI et al., 2017; MENEGASSI; ENDO, 2016); isso demanda a necessidade que dois ou mais *scripts* tenham que ser criados para versões diferentes do mesmo SO.

Nesta dissertação, foi apresentada uma abordagem para gerar *scripts* de teste para aplicações móveis multiplataforma em múltiplas configurações. Foram definidas estratégias de localização de elementos de UI baseadas em expressões XPath. São seis expressões individuais *(AbsolutePath, IdentifyAttributes, AncestorIndex, AncestorAttributes, ElementType e CrossPlatform)* e duas estratégias combinadas *(ExpressionsInOrder e ExpressionsMultiLocator)* foram derivadas delas. A abordagem baseia-se em um dispositivo de referência para cada SO, ou seja, um Android e outro iOS. As estratégias são geradas em duas versões, uma para cada referência. Uma ferramenta chamada `x-PATeSCO` *(cross-Platform App Test Script reCOrder)*, foi desenvolvida para apoiar a abordagem e as estratégias de localização. Nove aplicações móveis multiplataforma instaladas em seis dispositivos reais diferentes, entre eles, *smartphones* e *tablets* com diversas versões do Android e iOS, foram objetos de um estudo experimental realizado para avaliar a abordagem e a ferramenta, bem como comparar as estratégias de localização.

Na avaliação experimental, os resultados mostraram que quatro estratégias *(AbsolutePath, AncestorIndex, ExpressionsInOrder e ExpressionsMultiLocator)* são aplicáveis em todos os eventos sob teste para todas as plataformas usadas no experimento. As estratégias baseadas em atributos são menos aplicáveis: *AncestorAttributes* e *ElementType*

são aplicáveis em 82,6%, *CrossPlatform* em 79,9%, e *IdentifyAttributes* em 60,1%. Nenhuma das estratégias obteve 100% de taxa de executabilidade. No geral, as estratégias combinadas *(ExpressionsInOrder e ExpressionsMultiLocator)* tiveram o melhor resultado, alcançando 70,2% da executabilidade dos eventos e 65,5% de casos de teste executados com sucesso. Quando comparadas com expressões conhecidas e empregadas na prática, como *IdentifyAttributes* e *AbsolutePath*, as estratégias combinadas foram mais efetivas na localização de elementos de UI, superando em executabilidade de eventos para 8,9% e 17,6%, respectivamente. Na executabilidade em nível de *framework* de desenvolvimento, os resultados foram diferentes entre as estratégias; e não evidenciam uma melhor ou pior estratégia entre eles.

Para a executabilidade em nível de caso de teste, *IdentifyAttributes* alcançou 75%, apesar de ser menos aplicável para todos os casos de teste e configurações avaliadas (14,2%).

Referente ao tempo de execução, as estratégias combinadas foram mais lentas do que as estratégias individuais. Dentre as estratégias combinadas *ExpressionsInOrder* foi aproximadamente seis vezes mais rápida do que a *ExpressionsMultiLocator*. Os resultados evidenciam que a *ExpressionsInOrder* é estratégia de localização recomendada para ser aplicada com a abordagem proposta, sendo altamente aplicável e com razoável taxa de executabilidade e tempo. Em nível de plataforma, a execução dos eventos no Android foi aproximadamente 5 vezes mais rápida (4,87 minutos) comparada à iOS (25,52 minutos).

Na perspectiva do *script* de teste gerado pela `x-PATeSCO`, sua efetividade média para executar um caso de teste com sucesso nas múltiplas configurações foi de aproximadamente 42%. A falta de atributos nos elementos de UI comprometem a geração das expressões de localização, diminuindo a efetividade do *script*. Durante sua execução nas diferentes plataformas, os casos de teste executados com sucesso no Android em relação ao iOS foi 5,1% superior.

5.1 LIMITAÇÕES E TRABALHOS FUTUROS

A abordagem proposta nesta dissertação não resolve completamente o problema do teste de aplicações multiplataforma em múltiplas configurações, mas contribui para minimizá-lo e amplia seu entendimento. A falta de compatibilidade entre as estruturas XML de UI é de fato a maior barreira a ser vencida. Acredita-se que os fabricantes das plataformas estão longe para definição de um padrão comum. Portanto, cabe a comunidade técnica e científica a definição de mecanismos que auxiliem aos testadores garantir a qualidade desse tipo de aplicação.

As limitações e trabalhos futuros são apresentados a seguir:

Melhoria das estratégias. Baseados nos resultados fornecidos pela avaliação da abordagem é planejado uma alteração na ordem da sequência das expressões de consulta utilizadas pela estratégia combinada *ExpressionsInOrder*. A nova ordem sugerida inicia-se pela expressão baseada no atributo identificador do elemento seguida por expressões baseadas em atributos chave. No experimento esse tipo de expressão apresentou as melhores taxas de tempo de execução, e quando aplicáveis possuem a melhor taxa de executabilidade ao compará-las com as expressões de caminho absoluto. O objetivo é aumentar a cobertura da executabilidade de eventos e casos de teste com um tempo de execução menor, dessa forma a nova ordem sugerida é: *IdentifyAttributes* → *ElementType* → *AncestorAttributes* → *CrossPlatform* → *AbsolutePath* → *AncestorIndex*. Referente a estratégia *ExpressionsMultiLocator*, o mecanismo de votação utilizado pode obter elementos diferentes e com mesmo peso, portanto, faz-se necessário implementar uma estratégia para o desempate de elementos.

Elementos conflitantes. Os elementos de UI podem possuir atributos com valores iguais aos de outros elementos dentro da mesma UI. A repetição de valores pode interromper a execução do *script* de teste ao selecionar erroneamente um elemento não esperado. É necessário melhorar a geração de XPath para resolver esse tipo de conflito, de tal forma que garanta a localização de somente um elemento.

Novas comparações. Diversas ferramentas possibilitam gerar expressões de localização de elementos, por exemplo a Appium Studio (EXPERITEST, 2017). No futuro faz-se importante comparar as estratégias de localização propostas nesta pesquisa com outras estratégias, e se viável, incluí-las na `x-PATeSCO`.

Script de teste. A manutenção e reparação dos *scripts* de teste na evolução da aplicação é também um tópico importante a investigar no contexto multiplataforma. Outras melhorias para diminuir o tempo de execução dos testes precisam ser investigadas e aplicadas.

Melhorias na ferramenta. A versão atual da `x-PATeSCO` não suporta complexas interações de usuário, como multi-toque e gestos (por exemplo, pinça). A implementação dessas interações esta prevista para uma versão futura. Outra suplementação é a gravação das assertivas pelo testador a partir da ferramenta, e a definição dinâmica (por evento) da ordem das expressões utilizadas pela estratégia *ExpressionsInOrder*.

Novas avaliações experimentais. Outros estudos podem ser realizados com um número maior de aplicações industriais; outra questão a ser vista é qual o comportamento da ferramenta com um extenso conjunto de dispositivos móveis acessíveis através de serviços de nuvem como o Amazon Device farm, BitBar e outros.

Nova abordagem. É necessário investigar se modo de desenvolvimento das aplicações multiplataforma podem influenciar nas atividades de teste. Sugere-se executar o mesmo experimento utilizando aplicações móveis controladas, isso é, desenvolvidas pelos pesquisadores nos diferentes *frameworks*. Os novos resultados podem ajudar na melhoria das atividades de testes ao indicar pontos a serem tratados pelos programadores em nível de código-fonte da aplicação.

Novos tipos de aplicações. Independente do tipo de aplicação (nativa, baseada no navegador de Web, híbrida ou nativas construídas com *frameworks* nativo-multiplataforma) sua UI é transformada em uma estrutura XML durante a execução. A abordagem atual concentrou-se em somente aplicações híbridas e nativas-multiplataforma. No futuro é pretendido estender a abordagem para aplicações totalmente nativas além das baseadas em navegador Web.

5.2 DIVULGAÇÃO DOS RESULTADOS

Durante o desenvolvimento desta dissertação de mestrado, obteve-se a seguinte publicação:

- MENEGASSI, A. A.; ENDO, A. T. **An evaluation of automated tests for hybrid mobile applications.** In: 2016 XLII Latin American Computing Conference (CLEI). 2016. p. 1–11. DOI: 10.1109/CLEI.2016.7833337

Publicação submetida e aguardando parecer:

- MENEGASSI, A. A.; ENDO, A. T. **Automated Tests for Cross-Platform Mobile Apps in Multiple Configurations.** Submetido a uma conferência da área.

Ferramenta desenvolvida e pacote experimental:

- MENEGASSI, A. A.; ENDO, A. T. **x-PATeSCO - Cross-Platform App Test Script Recorder e Pacote Experimental**. Disponível em: https://github.com/andremenegassi/x-patesco

REFERÊNCIAS

ABLESON, W. F. et al. **Android em Ação**. 3ª. ed. Rio de Janeiro: Campus, 2012.

ALASDAIR, A. **Aprendendo Programação - iOS**. São Paulo: Novatec, 2013.

AMALFITANO, D. et al. **Testing Android Mobile Applications: Challenges, Strategies, and Approaches**. Oxford: Elsevier Inc., 2013. 1–52 p. ISSN 00652458. ISBN 9780124080942. Disponível em: <http://dx.doi.org/10.1016/B978-0-12-408094-2.00001-1>.

AMALFITANO, D. et al. MobiGUITAR Automated Model-Based Testing of Mobile Apps. **Software, IEEE**, v. 32, n. 5, p. 53–59, 2015. ISSN 0740-7459. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6786194>.

AMAZON. **AWS Device Farm - Amazon Web Services**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://aws.amazon.com/en/device-farm>.

ANDROID.COM. **The Monkey UI android testing tool**. 2017. Acessado em: 10-Out-2017. Disponível em: <http://developer.android.com/tools/help/monkey.html>.

APPIUM. **APPIUM**. 2017. Acessado em: 06-Out-2017. Disponível em: <http://appium.io/>.

BARBOSA, E. F.; MALDONADO, J. C.; VINCENZI, A. M. R. **Introdução ao Teste de Software**. 2004.

BARR, E. T. et al. The Oracle Problem in Software Testing: A Survey. v. 41, n. 5, p. 1–31, 2015.

BELLI, F.; BUDNIK, C. J.; WHITE, L. Event-based modelling, analysis and testing of user interactions: Approach and case study. **Softw. Test. Verif. Reliab.**, John Wiley and Sons Ltd., Chichester, UK, v. 16, n. 1, p. 3–32, mar. 2006. ISSN 0960-0833. Disponível em: <http://dx.doi.org/10.1002/stvr.v16:1>.

BILGIN, C. **Mastering cross-platform development with Xamarin**, 2016. 390 p.

BITBAR. **Mobile App Testing - Testdroid Technology by Bitbar**. 2017. Acessado em: 06-Out-2017. Disponível em: <http://www.bitbar.com/testing>.

BLOM, S. et al. Write once, run anywhere - A survey of mobile runtime environments. **Proceedings - 3rd International Conference on Grid and Pervasive Computing Symposia/Workshops (GPC 2008)**, p. 132–137, 2008.

BOUSHEHRINEJADMORADI, N. et al. Testing Cross-Platform Mobile App Development Frameworks. In: **Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference**, 2015. p. 441–451.

CAMDEN, R. K. **Apache Cordova in Action**. Shelter Island, NY: Manning Publications Co, 2016.

CHENG, J. et al. Mobile Compatibility Testing Using Multi-objective Genetic Algorithm. **2015 IEEE Symposium on Service-Oriented System Engineering**, p. 302–307, 2015. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7133545>.

CHOUDHARY, S. R.; GORLA, A.; ORSO, A. Automated test input generation for android: Are we there yet? (e). In: **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**, 2015. p. 429–440.

CORDOVA. **Cordova**. 2016. Acessado em: 06-Out-2017. Disponível em: <https://cordova.apache.org/>.

CORRAL, L.; SILLITTI, A.; SUCCI, G. Mobile Multiplatform Development: An Experiment for Performance Analysis. **Procedia Computer Science**, v. 10, p. 736–743, 2012. ISSN 1877-0509. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050912004516>.

COULOURIS, G. et al. **Distributed Systems: Concepts and Design**. 5. ed. USA: Addison-Wesley, 2012. ISBN 978-0-13-214301-1.

DEI, J.; SEN, A. Investigation on Trends of Mobile Operating Systems. **International Journal of Engineering Research & Technology (IJERT)**, v. 4, n. 07,, p. 764–775, 2015.

DEITEL, P.; HARVEY, D.; ALEXANDER, W. **Android 6 para Programadores - Uma abordagem baseada em aplicativos**. Porto Alegre: Bookman, 2016.

DEITEL, P. J.; DEITEL, H. M. **Ajax, Rick Internet Applications e desenvolvimento Web para programadores**. São Paulo: Pearson, 2008.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. Rio de Janeiro: Elsevier, 2007.

DEVELOPERS, G. **Progressive Web Apps - A new way to deliver amazing user experiences on the web.** 2017. Disponível em: <https://developers.google.com/web/progressive-web-apps/?hl=en>.

EISENMAN, B. **Learning React Native**: O'Reilly Media, 2016. ISBN 9781491929001.

ENDO, A. T. **Model based testing of service oriented applications**. Tese (Doutorado) — USP, 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20062013-140259/pt-br.php>.

EXPERITEST. **Appium Studio**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://experitest.com/appium-studio/>.

FARTO, G. de C.; ENDO, A. T. Reuse of model-based tests in mobile apps. In: **Proceedings of the 31st Brazilian Symposium on Software Engineering**. New York, NY, USA: ACM, 2017. (SBES'17), p. 184–193. ISBN 978-1-4503-5326-7. Disponível em: <http://doi.acm.org/10.1145/3131151.3131160>.

FAZZINI, M. et al. Barista: A technique for recording, encoding, and running platform independent android tests. In: **2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)**, 2017. p. 149–160.

FLANAGAN, D. **Javascript: O guia definitivo**. 6ª. ed. Porto Alegre: BOOKMAN, 2013. ISBN 978-85-65837-19-4.

GAO, J. et al. Mobile Application Testing: A Tutorial. **Computer**, v. 47, p. 46–55, 2014. ISSN 0018-9162.

GARTNER. **Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017**. 2017. Acessado em: 31-Out-2017. Disponível em: <https://www.gartner.com/newsroom/id/3725117>.

GOOGLE. **Espresso**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://google.github.io/android-testing-support-library>.

GRONLI, T.-M.; GHINEA, G. 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)**: IEEE, 2016. p. 5711–5720. ISBN 978-0-7695-5670-3.

GUDMUNDSSON, V. et al. Model-based Testing of Mobile Systems – An Empirical Study on QuizUp Android App. **Electronic Proceedings in Theoretical Computer Science**, v. 208, n. PrePost, p. 16–30, 2016. ISSN 2075-2180. Disponível em: <http://arxiv.org/abs/1606.00503>.

HEARD, P. **React Native Architecture : Explained!** 2017. Acessado em: 28-Jun-2017. Disponível em: <https://www.logicroom.co/react-native-architecture-explained/>.

HERMES, D. **Xamarin Mobile Application Development**: Apress, 2016. ISBN 9781484202159.

HIERONS, R. M. et al. Using Formal Specifications to Support Testing. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 41, n. 2, p. 1–76, 2009. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/1459352.1459354>.

HOFFMAN, D. Using Oracles in Test Automation. In: **Proceedings of the 19th Pacific Northwest Software Quality Conference (PNSQC 2001)**, 2001.

HOLL, K.; ELBERZHAGER, F. Quality assurance of mobile applications: A systematic mapping study. In: **Proceedings of the 15th International Conference on Mobile and Ubiquitous Multimedia**. New York, NY, USA: ACM, 2016. (MUM '16), p. 101–113. ISBN 978-1-4503-4860-7. Disponível em: <http://doi.acm.org/10.1145/3012709.3012718>.

IBM. **Native, web or hybrid mobile-app development**. 2012. Acessado em: 06-Out-2017. Disponível em: <ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf>.

IDC. **Smartphone OS Market Share, 2017 Q1**. 2017. Acessado em: 31-Out-2017. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>.

IEEE. **IEEE Standard Glossary of Software Engineering Terminology**. IEEE, 1990. ISSN 0-7381-0391-8. ISBN 155937067X. Disponível em: <http://ieeexplore.ieee.org/xpls/absall.jsp?arnumber=159342>.

IONIC. **IONIC**. 2017. Acessado em: 06-Out-2017. Disponível em: <http://ionicframework.com/>.

ISTQB. GLOSSÁRIO PADRÃO DE TERMOS UTILIZADOS EM TESTE DE SOFTWARE. v. 01, n. novembro, p. 1–111, 2014.

JOORABCHI, M. E.; ALI, M.; MESBAH, A. Detecting inconsistencies in multi-platform mobile apps. In: **IEEE 26th International Symposium on Software Reliability Engineering (ISSRE 2015)**: IEEE, 2015. p. 450–460. ISBN 978-1-5090-0406-5.

JOORABCHI, M. E.; MESBAH, A.; KRUCHTEN, P. Real Challenges in Mobile App Development. **IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013)**, p. 15–24, 2013. ISSN 19493770.

KURKOVSKY, S. Pervasive Computing : Past , Present and Future. 2007.

LATIF, M. et al. Cross platform approach for mobile application development: A survey. **2016 International Conference on Information Technology for Organizations Development (IT4OD)**, p. 1–5, 2016.

LECHETA, R. R. **Google Android - Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 4ª. ed. São Paulo: Novatec, 2015. ISBN 978-85-7522-440-3.

LECHETA, R. R. **Desenvolvendo para iPhone e iPad**. 4. ed. São Paulo: Novatec, 2016. ISBN 978-85-7522-480-9.

LEE, S.; DOLBY, J.; RYU, S. HybriDroid : Analysis Framework for Android Hybrid Applications. **31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)**, p. 250–261, 2016.

LEOTTA, M. et al. 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**, 2013. (JAMAICA 2013), p. 53–58. ISBN 978-1-4503-2161-7.

LEOTTA, M. et al. 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**, 2013. p. 487–488. ISSN 2159-4848.

LEOTTA, M. et al. Reducing web test cases aging by means of robust XPath locators. **Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014**, p. 449–454, 2014.

LEOTTA, M. et al. Using Multi-Locators to Increase the Robustness of Web Test Cases. In: **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**: IEEE, 2015. p. 1–10. ISBN 978-1-4799-7125-1.

LI, X. et al. ATOM: Automatic Maintenance of GUI Test Scripts for Evolving Mobile Applications. **Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017**, p. 161–171, 2017.

LOPES, S. **Aplicações mobiles híbridas com Cordova e PhoneGap**: Casa do Código, 2016.

MACHIRY, A.; TAHILIANI, R.; NAIK, M. Dynodroid: An input generation system for android apps. In: **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: ACM, 2013. (ESEC/FSE 2013), p. 224–234. ISBN 978-1-4503-2237-9. Disponível em: <http://doi.acm.org/10.1145/2491411.2491450>.

MALAVOLTA, I. et al. End Users' Perception of Hybrid Mobile Apps in the Google Play Store. **Proceedings - 2015 IEEE 3rd International Conference on Mobile Services, MS 2015**, n. iii, p. 25–32, 2015. ISSN 0740-7459.

MALAVOLTA, I. et al. Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation. In: **Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on**, 2015. p. 56–59.

MAO, K.; HARMAN, M.; JIA, Y. Sapienz: Multi-objective automated testing for android applications. In: **Proceedings of the 25th International Symposium on Software Testing and Analysis**. New York, NY, USA: ACM, 2016. (ISSTA 2016), p. 94–105. ISBN 978-1-4503-4390-9. Disponível em: <http://doi.acm.org/10.1145/2931037.2931054>.

MASIELLO, E.; FRIEDMAN, J. **Mastering React Native**: Packt Publishing Ltd, 2017. ISBN 9781785885785.

MCKEEMAN, W. M. Differential Testing for Software. **Digital Technical Journal**, v. 10, n. 1, p. 100–107, 1998. ISSN 0898-901X. Disponível em: <http://www.cs.dartmouth.edu/ mckeeman/references/DifferentialTestingForSoftware.pdf>.

MENEGASSI, A. A.; ENDO, A. T. An evaluation of automated tests for hybrid mobile applications. In: **2016 XLII Latin American Computing Conference (CLEI)**, 2016. p. 1–11.

MENEGASSI, A. A.; ENDO, A. T. **x-PATeSCO - Cross-Platform App Test Script Recorder e Pacote Experimental**. 2018. Acessado em: 10-Jan-2018. Disponível em: <https://github.com/andremenegassi/x-patesco>.

MICROSOFT. **Common Language Runtime**. 2016. Acessado em: 25-Jun-2017. Disponível em: <https://msdn.microsoft.com/pt-br/library/8bs2ecf4(v=vs.110).aspx>.

MICROSOFT. **O que é um aplicativo da Plataforma Universal do Windows (UWP)?** 2016. Acessado em: 06-Out-2017. Disponível em: <https://msdn.microsoft.com/windows/uwp/get-started/whats-a-uwp>.

MICROSOFT. **XAML**. 2016. Acessado em: 25-Jun-2017. Disponível em: <https://msdn.microsoft.com/pt-br/library/ms752059(v=vs.110).aspx>.

MICROSOFT. **Unit Testing Framework**. 2017. Acessado em: 15-Mar-2017. Disponível em: <https://msdn.microsoft.com/en-us/library/ms243147(vs.80).aspx>.

MILANI, A. **Programando para iPhone e IPad**. 2. ed. São Paulo: Novatec, 2014.

MIRSHOKRAIE, S.; MESBAH, A.; PATTABIRAMAN, K. JSEFT: Automated JS Unit Test Generation. In: **Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on**, 2015. p. 1–10.

MOHAMED, A.; ZULKERNINE, M. Failure Type-Aware Reliability Assessment with Component Failure Dependency. In: **Secure Software Integration and Reliability Improvement (SSIRI), 2010 Fourth International Conference on**, 2010. p. 98–105.

MOODLE. **Moodle Tracker**. 2017. Acessado em: 07-Abr-2017. Disponível em: <https://tracker.moodle.org/projects/mobile>.

MUCCINI, H.; Di Francesco, A.; ESPOSITO, P. Software testing of mobile applications: Challenges and future research directions. In: **2012 7th International Workshop on Automation of Software Test**. IEEE, 2012. p. 29–35. ISBN 978-1-4673-1822-8. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6228987>.

MYERS, G. J. **The Art of Software Testing**. Second edi. New Jersey: John Wiley, 2004. ISBN 0-471-46912-2.

NAGAPPAN, M.; SHIHAB, E. Future Trends in Software Engineering Research for Mobile Apps. In: **2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)**: IEEE, 2016. p. 21–32. ISBN 978-1-5090-1855-0.

NATIVESCRIPT. **NativeScript**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://www.nativescript.org>.

OKEDIRAN, O. O.; ARULOGUN, O. T.; GANIYU, R. a. Mobile Operating System and Application Development Platforms : A Survey. **Journal of Advancement in Engineering and Technology**, v. 1, n. 4, p. 1–7, 2014. ISSN 2348-2931.

OLIVEIRA, R. A. P. de.

Apoio à automatização de oráculos de teste para programas com interfaces gráficas — USP - São Carlos, 2012. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-30032012-144613/pt-br.php>.

PASTORE, S. Mobile operating systems and apps development strategies. **2013 International Conference on Systems, Control and Informatics**, p. 350–358, 2013.

PHONEGAP. **PhoneGap**. 2016. Acessado em: 06-Out-2017. Disponível em: <http://phonegap.com/>.

RAO, G.; PACHUNOORI, A. **Optimized Identification Techniques Using XPath**. 2013. Acessado em: 10-Set-2017. Disponível em: <https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/a82c60c3-d3d9-4444-9f9f-63678cf12c17/document/e52a6c11-d183-4030-81fd-2afbad35e805/media>.

REACTNATIVE. **ReactNative**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://facebook.github.io/react-native/>.

REACTXP. **Frequently Asked Questions** . 2017. Acessado em: 2017-Nov-16. Disponível em: <https://microsoft.github.io/reactxp/docs/faq>.

REACTXP. **ReactXP**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://microsoft.github.io/reactxp/>.

REYNOLDS, M. **Xamarin Essentials**, 2014. ISBN 9781783550838. Disponível em: <http://amazon.com/o/ASIN/178355083X/>.

SCHWEIGHOFER, T.; HERIČKO, M. Mobile Device and Technology Characteristics' Impact on Mobile Application Testing. **SQAMIA 2013 Software Quality Analysis, Monitoring, Improvement, and Applications**, p. 103–108, 2013. ISSN 16130073. Disponível em: <http://ceur-ws.org/Vol-1053/SQAMIA2013FullProceedings.pdf#page=109%5Cnhttp://ceur-ws.org/Vol-1053/>.

SENCHA. **Sencha**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://www.sencha.com/>.

SILVA, E. et al. Computação Ubíqua – Definição e Exemplos. **Revista de Empreendedorismo, Inovação e Tecnologia**, v. 2, n. 1, p. 23–32, jun 2015. ISSN 23593539. Disponível em: <http://www.bibliotekevirtual.org/index.php/2013-02-07-03-02-35/2013-02-07-03-03-11/1748-reit-imed/v02n01/18365-computacao-ubiqua-definicao-e-exemplos.html>.

SONG, H.; RYOO, S.; KIM, J. H. An integrated test automation framework for testing on heterogeneous mobile platforms. **Proceedings - 1st ACIS International Symposium on Software and Network Engineering (SSNE 2011)**, p. 141–145, 2011.

STATISTA. **Number of apps available in leading app stores as of June 2016**. 2016. Acessado em: 25-Fev-2017. Disponível em: <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.

STATISTA. **Android version market share distribution among smartphone owners as of September 2017**. 2017. Acessado em: 09-Out-2017. Disponível em: <https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/>.

STATISTA. **Most popular installed mobile development software development kits (SDKs) across global mobile apps as of December 2016**. 2017. Acessado em: 31-Out-2017. Disponível em: <https://www.statista.com/statistics/742418/leading-mobile-app-development-sdks/>.

STATISTA. **Number of apps available in leading app stores as of March 2017**. 2017. Acessado em: 31-Out-2017. Disponível em: <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.

STATISTA. **Share of Apple devices by iOS version worldwide from 2016 to 2017**. 2017. Acessado em: 09-Out-2017. Disponível em: <https://www.statista.com/statistics/565270/apple-devices-ios-version-share-worldwide/>.

STEVEN, A. **What are the best mobile testing tools?** 2016. Acessado em: 26-Out-2017. Disponível em: <https://www.quora.com/What-are-the-best-mobile-testing-tools>.

TANENBAUM, A. S.; BOS, H. **Sistemas Operacionais Modernos**. 4ª. ed.São Paulo: , 2016.

TAO, C.; GAO, J. Modeling Mobile Application Test Platform and Environment: Testing Criteria and Complexity Analysis. In: **Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing**. New York, NY, USA: ACM, 2014. (JAMAICA 2014), p. 28–33. ISBN 978-1-4503-2933-0.

TELERIK. **AppBuilder**. 2016. Acessado em: 07-Out-2017. Disponível em: <http://www.telerik.com/platform/appbuilder>.

TESTDROID. **Testdroid Recorded**. 2017. Acessado em: 06-Out-2017. Disponível em: <http://www.testdroid.com>.

TESTOBJECT. **TestObject**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://testobject.com>.

TESTPROJECT. **Test Automation Tools and Trends for 2016**. 2016. Acessado em: 2017-Mar-03. Disponível em: <https://blog.testproject.io/2016/03/16/test-automation-survey-2016>.

TRICE, A. **Fresh-Food-Finder App**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://github.com/triceam/Fresh-Food-Finder>.

VILKOMIR, S.; AMSTUTZ, B. Using Combinatorial Approaches for Testing Mobile Applications. **2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops**, p. 78–83, 2014. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6825641>.

VILKOMIR, S. et al. Effectiveness of Multi-device Testing Mobile Applications. In: **Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on**, 2015. p. 44–47.

W3C. **Selectors Level 3**. 2011. Acessado em: 05-Abr-2017. Disponível em: <https://www.w3.org/TR/2011/REC-css3-selectors-20110929/>.

W3C. **Extensible Markup Language (XML) 1.0**. 2017. Acessado em: 12-Nov-2017. Disponível em: <https://www.w3.org/TR/xml/>.

W3C. **W3C**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://www.w3.org/>.

W3C. **What is the Document Object Model?** 2017. Acessado em: 25-Set-2017. Disponível em: <http://www.w3.org/TR/DOM-Level-3-Core/introduction.html>.

W3C. **XML Path Language (XPath) Version 1.0**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://www.w3.org/TR/xpath/>.

WARGO, J. M. **Apache Cordova 3 - Programming**: Pearson Education, 2013.

WASSERMAN, A. I. Software Engineering Issues for Mobile Application Development. **FoSER 2010 Proceedings of the FSE/SDP workshop on Future of software engineering research**, p. 397–400, 2010. ISSN 9781450304276.

WEEX. **Weex**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://weex.incubator.apache.org/>.

WEI, L.; LIU, Y.; CHEUNG, S.-c. Taming Android Fragmentation : Characterizing and Detecting Compatibility Issues for Android Apps. **ASE '16 (31st IEEE/ACM International Conference on Automated Software Engineering)**, p. 226–237, 2016.

WEISER, M. The computer for the 21st century. **Scientific American**, v. 265, n. 3, p. 66–75, set. 1991.

WILLOCX, M.; VOSSAERT, J.; NAESSENS, V. A Quantitative Assessment of Performance in Mobile App Development Tools. In: **Mobile Services (MS), 2015 IEEE International Conference on**, 2015. p. 454–461.

WILLOCX, M.; VOSSAERT, J.; NAESSENS, V. Comparing performance parameters of mobile app development strategies. In: **Proceedings of the International Workshop on Mobile Software Engineering and Systems - MOBILESoft '16**, 2016. p. 38–47. ISBN 9781450341783. Disponível em: <http://dl.acm.org/citation.cfm?doid=2897073.2897092>.

XAMARIN. **Xamarin**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://www.xamarin.com/>.

XAMARIN, S. **Tasky App**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://github.com/xamarin/mobile-samples/tree/master/TaskyPortable>.

XANTHOPOULOS, S.; XINOGALOS, S. A comparative analysis of cross-platform development approaches for mobile applications. In: **Proceedings of the 6th Balkan Conference in Informatics**. New York, NY, USA: ACM, 2013. (BCI '13), p. 213–220. ISBN 978-1-4503-1851-8. Disponível em: <http://doi.acm.org/10.1145/2490257.2490292>.

XDK, I. **Intel XDK**. 2017. Acessado em: 06-Out-2017. Disponível em: <https://software.intel.com>.

ZEIN, S.; SALLEH, N.; GRUNDY, J. A systematic mapping study of mobile application testing techniques. **Journal of Systems and Software**, Elsevier Science Inc., New York, NY, USA, v. 117, n. C, p. 334–356, jul. 2016. ISSN 0164-1212. Disponível em: <https://doi.org/10.1016/j.jss.2016.03.065>.

ZHANG, T. et al. Compatibility Testing Service for Mobile Applications. **Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on**, n. April, p. 179–186, 2015.