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

ANDRÉ AUGUSTO MENEGASSI

# INVESTIGAÇÃO DE UMA ABORDAGEM DE TESTE PARA APLICAÇÕES MÓVEIS HÍBRIDAS MULTIPLATAFORMA

PROPOSTA DE DISSERTAÇÃO DE MESTRADO

CORNÉLIO PROCÓPIO
2016

ANDRÉ AUGUSTO MENEGASSI

# INVESTIGAÇÃO DE UMA ABORDAGEM DE TESTE PARA APLICAÇÕES MÓVEIS HÍBRIDAS MULTIPLATAFORMA

Proposta de Dissertação de Mestrado 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

2016

# RESUMO

MENEGASSI, André Augusto. **INVESTIGAÇÃO DE UMA ABORDAGEM DE TESTE PARA APLICAÇÕES MÓVEIS HÍBRIDAS MULTIPLATAFORMA**. 2016. 66 f. Proposta de Dissertação – Mestrado – Programa de Pós-graduação em Informática, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

**Contexto:** atualmente os dispositivos móveis fazem parte do dia-a-dia das pessoas e estão disponíveis em vários formatos como *smartphones, tablets e wearables*. Modernos sistemas operacionais (SOs) controlam o hardware desses dispositivos e servem como plataforma para execução de uma ampla variedade de software então denominadas aplicações móveis. O desenvolvimento de aplicações móveis pode ser classificado em três grupos: aplicações nativas, aplicações baseadas no navegador Web e aplicações híbridas. No atual cenário, os diversos SOs móveis fornecem cada qual sua plataforma desenvolvimento de aplicações nativas. Essa fragmentação dos SOs é considerada uma barreira para o desenvolvimento de aplicações móveis, uma vez que diminui o possível números de usuários, bloqueando-os para plataformas específicas. O uso de aplicações híbridas é considerado uma solução para evitar tal fragmentação, visto que sua principal característica é a capacidade de execução em múltiplas plataformas. No entanto, o teste em aplicações híbridas pode ser desafiador, uma vez que além da execução dos testes para identificar defeitos inseridos durante a fase desenvolvimento, testes para detectar defeitos de inconsistência provenientes da característica multiplataforma precisam ser realizados. Além disso, é importante que os testes automatizados sejam reaproveitados entre as diversas plataformas. **Objetivo:** esta proposta objetiva investigar como é possível implementar mecanismos de testes automatizados e reaproveitá-los no contexto das aplicações híbridas sob execução em diversas plataformas. Além disso, pretende-se investigar como inconsistências geradas a partir de um comportamento diferente da aplicação em outra plataforma se manifestam e se, por meio de execução de teste automatizados, tais defeitos podem ser revelados. **Método:** inicialmente pretende-se realizar um estudo exploratório para avaliar a automatização de teste de aplicações móveis híbridas multiplataforma. Em uma segunda etapa, serão investigadas soluções para a definição de uma abordagem e seu emprego em uma ferramenta para apoio ao teste de tais aplicações. Para finalizar, um estudo de caso será conduzido para avaliar o emprego da abordagem e a ferramenta de apoio.

**Palavras-chave:** Aplicações Móveis, Aplicações Híbridas, Computação Móvel, Multiplataforma, Teste de Software.

## ABSTRACT

MENEGASSI, André Menegassi. **RESEARCH OF AN APPROACH FOR TESTING HYBRID MOBILE APPLICATIONS CROSS-PLATFORM.** 2016. 66 f. Proposta de Dissertação – Mestrado – Programa de Pós-graduação em Informática, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

**Background:** currently, mobile devices are part of people's daily life and are available in several formats such smartphones, tablets and wearables. Moderns operacional systems (OSs) control the hardware of these devices and serve as a platform for performing a wide variety of software called mobile applications. The development of these applications can be grouped into three categories: native applications, Web applications and hybrid applications. In the current scenario, the various mobile OSs provides native application development. This OSs fragmentation's is considered a barrier to the development of mobile applications, because it decreases the possible numbers of users by blocking them for specific platforms. The use of hybrid applications is considered a solution to avoid such fragmentation, because its main feature is the ability to run on multiple platforms. However, the test on hybrid applications can be a challenge, besides identify defects entered during the development, tests to detect inconsistency from the cross-platform feature need to be performed. In addition, it is important that your automated tests to be reused between different platforms. **Objective**: this proposal aims to investigate how you can implement automated test engines and reuse them in the context of hybrid applications under running on multiple platforms. In addition, we intend to investigate how inconsistencies are generated from a different behavior in another application platform manifest themselves and, through automated test execution, such defects can be revealed. **Method:** initially, we intend to conduct an exploratory study to evaluate the test automation of hybrid mobile cross-platform applications. In the second stage, will be investigated solutions for the definition of an approach and your use on a tool to test support such applications. Finally, a case study will be conducted to evaluate the approach and tool.



**Keywords**: Mobile Application, Hybrid Application, Mobile Computing, Crossplatform, Software Testing.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| API | Aplication Programming Interface |
| ASF | Apache Software Foundation |
| BDD | Behavior Driven Development |
| DLL | Dynamic Link Library |
| DOM | Document Object Model |
| ES | Engenharia de Software |
| ESG | Event Sequence Graph |
| FSM | Finite State Machine |
| GUI | Graphical User Interface |
| IDC | International Data Corporation |
| MS | Mapeamento Sistemático |
| OHA | Open Handset Alliance |
| QP | Questão de Pesquisa |
| SDK | Software Developers Kit |
| SO | Sistema Operacional |
| UML | Unified Modeling Language |

# 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 da *International Data Corporation* (IDC, 2015) sobre o *Market Share* no uso dos SOs móveis, as plataformas Android, Apple iOS e WindowsPhone foram as mais consumidas no ano de 2015. Em outra pesquisa, a Gartner (2016) apresentou as vendas de *smartphones* no primeiro trimestre de 2016; o SO Android é o líder de mercado, seguido pelo iOS da Apple, Windows Phone da Microsoft e Blackberry da RIM.

Esses modernos SOs servem como plataforma para execução de uma ampla variedade de *software* então denominadas aplicações móveis. O site Statista (Statista, 2015, 2016) oferece estatísticas de julho de 2015 e 2016 sobre o número de aplicações disponíveis para *download* nas principais lojas de distribuição. A plataforma Android[2] possui o maior número de aplicações disponíveis para seus usuários, em segundo está Apple[3] e na terceira colocação a loja da Microsoft[4].

O desenvolvimento de aplicações móveis pode ser classificado em três grupos: aplicações nativas, aplicações baseadas no navegador Web e aplicações híbridas (IBM, 2012; Latif et al., 2016). As aplicações nativas são desenvolvidas utilizando a plataforma nativa do dispositivo, e com acesso direto a funções do SO móvel sem utilização de recursos intermediários. As aplicações Web são desenvolvidas com tecnologias empregadas na construção de software para Web como HTML5 (W3C, 2016a), CSS3 (W3C, 2016a) e Javascript (W3C, 2016a). São armazenadas em um servidor Web, executadas sob o navegador de Internet e não tem acesso a recursos avançados do SO. E finamente, as aplicações híbridas combinam recursos comuns das aplicações Web como HTML5, CSS3, Javascript e

[1] Dispositivos eletrônicos vestíveis como óculos e relógios.
[2] https://www.android.com
[3] http://www.apple.com
[4] https://www.microsoft.com/pt-br/windows/phones

com suporte direto às *Application Programming Interfaces* (APIs) nativas do SO, tal como as aplicações nativas. Sua principal característica é desenvolver uma vez e implantá-la em diversas plataformas.

No atual cenário, diversos SOs móveis estão disponíveis, cada qual com sua plataforma desenvolvimento de aplicações. Uma aplicação nativa não é capaz de ser executada diretamente em uma outra plataforma. O conceito do "escreva uma vez, execute em qualquer lugar" não pode ser aplicado às aplicações nativas. Essa fragmentação dos SOs é vista como uma barreira para o desenvolvimento de aplicações móveis, uma vez que diminui o possível números de usuários, bloqueando-os para plataformas específicas. Além disso, o desenvolvimento para diversas plataformas envolve a criação de diversos times de desenvolvedores específicos para cada plataforma, gerando redundância e aumentando os custos envolvidos no projeto (Malavolta et al., 2015a; Xanthopoulos; Xinogalos, 2013).

Boushehrinejadmoradi et al. (2015) e Joorabchi; Ali; Mesbah (2015) sugerem que para atrair mais usuários, implementar a mesma aplicação móvel em diferentes plataformas tornou-se uma prática comum da indústria e, portanto, pode gerar mais receita por meio da comercialização da aplicação e venda de anúncios comerciais. Idealmente, uma determinada aplicação multiplataforma deve fornecer a mesma funcionalidade e comportamento nas diferentes plataformas suportadas.

A Engenharia de Software (ES) para aplicações móveis se difere da tradicional por ser mais especializada. Alguns requisitos de software são específicos (ou ganham maior importância) nas aplicações móveis (Wasserman; Fosser, 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 plataformas de *hardware* e SO, (v) segurança das aplicações envolvendo acesso não permitido aos dados, (vi) interface gráfica de usuário (*Graphical User Interface* - GUI) deve atender às diretrizes da plataforma, (vii) otimização do consumo de energia e (viii) complexidade nas atividades de teste.

Com o objetivo de assegurar a qualidade do produto final, as aplicações móveis precisam passar por atividades de teste ao longo do processo de desenvolvimento. Teste de aplicações móveis vem ganhando importância na área, tal fato é compreendido pela quantidade de pesquisas exploradas nesse contexto. Zein, Salleh e Grundy (2016) conduziram uma pesquisa abrangente e em profundidade no

contexto de teste de aplicações móveis, no qual exploram diversos estudos que evidenciam o interesse de pesquisadores. Myers (2004) define teste 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 da plataforma, o usuário poderá classificá-la negativamente. Após a correção, uma nova submissão deverá ser realizada para nova aprovação, o que pode demorar alguns dias, retardando o acesso dos usuários à nova versão. Além disso, não é possível obrigar o usuário a realizar a atualização (Gronli; Ghinea, 2016).

## 1.1 MOTIVAÇÃO

As aplicações móveis são distribuídas em diversas plataformas, SOs e construídas em diversas linguagens de programação. Toda essa fragmentação torna necessário garantir que a aplicação se comporte da mesma forma, independentemente da plataforma na qual ela 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 (Wasserman; Fosser, 2010; Zein; Salleh; Grundy, 2016). Os dispositivos móveis trazem consigo portabilidade e diversidade, características que geram algumas restrições, como: telas limitadas, autonomia de energia limitada, mecanismos de conectividade e poder de processamento reduzido (Muccini; Di Francesco; Esposito, 2012).

Neste trabalho define-se Múltiplas Configurações como a variabilidade (características que variam em relação aos diversos produtos) de dispositivos móveis com configurações diferentes, tais com SO, tamanho de tela, vários tipos de sensores e hardware. Dada a diversidade de plataformas móveis, dispositivos e técnicas de desenvolvimento, o teste de aplicações móveis para múltiplas configurações é um desafio para quem as produzem. 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; Mesbah; Kruchten, 2013). Além do exposto

anteriormente, o teste em um único dispositivo não garante o funcionamento correto em outro (Nagappan; Shihab, 2015).

No cenário das aplicações multiplataformas, o uso de aplicações híbridas é considerado uma solução para evitar a fragmentação da aplicação em diversos SOs (Malavolta et al., 2015b), visto que sua principal característica é a capacidade de execução em múltiplas plataformas. No entanto, o teste em aplicações híbridas pode ser mais desafiador, uma vez que além da execução dos testes para identificar defeitos inseridos durante a fase desenvolvimento, testes para encontrar defeitos de inconsistência provenientes da característica multiplataforma da aplicação precisam ser identificados. Baseado no trabalho de Joorabchi, Ali e Mesbah (2015), um defeito de inconsistência está relacionado a qualquer diferença no comportamento da execução da aplicação sob diversos dispositivos de múltiplas configurações. 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 (Cortez, 2015).

## 1.2 OBJETIVOS

O objetivo desta proposta de mestrado é desenvolver uma abordagem para o teste automatizado de aplicações híbridas multiplataforma. Em particular, pretende-se responder as seguintes questões de pesquisa (QP):

***QP1**: Como desenvolver um teste automatizado para aplicações móveis híbridas que seja executável em múltiplas configurações?*

Partindo do princípio que a aplicação híbrida é a mesma para cada plataforma, a QP1 objetiva investigar como é possível implementar mecanismos de testes automatizados e reaproveitá-los em múltiplas configurações, de tal forma a evitar a reescrita do *script* de teste para cada plataforma. A ideia é manter um único *script* de teste para aplicação independentemente de seu ambiente de execução.

***QP2**: Como usar o teste automatizado para encontrar defeitos de inconsistência entre as múltiplas configurações?*

Uma aplicação híbrida pode apresentar defeitos de inconsistência gerados a partir de comportamentos diferentes em cada configuração. Dessa forma, a QP2 objetiva investigar como tais inconsistências se manifestam e se, por meio de execução de teste automatizados, tais defeitos podem ser revelados automaticamente.

## 1.3 ORGANIZAÇÃO DO TEXTO

Esta proposta de mestrado está organizada da seguinte forma: no Capítulo 2 são apresentados os principais conceitos sobre aplicações móveis e, especificamente, sobre aplicações híbridas multiplataforma. No Capítulo 3 são abordados os principais conceitos sobre teste de software, principalmente no âmbito das aplicações móveis, bem como seus desafios e trabalhos relacionados. E finalizando, o Capítulo 4 descreve as etapas que serão seguidas para cumprir os objetivos, os resultados esperados e o cronograma de desenvolvimento.

# 2 APLICAÇÕES MÓVEIS

## 2.1 CONSIDERAÇÕES INICIAIS

Neste capítulo são apresentados a fundamentação teórica e a contextualização sobre aplicações móveis. Na Seção 2.2 é apresentada uma introdução geral sobre a computação móvel e seu estado atual. Na Seção 2.3 são apresentados os conceitos gerais de aplicações móveis. Na Seção 2.4 são apresentados definições sobre aplicações multiplataforma, e na subseção 2.4.1 são apresentados os principais *frameworks* para desenvolvimento de aplicações móvel híbridas com suporte a execução em multiplataformas, e por fim, na subseção 2.4.2 são detalhados a estrutura do *framework* Apache Cordova e de uma aplicação de exemplo.

## 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 de 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), cunha o termo computação ubíqua (as vezes chamada de pervasiva), e faz referência a onipresença dos dispositivos móveis por meio da integração crescente da computação com o mundo físico, é a computação visível ou invisível incorporada ao ambiente humano (Kurkovsky, 2007; Silva et al., 2015).

Atualmente, os dispositivos móveis estão disponíveis em vários formatos e equipados com poderosos processadores, ampla capacidade de armazenamento de dados e diversos sensores (Boushehrinejadmoradi et al., 2015).

Uma pesquisa da IDC (2015), destaca as plataformas Android, Apple iOS e Windows Phone como as mais utilizadas no ano de 2015. No mercado de desenvolvimento de software, as aplicações para dispositivos móveis são ofertadas e disponibilizadas em modernas lojas de distribuição. A plataforma Android possui o maior número de aplicações disponíveis para seus usuários, em segundo está Apple e na terceira colocação a loja da Microsoft (Statista, 2015, 2016).

De acordo com Dei e 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.

Um SO 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 SO 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.

No estudo conduzido por Okediran, Arulogun e Ganiyu (2014), foram apresentados os seis mais populares SO móveis: Android, iOS, Windows Phone, Blackberry OS, webOS e Symbian OS. Segundo uma pesquisa da Gartner (2016) apresentando as vendas de *smartphones* no primeiro trimestre de 2016, o SO Android é o líder de mercado, seguido pelo iOS da Apple, Windows Phone da Microsoft e Blackberry da RIM. A seguir são apresentados os três SO móveis mais utilizados (IDC, 2015).

**Android.** O Android é um SO móvel livre e de código aberto *(open source)* e está disponível para diversas plataformas, como *smartphones* e *tablets*, TV, relógios, óculos e automóveis (Deitel; Harvey; Alexander, 2016; Lecheta, 2015). O SO 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*. Sua primeira versão chegou ao mercado em 2008 (Filho, 2014; Lecheta, 2015).

A plataforma é um resultado de uma aliança chamada *Open Handset Alliance* (OHA), um grupo formado por gigantes do mercado de telefonia de celulares liderados pelo Google. O objetivo do grupo é definir uma plataforma única e aberta

para celulares, moderna e flexível para o desenvolvimento de aplicações coorporativas (Ableson et al., 2012; Lecheta, 2015).

As aplicações Android são desenvolvidas em linguagem Java. Após compilação, o código é convertido para o formato *Dalvik Executable*[5] (.dex), 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 (Lecheta, 2015; Tanenbaum; Bos, 2016). O principal ponto de distribuição das aplicações Android é a loja Google Play[6].

**iOS**. 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 *Software Developers Kit* (SDK) oficial para desenvolvimento de aplicativos juntamente com loja de distribuição de aplicativos AppStore[7] (Alasdair, 2013; Lecheta, 2016). 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 aos *hardwares* construídos pela Apple, portanto, somente os dispositivos da própria autora são os que executam com sucesso o iOS (Milani, 2014).

As linguagens de programação oficiais para a plataforma iOS são Objective-C e a linguagem Swift, sendo esta 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 Mac (Lecheta, 2016; Milani, 2014). Para disponibilizar aplicações na AppStore, é necessário adquirir uma licença da Apple como um desenvolvedor.

**Windows Phone**. O Windows Phone foi lançado em 2010 pela Microsoft, substituindo sua plataforma Windows Mobile (Shackles, 2012). A segunda geração denominada Windows Phone 8 foi lançada em outubro de 2012. Numa nova tentativa

[5] 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).

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

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

de aumentar sua participação no mercado de *smartphones* e ser um alternativa ao iOS e Android, a Microsoft lança a Plataforma Universal do Windows (UWP), objetivando o desenvolvimento de aplicativos para suas diversas plataformas, tais como Windows Phone, Windows 8/10 (Microsoft, 2016a). A linguagem C# em conjunto com o Microsoft VisualStudio formam o cenário padrão para desenvolvimento de aplicações para a plataforma.

## 2.3 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[8] ou Twitter[9]. 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 aplicações móveis das aplicações convencionais. As aplicações móveis são oferecidas para um conjunto de dispositivos e plataformas 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 em matéria de poder de processamento, memória e armazenamento em relação ao computador tradicional, tais características não podem ser ignoradas, caso contrário a experiência do usuário ao usar a aplicação será negativa. Além disso, a presença específica de sensores (por exemplo, GPS e câmera) podem ser exploradas no desenvolvimento da aplicações (Pastore, 2013).

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

[8] http://www.facebook.com
[9] http://www.twitter.com

**(a) aplicações nativas:** Aplicações nativas possuem arquivos binários executáveis instalados e armazenados diretamente para o dispositivo. A maneira mais popular para obter e instalar uma aplicação nativa é visitando a loja de distribuição de aplicações da plataforma móvel. Uma aplicação nativa é codificada em um 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 API que são disponibilizadas (IBM, 2012), conforme esquematizado na Figura 1. Uma das principais vantagens desse tipo de aplicação envolve seu desempenho superior comparado com o demais tipos, acesso direto ao hardware do dispositivo, além de fornecer uma aparência consistente (Xanthopoulos; Xinogalos, 2013).

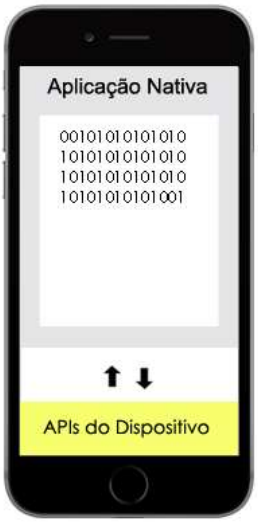


**Figura 1: Aplicação móvel nativa (adaptado de IBM, 2012)**

As diferentes ferramentas, linguagens de programação e o formato de empacotamento da aplicação nativa estão relacionados na Tabela 1.

**Tabela 1: Sistema Operacionais e detalhes de desenvolvimento (adaptada de IBM, 2012 e Latif et al., 2016)**

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

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

**(b) aplicações baseadas no navegador Web:** Os atuais dispositivos móveis oferecem modernos e consistentes navegadores Web, com suporte aos novos recursos do HTML5 (W3C, 2016a), CSS3 (W3C, 2016a) e Javascript (W3C, 2016a). A especificação do HTML5 inclui recursos de geolocalização, armazenamento local *off-line*, formatos de mídias (áudio e vídeo), além de avançados componentes de interface de usuário (IBM, 2012). É um tipo de aplicação construída com recursos da Web, armazenada num servidor Web e com execução diretamente no navegador (baseada no navegador *(em inglês, browser-based))* do dispositivo (Figura 2).


**Figura 2: Aplicações móveis baseadas na Web (adaptado de IBM, 2012)**

Uma desvantagem desse tipo de aplicação é a ausência de acesso aos vários recursos oferecidos pelo SO móvel, limitando o tipo de aplicação que pode ser construída. Outra desvantagem é que as aplicações não estão disponíveis nas lojas de distribuição oficial das plataformas, sendo necessário a utilização de outros meios para disponibilizar à aplicação ao usuário. Como vantagem é possível citar a disponibilização da mesma aplicação para todas as plataformas móveis *(cross-platform)*, como também o clássico *Desktop*, devido às características de portabilidade oferecidas pelas tecnologias Web utilizadas.

**(c) aplicações híbridas:** Combina características das aplicações nativas e das aplicações baseadas no navegador Web. As aplicações são desenvolvidas usando os recursos comuns das aplicações Web como HTML5, CSS3, 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 (IBM, 2012; Latif et al., 2016). A aplicação é dividida em duas partes, uma nativa e outra conhecida como *WebView*, responsável por executar o HTML, CSS e Javascript, tal como ilustrado na Figura 3 (Xanthopoulos; Xinogalos, 2013).


**Figura 3: Aplicações móveis híbridas (adaptado de IBM, 2012; LATIF et al., 2016)**

Nesse contexto, são utilizados *frameworks* de desenvolvimento híbridos como PhoneGap[10]/Cordova[11] e Sencha Touch[12]. O desenvolvedor especifica a lógica da aplicação e interface do usuário 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 aos recursos nativos do dispositivo (Boushehrinejadmoradi et al., 2015). A vantagem desse tipo de aplicação está no resultado final, uma aplicação que pode ser disponibilizada nas lojas de distribuição de todas as plataformas móveis. A porção Web das aplicações híbridas pode ser uma página Web armazenada num servidor ou arquivos HTML, CSS e Javascript empacotados e armazenados localmente dentro da aplicação.

[10] http://phonegap.com
[11] https://cordova.apache.org
[12] https://www.sencha.com

Uma desvantagem está no desempenho da aplicação. Estudos indicam um desempenho inferior comparado com aplicações nativas (Corral; Sillitti; Succi, 2012).

Levando em consideração as características apresentadas sobre os tipos de aplicações móveis, a Tabela 2 relaciona as principais.

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

| Características | Nativa | Baseadas na 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 | Sim (limitado) |

*O estudo conduzido por CORRAL; SILLITTI; SUCCI (2012) indica um desempenho inferior comparado com aplicações nativas.

## 2.4 Aplicações móveis multiplataforma

Conforme 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 uma aplicação apenas uma vez e então executá-la em qualquer plataforma, independentemente do seu SO. Com o mesmo objetivo a Microsoft introduziu o .NET Compact Framework, uma versão compactada do seu .NET Framework, mas especificamente para dispositivos móveis. Com 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.

A fragmentação de SOs móveis impossibilita que uma aplicação nativa se beneficie do conceito "escreva uma vez, execute em qualquer lugar". Tal fragmentação é vista como problema, pois diminui o possível números de usuários, bloqueando-os para plataformas específicas. Além disso, o desenvolvimento para diversas plataformas envolve a criação de diversos times de desenvolvedores específicos para cada plataforma, gerando redundância e aumentando os custos envolvidos no projeto (Malavolta et al., 2015a; Xanthopoulos; Xinogalos, 2013). Uma primeira solução é o uso de *frameworks* de desenvolvimento nativo capazes de

traduzir uma linguagem nativa para outra linguagem compatível com outras plataformas. Um exemplo desse tipo de *framework* é o Xamarin[13] e o Apportable[14] (Boushehrinejadmoradi et al., 2015). No caso do Xamarin, a aplicação é desenvolvida com código-fonte em linguagem C#, compilada para Windows Phone e traduzida para Apple iOS ou Android (Latif et al., 2016), e o Apportable traduz código construído para Android originalmente construído para Apple iOS.

Uma segunda solução é o uso de *frameworks* para desenvolvimento aplicações híbridas capazes de execução multiplaforma. O desenvolvedor especifica a lógica da aplicação e interface de usuário usando componentes do desenvolvimento Web, no caso HTML, CSS e Javascript.

### 2.4.1 *Frameworks* para desenvolvimento de aplicações móveis híbridas multiplataforma

Os termos “híbrido” e “multiplataforma” estão associados a características diferentes em uma aplicação móvel. O termo *Híbrido* está relacionado ao uso de recursos da programação Web e nativos da plataforma móvel durante a implementação e execução da aplicação. E o termo *multiplataforma* está relacionado ao conceito de escrever uma vez e executar em várias plataformas. A acoplamento dessas duas características é explorada por *frameworks* comerciais e *open source*, dessa forma, os desenvolvedores podem desenvolver aplicações móveis para diferentes plataformas sem duplicações de esforços, tais como o custo do projeto e tempo. Alguns *frameworks* são apresentados a seguir.

**Cordova:** Disponibilizado gratuitamente, é um *framework open source* para desenvolvimento móvel. O Cordova tem como plataformas móveis alvo: Android, Blackberry, iOS e Windows Phone. Outras plataformas como Ubuntu, Windows (8.1/10) e OS X também são suportadas. O acesso aos recursos nativos do SO é baseado em *plugins* capazes de serem invocados por meio de código Javascript. Posteriormente, o Cordova será descrito em detalhes.

[13] https://xamarin.com
[14] http://www.apportable.com

**PhoneGap:** O PhoneGap é uma distribuição de *open source* do projeto Cordova pela Adobe. A Adobe[15] automatizou o processo de compilação para ocorrer em nuvem (PhoneGapBuild[16]), sem a necessidade de instalação local do SDK de desenvolvimento do SO móvel. A licença não-comercial possui algumas limitações referente a compilação do código-fonte em nuvem (PhoneGap, 2016).

**Sencha Touch:** É um *framework* comercial que fornece suporte para criar aplicações que funcionam no Android, iOS, BlackBerry, Windows Phone e Tizen. O Senha Touch utiliza Cordova/PhoneGap para acesso nativos aos recursos do dispositivo (Sencha, 2016).

**IONIC:** É um *framework open source* focado principalmente na aparência e interação da interface de usuário da aplicação. É capaz de gerar aplicações para Android, iOS e Windows 10 Universal App[17]. O acesso aos recursos nativos ocorre por meio do Cordova/PhoneGap (IONIC, 2016).

**Intel XDK:** É um *framework* não comercial fornecido pela Intel[18] suportando 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. Oferece um ambiente de desenvolvimento próprio (XDK, 2016).

**AppBuilder:** Disponibilizada sob licença comercial é uma plataforma da Telerik[19] capaz de gerar aplicações híbridas para o Android, iOS e Windows Phone. O ambiente de desenvolvimento é baseado em uma plataforma de nuvem, dispensando a configuração de ambientais locais para desenvolvimento. O acesso nativo da aplicação aos recursos do dispositivo é realizado com *plugins* do Cordova/PhoneGap (Telerik, 2016).

Foi possível perceber que o Apache Cordova é a base de todos os outros *frameworks* apresentados.

---

[15] http://www.adobe.com/
[16] https://build.phonegap.com/
[17] Desenvolvimento de aplicações destinadas a uma ampla variedade de dispositivos, incluindo móveis, área de trabalho, HoloLens, Surface Hub e Xbox (Microsoft, 2016b).
[18] https://software.intel.com/pt-br
[19] http://www.telerik.com/

### 2.4.2 Apache Cordova

Atualmente o Apache Cordova (Cordova, 2016) é o *framework* mais comum para construção de aplicações móveis e multiplataforma (Lopes, 2016). 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 *Apache Software Foundation* (ASF), sendo rebatizado como Apache Cordova e disponibilizado como um projeto *open source.* O PhoneGap atualmente é apenas uma distribuição mantida pela Adobe. A maior diferença entre os dois é que o PhoneGap está integrado com o serviço comercial de compilação em nuvem mantido pela Adobe, dispensando a instalação local do SDK de desenvolvimento do SO móvel.

O Cordova consiste nos seguintes componentes (Wargo, 2013):

- Um projeto nativo (Figura 4-a) para cada SO suportado, contendo um componente nativo chamado *WebView* (Figura 4-b) usado para processar HTML5, CSS e Javascript.
- Um conjunto de APIs chamadas de *plugins* (Figura 4-c) para fornecer acesso aos recursos nativos do SO por meio do *WebView* (Figura 4-b). Um *plugin* é implementado em duas partes, a primeira parte (Figura 4-c) é uma biblioteca Javascript e a segunda parte (Figura 4-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 parte Javascript do *plugin,* e o *WebView* e por sua vez fornece uma interface entre as partes (biblioteca Javascript (Figura 4-c) e nativa (Figura 4-d)), assim tornando possível o uso dos recursos nativos (Figura 4-e).
- E um conjunto de ferramentas chamadas de Cordova-CLI usadas para gerenciar a criação de projetos de aplicações móveis, instalação de *plugins*, e atalhos para gerar compilações das aplicações através dos SDKs nativos, além de outras funcionalidades.

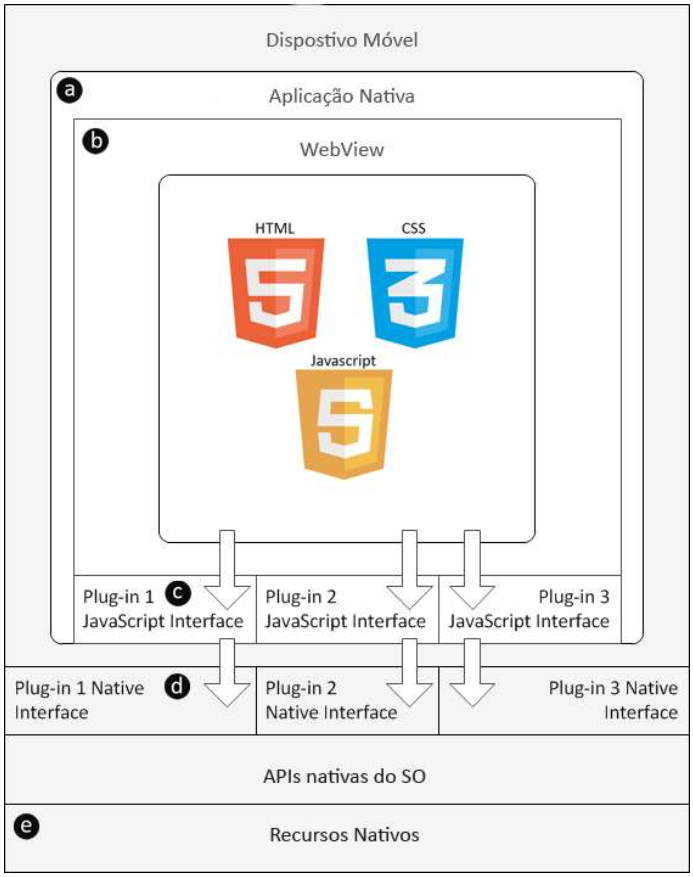


**Figura 4: Arquitetura de uma aplicação híbrida (adaptado de WARGO (2013)).**

Além dos *plugins* oficiais criados pela ASF, novos *plugins* podem ser criados por terceiros e disponibilizados para comunidade de desenvolvedores. O site do *framework*[20] disponibiliza um repositório de *plugins* para pesquisa e *download*. Os recursos nativos suportados pela API do Cordova e acessados via Javascript por *plugins* estão disponíveis na Tabela 3.

**Tabela 3: Recursos nativos suportados pelo Cordova (adaptado de Camden (2016))**

| Recurso | Descrição |
|---|---|
| Situação da Bateria | Provê informações sobre os níveis da bateria. |
| Câmera | Provê acesso a câmera e imagens existentes no dispositivo. |
| Contatos | Acesso completo a agenda de contatos. |
| Informações do dispositivo | Provê informações sobre o nome do dispositivo e nome do SO. |
| Sensores de movimento e orientação. | Detecção de movimentação e orientação. |
| Diálogos e recursos de vibração. | Recursos de alertas em áudio, visual e tátil. |
| Sistema de Arquivos | Acesso ao sistema de arquivos do SO. |
| Geolocalização | Provê informações sobre a localização do dispositivo. |

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

| | |
|---|---|
| Globalização | Provê informações sobre formato de data, valores e dinheiro de acordo com a região do mundo. |
| *InAppBrowser* | Suporte a janela *popup*. |
| Captura de mídia | Gravação de áudio e vídeo. |
| Informações da rede | Provê informações sobre o status da rede. |
| Tela de abertura *(Splashscreen)* | Definição de tela de abertura da aplicação. |
| Barra de *status*. | Comunicação com a barra de *status* do SO móvel. |
| *Whitelist* | Especifica quais recursos remotos são permitidos pela aplicação (por exemplo, requisição HTTP assíncrona), ajudando a prevenir problemas de segurança de conteúdo. |

Sugerindo que os *plugins* são um tipo de componente utilizado pelo *framework.* Um componente de software pode ser definido como uma unidade de composição com interfaces especificadas contratualmente e com dependências de contexto explícitas; um componente de software pode ser implantado de forma independente ou se combinar com outros (Szyperski, 1997). Um componente de software é uma unidade coesa e de baixo acoplamento que denota a simples abstração de um serviço (Booch, 1987).

Os elementos HTML da aplicação, como <*body*> e <*p*> são organizados em tempo de execução numa estrutura hierárquica de árvore chamada *Document Object Model (DOM)* (W3C, 2016b), a qual oferece uma API para representar e manipular o conteúdo da aplicação (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. O Cordova não tem a capacidade de compilar as aplicações. A compilação ocorre somente pelo SDK fornecido pela organização proprietária do SO móvel, dessa forma, sendo necessário sua instalação e configuração no computador de desenvolvimento.

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 5. A aplicação é capaz de receber dados de um contato (nome, telefone e e-mail), tirar uma foto ou usar uma foto da galeria de

imagens do usuário, e armazenar os dados cadastrados em formato JSON[21] diretamente no sistema de arquivos local. Outras funcionalidades também estão disponíveis, tais como pesquisa de contatos, alteração e exclusão dos dados.

A aplicação usou recursos de desenvolvimento Web e nativo, conforme caracteriza-se uma aplicação híbrida. A interface de usuário foi construída com HTML e CSS, e com apoio da biblioteca Material Design[22] 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 GUI e nas regras de negócio, além da manipulação de *plugins* do Cordova utilizados para acesso aos recursos nativos do SO, tais como o sistema de arquivos e a câmera do dispositivo.

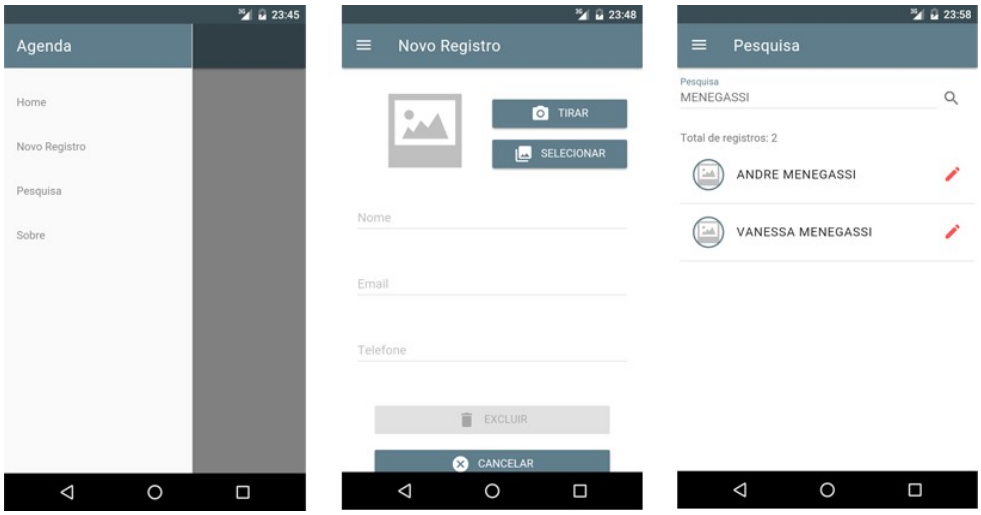


**Figura 5: Aplicação de Exemplo (fonte própria)**

Um projeto Cordova é organizado em diretórios, seguindo uma estrutura padrão concebida para separar diferentes tipos de arquivos (Figura 6), como descrito a seguir:

- ***hooks:*** Contém *scripts* usados para personalizar os comandos da ferramenta de interface de linha do Cordova (Cordova-Cli).

[21] Javascript Notation Object (JSON) é uma estrutura textual para estruturação de dados e usualmente comum para realizar troca de dados entre ambientes computacionais diferentes.
[22] https://material.google.com/

- ***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 extraídos para esse diretório.
- ***www***: Contém os artefatos da programação Web do projeto, tais como arquivos HTML, CSS e Javascript.

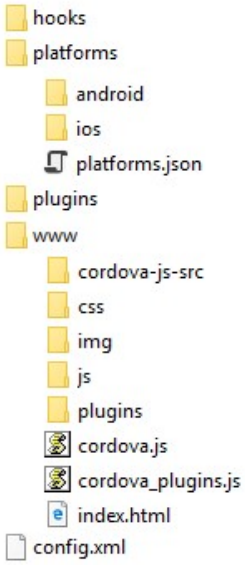


**Figura 6: Estrutura padrão de diretórios da aplicação.**

O arquivo *config.xml* (Figura 7) é arquivo no formato XML baseado na especificação do W3C para empacotamento de aplicativos Web (Camden, 2016). Contém dados referente a configuração da aplicação para o projeto Cordova, 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 da aplicação móvel. 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 7: Arquivo de configuração do projeto Cordova.**

A Figura 8 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*[23] 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 da Linha 2) com as configurações do *plugin*.

[23] https://cordova.apache.org/docs/en/2.4.0/cordova/camera/camera.getPicture.html

```
getFoto: function(sourceType) {
    var config = {
        allowEdit: true,
        quality: 70,
        encodingType: Camera.EncodingType.PNG,
        destinationType: Camera.DestinationType.DATA_URL,
        sourceType: sourceType,
        targetWidth: 250,
        targetHeight: 500,
        mediaType: Camera.MediaType.PICTURE
    };
    navigator.camera.getPicture(function(imageData) {
        imageData = "" + imageData;
        document.querySelector("#imgCadastro").src = imageData;
        }, function() {
            alert("Imagem não foi selecionada");
        }, config);
}
```

**Figura 8: Demonstração do *plugin* para acesso ao recurso câmera (nativo).**

## 2.5 CONSIDERAÇÕES FINAIS

Neste capítulo foram introduzidos os principais conceitos e características sobre a computação móvel, SOs móveis, aplicações móveis e seus diversos tipos. Foram apresentados os principais conceitos sobre as aplicações híbridas e multiplataformas. Alguns *frameworks* para desenvolvimento dessas aplicações também foram descritos. O Apache Cordova foi detalhado e uma aplicação de exemplo foi gerada para facilitar o entendimento da arquitetura adotada.

No próximo capítulo, são apresentados os principais conceitos sobre teste de software, 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 proposta são descritos e um mapeamento sistemático da literatura é apresentado.

# 3 TESTE DE APLICAÇÕES MÓVEIS

## 3.1 CONSIDERAÇÕES INICIAIS

Neste capítulo são apresentas a contextualização e a fundamentação teórica necessárias para o entendimento do trabalho. Na Seção 3.2 são apresentados conceitos de teste de software, bem como uma breve introdução ao TBM. Na Seção 3.3 os desafios em testar aplicações móveis são relacionados. Na Seção 3.4 é referenciado e apresentado um mapeamento sistemático da literatura na área de estudo deste projeto. A Seção 3.5 complementa a seção anterior com trabalhos relacionados. Na seção 3.6 são apresentadas algumas ferramentas específicas e suas características para o teste de aplicações móveis são identificadas. E por fim, a Seção 3.7 discute os testes inerentes as aplicações híbridas multiplaforma.

## 3.2 FUNDAMENTOS DE TESTE DE SOFTWARE

Com o objetivo de assegurar a qualidade do produto final, as aplicações móveis precisam passar por atividades de teste ao longo do processo de desenvolvimento. Na literatura, a definição para Teste de Software apresentada por Myers (2004) é utilizada amplamente em outros trabalhos, e define teste como um processo de execução de um programa com a intenção de encontrar defeitos.

As atividades de testes ocorrem em três fases (Barbosa; Maldonado; Vincenzi, 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 testes. Um caso de teste consiste em uma especificação de valores de entradas 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. Num 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 testes, 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 dados de teste, uma ou “n” vezes e avaliar seus resultados identificando se o teste passou ou falhou (oráculo) (Hoffman, 2001). O emprego de testes automatizados é uma boa prática para o aumento da produtividade e para diminuição dos custos, além de serem reproduzidos diversas vezes com possibilidade de avaliação dos resultados gerados 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; Maldonado; Vincenzi, 2004).

***(i) Teste funcional:*** Conhecido como teste caixa-preta, tem por objetivo verificar as funções do software, observando somente as entradas e saídas geradas pelo software. Os detalhes da implementação a 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; Maldonado; Vincenzi, 2004; Myers, 2004).

***(ii) Teste estrutural:*** Conhecido como teste caixa-branca, tem por objetivo verificar a estrutura interna do software para derivar casos de testes (Barbosa; Maldonado; Vincenzi, 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; Maldonado; Jino, 2007). Estão categorizadas com base na complexidade, no fluxo de controle e no fluxo de dados.

***(iii) Teste baseados 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; Maldonado; Vincenzi, 2004).

### 3.2.1 Conceito de Teste Baseado em Modelo (TBM)

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 testes são modelados baseando-se na arquitetura e/ou modelagens detalhadas de um componente ou sistema (por exemplo: testes de interfaces entre componentes ou sistemas), dessa forma, descrevendo o comportamento pretendido durante a execução do sistema controlado por um software de teste (ISTQB, 2014). As pesquisas sugerem uma divisão do TBM em quatro passos principais, categorizando-os em (i) modelagem com base nos requisitos de software, (ii) geração de casos de testes baseado do modelo adotado, (iii) concretização do casos de testes num formato executável por um software de teste, e por fim (iv) execução dos testes (Bouquet et al., 2006; El-Far; Whittaker, 2002; Pretschner; Philipps, 2005).

O uso do TBM oferece algumas vantagens como, por exemplo, (i) geração automática de casos de teste, (ii) rastreabilidade entre requisitos de sistema e critérios de teste e (iii) adequação a mudanças em requisitos (Bouquet et al., 2007; Grieskamp et al., 2011; Utting; Legeard, 2007). A geração de casos de teste é automatizada por ferramentas de apoio ao TBM que se utilizam de um modelo formal que descreve os requisitos do software sob teste. O TBM pode ser aplicado para avaliar se as funcionalidades implementadas estão de acordo com os documentos e artefatos de especificação, pois o modelo de teste elaborado representa os eventos e suas transições. Por fim, pode-se, quando necessário, atualizar o modelo de teste para que esse acompanhe as evoluções do software. Assim, as alterações no software se refletem no modelo e novos casos de teste são gerados, reduzindo o tempo e custo em relação ao teste manual. No entanto, em contextos mais específicos, partes de um software podem ser difíceis de modelar, assim o teste manual pode ser mais efetivo. Além disso, quando um teste falha, o defeito pode estar no próprio modelo, dificultando ainda mais a descoberta do defeito (El-Far; Whittaker, 2002).

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 formal, ou 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 Estado Finitos *(FSM - do inglês, Finite State Machine)* da *Unified Modeling Language (UML),* que mapeiam os eventos e as sequências válidas de transição entre os eventos da funcionalidade sob teste (Endo, 2013).

## 3.3 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, qualidades que geram algumas restrições (Muccini; Di Francesco; Esposito, 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 diferentes sensores (GPS, bússola, acelerômetro, etc.), vários tipos de telas com diferentes tamanhos, além de capacidade de processamento diferentes (Amalfitano et al., 2013). Estas características geram desafios na condução dos testes em aplicações móveis. Nesse contexto, Muccini, Di Francesco e Esposito (2012) definem dez peculiaridades referentes as aplicações móveis e suas implicações nos testes:

- **Conectividade:** É uma das características mais peculiares e críticas de uma aplicação móvel. Está relacionada como a aplicação se conecta a uma rede. A confiabilidade da aplicação depende fortemente do tipo de conectividade disponível. Testes devem ser realizados nas funcionalidades que envolvem o uso de redes de dados.
- **Recursos Limitados:** Recursos como processador e memória dos dispositivos móveis são ainda pequenos ao compará-los com os recursos dos computadores tradicionais. O uso de tais recurso precisam ser continuamente monitorados para evitar a degradação do desempenho e o funcionamento incorreto do sistema. Ações devem ser tomadas em caso de escassez de recursos.
- **Autonomia:** Esta peculiaridade está relacionada ao consumo de energia pela aplicação. A redução da energia pode ser um problema para o seu correto funcionamento. O consumo de energia pode ser avaliado através de testes e monitoração contínua.
- **Interface de Usuário:** Os desenvolvedores devem seguir diretrizes *(guidelines)* para construção de GUIs. As aplicações móveis podem ter uma aparência diferente do esperado dependendo da resolução ou dimensão de tela do dispositivo móvel. Testes de GUI é uma das primeiras necessidades envolvendo as atividades de teste das aplicações.
- **Sensibilidade ao Contexto:** As aplicações móveis dependem de sensores como provedores de dados sobre o contexto (ambiente em que o dispositivo está inserido), tais como ruído/som, luminosidade, movimento, largura de banda, dispositivos próximos, localização e

outros. Testar se a aplicação vai funcionar corretamente em qualquer ambiente e sob qualquer entrada contextual é um desafio e pode levar a uma explosão combinatorial das possíveis entradas.

- **Adaptação:** Esta peculiaridade está relacionada se a aplicação pode se adaptar e evoluir conduzida por informações contextuais. Tal evolução pode acontecer durante o uso da aplicação e não deve interrompê-la. A confiabilidade da aplicação pode ser afetada por adaptações imprevistas. Abordagens para teste de adaptação/evolução precisam ser desenvolvidas.
- **Novas linguagens de programação:** Novas linguagens de programação são concebidas para apoiar a mobilidade. Técnicas de testes estruturais precisam ser revista, a fim de atender tais linguagens.
- **Novos SOs:** As aplicações móveis são executadas sob novos SOs que ainda são apenas parcialmente confiáveis. Novas versões são frequentemente lançadas e nem sempre é garantido a compatibilidade com versões anteriores. Falhas nas aplicações podem ocorrer devido à variabilidade e falta de confiabilidade do SO.
- **Diversidade de dispositivos e SOs:** Diversos e diferentes dispositivos móveis são ofertados no mercado, produzidos por diferentes fornecedores e com características diferentes de SO e hardware. As aplicações móveis, durante a execução em diferentes dispositivos, podem se comportar de forma não esperada devido às variações no hardware ou SO. Técnicas de teste para maximizar a cobertura da diversidade devem ser planejadas.
- **Tela sensível ao toque:** As telas sensíveis ao toque são os principais meios para entrada de dados do usuário em uma aplicação móvel. O tempo de resposta do sistema ao ocorrer um evento de toque depende da utilização de recursos do dispositivo no mesmo momento, e pode tornar-se lento em determinadas circunstâncias (por exemplo, processador ocupado). Técnicas de testes têm de ser introduzidas para testar o funcionamento da tela sob circunstâncias diferentes.

Segundo Nagappan e Shihab (2015), uma vasta gama de estudos apresentam técnicas para ajudar os desenvolvedores a melhorar os testes em aplicações móvel, em particular, tentando melhorar o teste de GUI e de cobertura de sistema. Um dos maiores desafios nessa linha é a incapacidade em atingir uma alta cobertura de código. Schweighofer e 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 GUI apresenta-se diferente com base na resolução de tela do dispositivo e suas dimensões. Os autores recomendam testar a GUI no máximo de dispositivos móveis possíveis.

Segundo Griebe e Gruhn (2014), mudanças no contexto expandem o conjunto de casos de teste significativamente. O testador não tem apenas que testar as funcionalidades da aplicação, fornecendo as interações do usuário, ele também tem que fornecer essas interações sob diferentes contextos.

O estouro do limite do recurso (ou vazamento) pode afetar significativamente a confiabilidade do software e experiência a usuário. No entanto não existe uma abordagem abrangente e fundamenta para a detecção de tais vazamentos (Yan; Yang; Rountev, 2013).

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 capazes de execução multiplataforma foram disponibilizadas nas lojas de distribuição. Além disso, as aplicações precisam executar em hardware diferente com diferentes versões do SO. O teste em um único dispositivo não garante o funcionamento correto em outro (Nagappan; Shihab, 2015). 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; Mesbah; Kruchten, 2013).

De acordo com Wang e Alshboul (2015), teste de segurança envolvendo aplicações móveis é uma questão desafiadora e não há nenhuma abordagem eficaz. Ferramentas podem ser utilizadas para decompilar e recompilar um arquivo APK (pacote de instalação da aplicação no Android), permitindo a alteração do código por um usuário mal-intencionado e, por fim, novamente recompilado.

## 3.4 MAPEAMENTO SISTEMÁTICO EM TESTE DE APLICAÇÕES MÓVEIS

Uma revisão sistemática da literatura é um meio de identificar, avaliar e interpretar todas as pesquisas disponíveis relevantes para determinadas questões de investigação, tópico de uma área ou fenômeno de interesse. Um tipo específico de revisão sistemática é o mapeamento sistemático (MS), no qual os termos de pesquisa são menos focados do que para revisões sistemáticas e provavelmente retorna um número muito mais amplos de estudos (Kitchenham; Charters, 2007). Ainda segundo Kitchenham e Charters (2007), as principais diferenças entre um estudo de mapeamento e revisão sistemática são:

- Estudos de mapeamento geralmente têm perguntas mais amplas de pesquisa.
- Os termos de pesquisa para estudos de mapeamento serão menos focados do que revisões sistemáticas e são propensos a retornar um grande número de estudos.
- O processo de extração de dados para estudos de mapeamento também é muito mais amplo do que o processo de extração de dados de revisões sistemáticas e podem mais precisamente ser denominado uma fase de classificação ou categorização. A finalidade dessa fase é classificar documentos com detalhes suficientes para responder às perguntas de pesquisa ampla e identificar documentos posteriores e avaliações sem ser uma tarefa demorada.

Objetivando encontrar estudos publicados na área de teste de aplicações móveis, Endo et al. (2016) realizaram um MS, no qual parte dos resultados serão exploradas neste texto.

### 3.4.1 Processo de Busca e Condução

As bases de dados utilizadas no MS foram *ScienceDirect, SCOPUS, IEEE xplore, ACM digital library, Springer Link e Web of Science*. Para cada base foi preparada uma expressão de busca adequada, tendo como parâmetro a seguinte expressão genérica derivadas de “aplicação móveis” e “teste de software”: *("mobile*

*application" OR "mobile software" OR "mobile app" OR "smartphone application" OR "smartphone app") AND (testing OR test OR "fault detection").* Também foram realizadas buscas manuais nos *sites* de alguns dos principais eventos e publicações da área de teste de software, foram eles: *International Conference on Software Engineering (ICSE), IEEE Transactions on Software Engineering (TSE), ACM Transactions on Software Engineering and Methodology (TOSEM), Journal of Systems and Software (JSS), Information and Software Technology (IST), ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), IEEE/ACM International Conference on Automated Software Engineering (ASE), Empirical Software Engineering Journal (ESEJ), International Symposium on Software Reliability Engineering (ISSRE), International Symposium on Software Testing and Analysis (ISSTA), IEEE International Conference on Software Testing, Verification and Validation (ICST) e Software Testing, Verification and Reliability (STVR).*

Em complementação à lista inicial de estudos, suas referências foram analisadas *(Backward Snowballing)* com o objetivo de encontrar novos estudos. Além disso, o MS conduzido por Zein, Salleh e Grundy (2016), forneceu estudos para compor a lista inicial.

Alguns critérios de inclusão e exclusão foram aplicados para redução dos estudos retornados das bases. Ao final, 146 estudos foram incluídos no MS, e posteriormente analisados pelos pesquisadores, de tal forma a responder as questões de pesquisas (QP) propostas por eles e apresentadas a seguir:

- **QP01.** Quais os estudos existentes sobre o teste de aplicações móveis?

  Esta questão objetivou identificar o que foi investigado sobre teste de aplicações móvel, fornecendo uma lista abrangente de pesquisas existentes na área.

- **QP02.** Quando e quais os canais os estudos sobre testes em aplicações móveis foram publicados?

  O objetivo foi identificar a frequência de estudos considerando o ano e eventos de publicação.

- **QP03.** Que desafios/particularidades para testar aplicações móveis foram identificados e em que medida foram tratadas pelas abordagens identificadas?

  Os pesquisadores definiram uma lista de desafios relacionados ao teste de aplicações móveis. A frequência desses desafios será analisada.

- **QP04.** Quais estratégias de testes foram aplicadas nos testes de aplicações móveis?

  O objetivo da questão foi identificar a técnica: funcional (caixa-preta) ou estrutural (caixa-branca), e os níveis do teste: unidade, integração, sistema ou aceitação utilizados nos estudos.

- **QP05.** Quais foram as plataformas móveis adotadas nos estudos?

  O objetivo foi classificar as plataformas utilizadas nos estudos, tais como Android, iOS, Windows Phone, Symbian, JME e outras.

- **QP06.** Quais os tipos de aplicação móvel foram adotados nos estudos?

  O objetivo foi classificar as aplicações utilizadas nos estudos entre nativa, Web ou híbrida.

- **QP07.** Que tipo de método científico de pesquisa foi utilizado nos estudos?

  O objetivo da questão foi identificar os métodos científicos de pesquisas utilizados nos estudos, para tal, foi empregue os métodos de pesquisas sugeridos por Wieringa et al. (2006): Pesquisa de Avaliação *(em inglês, Evaluation Research)*, Proposta de Solução *(em inglês, Proposal of Solution)*, Pesquisa de Validação *(em inglês, Validation Research)*, Trabalhos Filosóficos *(em inglês, Philosophical Papers)* e Artigo de Experiência Pessoal *(em inglês, Personal Experience Papers).*

- **QP08.** Se houver um estudo empírico, que tipo são representados?

  O objetivo da questão foi classificar os estudos empíricos, tal como estudo de caso, *survey*, experimento controlado, entre outros (Wohlin et al., 2012).

- **QP09.** Se houver um estudo empírico, que tipo de projeto foi adotado?

Esta questão objetivou encontrar o tipo de aplicações utilizada nos testes, tais como aplicações específicas para o estudo *(toy)*, *open source* ou industrial.

- **QP10.** Há algum suporte automatizado?

  Esta questão verificou por evidências de suporte a teste automatizados para as aplicações.

### 3.4.2 Análise Parcial de Resultados e Discussão

Após a identificação dos estudos relevantes para o MS, uma tabulação dos resultados foi realizada com base nas QPs a serem respondidas e as características procuradas nos estudos. Analisando a evolução das publicações na área ilustrada pela Figura 9, percebe-se que a partir de 2011 o número de estudos sobre teste de aplicações móveis começou a crescer, chegando o seu topo em 2014, com 38 estudos relevantes. O MS somente considerou o primeiro semestre para o ano de 2016, dessa forma justificando o baixo número de estudos incluídos nesse ano.


**Figura 9: Número de publicações por ano (adaptado de ENDO et al., (2016)).**

Os pesquisadores identificaram que 55,48% dos estudos evidenciam de alguma forma o uso de teste automatizado nos projetos de aplicações móveis, enquanto 36,99% dos estudos não abordam a automatização de teste. E apenas 7,53% dos estudos não tratam diretamente de teste automatizado, tal como estudos

que contextualizam o cenário de teste de aplicações móveis, MS, Revisão Sistemática, *position papers* e outros tipos de estudos. A distribuição dos estudos considerado o uso de teste automatizado está disponível da Figura 10.

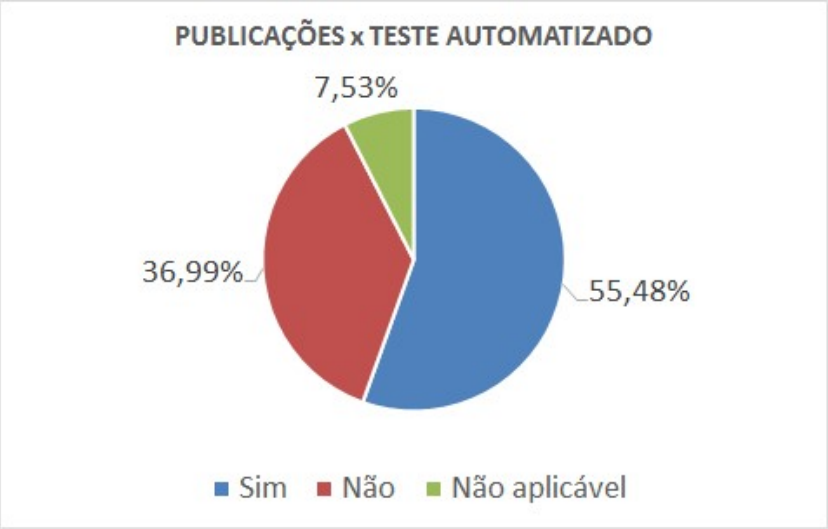


**Figura 10: Evidências de teste automatizados nos estudos (adaptado de ENDO et al., (2016)).**

Os pesquisadores procuram nos estudos indicações sobre testes em aplicações nativas, Web e/ou híbridas. A maioria dos estudos (59,86%) abordam apenas aplicações nativas, dois estudos (1,36%) abordam aplicações Web e apenas um estudo (0,68%) abordou teste em aplicações híbridas, Tao e Gao (2014), explorado na Seção Trabalhos Relacionados. E 38,10% não abordam diretamente o uso de teste automatizado, tal como estudos de MS, Revisão Sistemática, *position papers* e outros tipos de estudos. Percebe-se uma carência em estudos abordando teste de software em aplicações móveis híbridas. A distribuição dos tipos de aplicações abordadas nos estudos está ilustrada na Figura 11.

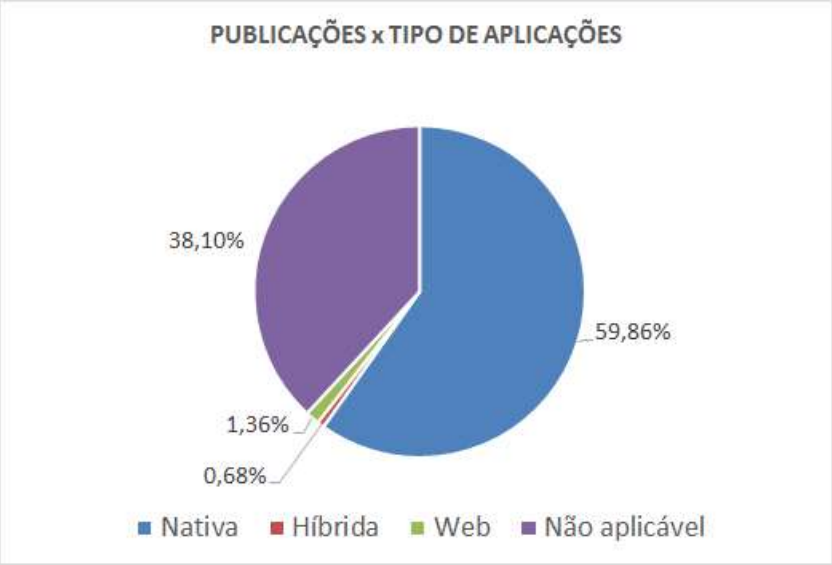


**Figura 11: Tipos de aplicações abordadas nos estudos (adaptado de Endo et al., (2016)).**

## 3.5 TRABALHOS RELACIONADOS

O único estudo incluído no MS liderado por Endo et al. (2016) a abordar aplicações móveis híbridas foi conduzido por Tao e Gao (2014). O foco do estudo 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 móvel 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.

Zein, Salleh e Grundy (2016) conduziram um outro MS com propósito de identificar pesquisas existentes para técnicas de teste em aplicações móveis, 79 pesquisas foram incluídas, avaliadas e discutidas. As técnicas de testes identificadas estão focadas em aplicações móveis nativas, salvo a exceção do trabalho de Gao et al. (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, linguagens de *script* e protocolos de conectividade entre ferramentas de teste e plataformas. É sugerido o uso de computação em nuvem para simular diversos dispositivos conectados, gerando uma solução de controle e execução de teste 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)* 445 aplicações móveis são híbridas (3,73%) dentre as mais de 11 mil aplicações selecionadas para a pesquisa, *(ii)* observou que categorias como fotografia, música e áudio, jogos e personalização têm em menor número aplicações híbridas, provavelmente pela forte necessidade de alto desempenho, e *(iii)* revelou o Apache Cordova como o *framework* mais utilizado na construção de aplicações híbridas. Os pesquisadores estenderam esse trabalho em um novo estudo (Malavolta et al., 2015b), no qual foram coletadas 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 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 empreda 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.

Gudmundsson et al. (2016) aplicaram o uso do TBM no contexto nas aplicações móveis para o SO Android. O principal objetivo do estudo foi examinar se

TBM pode ser usado de maneira eficaz, eficiente e com um esforço razoável, para testar aplicações móveis usando a mesma abordagem associadas a outros tipos de sistemas. Os pesquisadores descobriram que TBM fornece uma maneira sistemática para testar aplicações móveis e embora existam ainda etapas manuais envolvidas, é possível alcançar um alto grau de automação com esforço razoável.

Amalfitano et al. (2015) propuseram uma abordagem para teste em aplicações móveis usando o TBM, modelos de aprendizagem e teste combinatorial *(pairwise).* Para cada par do teste combinatorial foi gerado um caso de teste cobrindo um caminho da Máquina de Estado. A abordagem deu origem a ferramenta MobiGUITAR, na qual foi colocada em demonstração para encontrar falhas em quatro aplicações móveis da loja do Android (Google Play). No total foram gerados 7.711 casos de teste que ajudaram a identificar dez defeitos. Os pesquisadores concluíram que a combinação de TBM e modelo de aprendizagem é uma abordagem promissora para alcançar melhor detecção de defeitos em aplicações Android.

Lee, Dolby e Ryu (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 intercomunicação entre a linguagem Java e Javascript (ambiente nativo e Web) utilizada nas aplicações híbridas. *Frameworks* híbridos utilizam-se de mecanismos de mapeamento ou ponte *(em inglês, bridge)* para realizar a partir do Javascript chamadas ao código nativo. 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.

Saad e Bakar (2014) apresentaram uma lista com dez *frameworks* para teste de aplicações móveis e destacam características relacionadas a capacidade de

teste em multiplataformas. Na avaliação geral, os pesquisadores escolheram o *framework* comercial *Micro Focus Silk Mobile*. A escolha foi baseada em alguns benefícios e em funções atrativas oferecidas, além do suporte efetivo a teste em múltiplas plataformas móveis.

Joorabchi, Ali e Mesbah (2015) afirmam que idealmente uma determinada aplicação multiplataforma deve fornecer a mesma funcionalidade e comportamento nas diferentes plataformas suportadas. Nesse cenário o teste de software é apresentado como um desafio. Os pesquisadores 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 GUI) 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 GUI 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 GUI, tais como botões e rótulos.

Boushehrinejadmoradi et al. (2015) realizaram testes em *frameworks* de desenvolvimento de aplicações móveis multiplataforma. O estudo sugere que há duas classificações para *frameworks* de desenvolvimento de aplicações multiplataforma: (i) *Web-based framework* e (ii) *Native framework*. Esta 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[24]. 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, Ryoo e Kim (2011) objetivaram desenvolver um *framework* de automação de teste com suporte a plataformas heterogêneas. A ideia foi gerar casos de teste compatíveis 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. Os pesquisadores afirmam que esse tipo de teste quando voltado para aplicações móveis é um desafio devido a sua complexidade e custo relacionado a diversidade de plataformas e dispositivos. Nesse contexto é proposto uma estratégia para reduzir os custos e melhorar a eficiência da execução do teste de compatibilidade baseando-se num algoritmo na qual seleciona os dispositivos móveis, diversas plataformas e configurações. A ideia é 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

[24] 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).

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.

Starov et al. (2015) investigaram o estado da arte sobre tecnologias relacionadas ao teste de aplicações móveis como serviço. Diferentes serviços de teste usam dos benefícios da computação em nuvem. Entre os benefícios está a redução do tempo e custos das atividades de teste, e a aplicação do teste em um grande número de possíveis combinações de dispositivos móveis e SOs diferentes. O estudo apresenta um comparativo entre 13 serviços de teste em nuvem.

## 3.6 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 4. Observa-se que o Appium[25] e Calabash[26] dentre os *frameworks* relacionados, são os únicos *open source* com suporte a teste multiplataforma e em aplicações híbridas.

**Tabela 4: Ferramentas x Características (adaptado de STEVEN (2016)).**

| | **Tipo de Aplicação** | | | **SO** | | | | **Linguagem do script** | | | | | **Licença** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Características x Ferramentas** | *Nativa* | *Web* | *Híbrida* | *Android* | *iOS* | *Windows Phone* | *Outros* | *Java* | *C#* | *ObjectiveC/Swift* | *Javascript* | *Outras* | *Open Source* | *Comercial* | *Outra* |
| **Calabash** | ✔ | | ✔ | ✔ | ✔ | | | | | | | ✔ | ✔ | | |
| **XCTest UI** | ✔ | | | | ✔ | | | | | ✔ | | | | | ✔ |
| **Ranorex** | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | | | | | ✔ | | ✔ | |

[25] http://appium.io
[26] http://calaba.sh

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **EggPlant** | ✔ | | | ✔ | ✔ | ✔ | ✔ | | | | | ✔ | | | |
| **Robotium** | ✔ | ✔ | ✔ | ✔ | | | | ✔ | | | | | ✔ | | |
| **Monkey Talk** | ✔ | ✔ | ✔ | ✔ | ✔ | | | | | | | ✔ | ✔ | | |
| **Monkey Test** | ✔ | ✔ | ✔ | ✔ | | | | | | | | ✔ | ✔ | | |
| **Selendroid** | ✔ | ✔ | ✔ | ✔ | | | | ✔ | | | ✔ | ✔ | ✔ | | |
| **Appium** | ✔ | ✔ | ✔ | ✔ | ✔ | | | ✔ | ✔ | | | ✔ | ✔ | | |
| **SmartBear** | ✔ | ✔ | ✔ | ✔ | ✔ | | | ✔ | ✔ | ✔ | ✔ | ✔ | | ✔ | |
| **UI Automator** | ✔ | ✔ | ✔ | ✔ | | | | ✔ | | | | | | | ✔ |
| **Robolectric** | ✔ | | | | | | | ✔ | | | | | ✔ | | |
| **Espresso** | ✔ | | ✔ | ✔ | | | | ✔ | | | | | | | ✔ |

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, Android e Windows, usando uma API do Selenium chamada de WebDriver[27]. Uso de tal API possibilita a reutilização de *script* de teste entre as diversas plataformas. 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 e executa-os em um dispositivo móvel. A resposta é uma mensagem HTTP que representa o resultado da execução do comando. Esta abordagem dá o poder de codificar casos de teste em qualquer linguagem de programação com suporte a clientes HTTP (Appium, 2016).

O Calabash é um *framework open source* para automatização de testes para plataformas Android e iOS e suporta aplicações nativas e híbridas. As bibliotecas do Calabash para cada plataforma são fornecidas separadamente. O Calabash utiliza a técnica de *Behavior Driven Development* (BDD) para expressar o comportamento esperado da aplicação em linguagem natural. O *script* de teste é codificado utilizando a abordagem BDD e em linguagem Ruby e pode ser reutilizado entre as plataformas (Calabash, 2016).

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

## 3.7 DEFEITOS EM APLICAÇÕES MÓVEIS HÍBRIDAS MULTIPLATAFORMAS

A Seção 2.4.2 demonstrou que uma aplicação móvel híbrida possui duas camadas, (i) uma camada nativa chamada de *WebView* capaz de executar código em uma linguagem específica da plataforma (Java, ObjectiveC, Swift, C#, etc), e (ii) uma camada capaz de executar recursos da Web, nesse caso HTML, CSS e Javascript. Por exemplo, uma aplicação híbrida para Android utiliza Javascript para mapear as interações do usuário na GUI e a linguagem Java para características específicas do dispositivo e/ou SO. A camada nativa é gerenciada pelo *WebView*, implementado para cada plataforma e defeitos de inconsistência relacionados a compatibilidade entre as plataformas podem surgir. O processamento de HTML, CSS e Javascript entre os distintos SOs podem ser ligeiramente diferentes e também causar uma falha na aplicação (Cortez, 2015).

Em específico para o Cordova, um *plugin* é implementado em duas partes, a primeira parte em Javascript e a segunda parte corresponde a sua implementação nativa específica para cada plataforma. Ao se entender que os *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 e Zulkernine (2010) é observado que a maioria das técnicas de avaliação da confiabilidade de um software não considera as falhas inseridas por componentes.

Levando em consideração a caraterística “multiplataforma”, falhas relacionadas a compatibilidade da execução da mesma aplicação em plataformas diferentes podem surgir. A aplicação pode funcionar corretamente na plataforma X e falhar na plataforma Y.

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; Mesbah; Pattabiraman, 2015). Além disso, o interpretador

HTML pode gerar resultados diferentes nas diversas plataformas, provocando inconsistências de GUI.

## 3.8 CONSIDERAÇÕES FINAIS

Neste capítulo foram introduzidos os principais conceitos sobre teste de software. Algumas ferramentas para teste automatizado em aplicações móveis foram sumarizadas com suas respectivas características. Além disso, foi descrito um trabalho de mapeamento sistemático da literatura e alguns trabalhos relacionados na área deste projeto. E por fim, foram discutidos os testes inerentes as aplicações híbridas e multiplaforma.

No próximo capítulo, é apresentada a proposta de desenvolvimento deste projeto que tem como principal objetivo contribuir para a atividade de teste de aplicações móveis híbridas multiplataforma.

# 4 PROPOSTA

Alguns estudos sugerem o interesse da academia no teste de aplicações móveis. No trabalho conduzido por Zein, Salleh e Grundy (2016), foram analisados diversos estudos que evidenciam o interesse de pesquisadores na área. Esse interesse também é verificado em Endo et al. (2016) ao analisar 146 artigos completos, publicados em eventos e revistas da área. No âmbito industrial, o interesse também é manifestado. No estudo qualitativo conduzido por Joorabchi, Mesbah e Kruchten (2013), no qual 188 desenvolvedores foram entrevistados, levantou as práticas e desafios no desenvolvimento dessas aplicações, sendo o teste uma das mais mencionadas.

Nesse contexto, as aplicações híbridas multiplataformas destacam-se por sua característica de execução em diversas plataformas, dispensando a necessidade da reescrita da aplicação para atender um conjunto de diferentes dispositivos. Essa característica introduz um desafio no teste das aplicações devido a variabilidade de múltiplas configurações de dispositivos existentes no mercado (Boushehrinejadmoradi et al., 2015; Gronli; Ghinea, 2016; Joorabchi; Mesbah; Kruchten, 2013). Para ilustração da variabilidade no contexto dos dispositivos móveis, a Figura 12 utiliza-se de um *Feature Model*[28] para representar todas as informações sobre um produto (dispositivo móvel fictício) em linha de produção. O modelo é uma representação hierárquica das características e relações entre elas.

[28] *Feature Model* (FM) é um método para descrever semelhanças e variabilidades dentro de uma família de sistemas, tal como um linha de produtos de software (Asikainen; Mannisto; Soininen, 2006). O FM representa as características padrão de uma família de sistemas dentro de um domínio e seus relacionamentos (Kang et al., 1990).

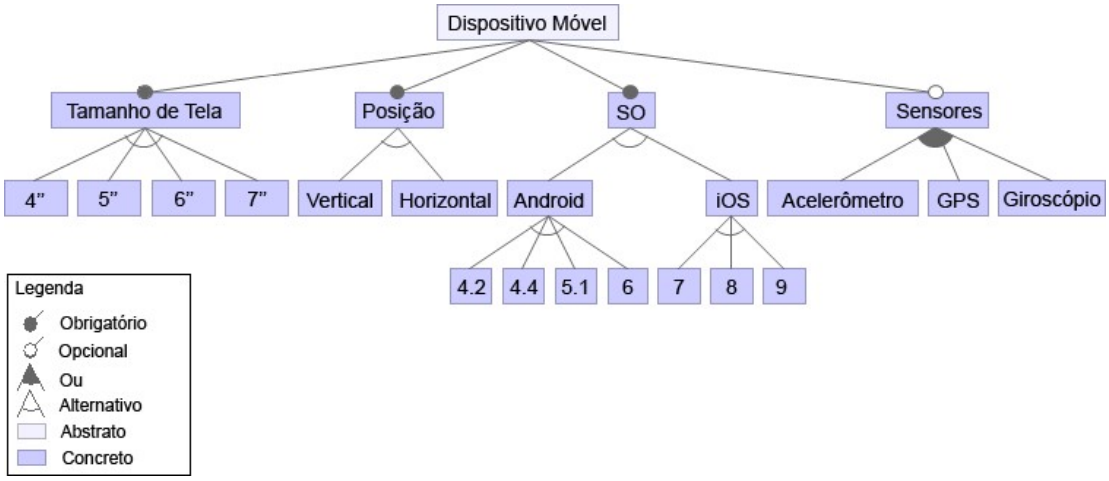


**Figura 12: Variabilidade de características de um dispositivo móvel.**

Cada combinação do FM pode representar uma configuração na qual a aplicação precisa ser testada. Neste projeto, essa característica (execução em diversas plataformas) será explorada no âmbito de teste de software, por meio do desenvolvimento de uma abordagem para teste automatizado sob execução em múltiplas configurações. As duas QPs propostas são reapresentadas e discutidas a seguir.

***QP1**: Como desenvolver um teste automatizado para aplicações móveis híbridas que seja executável em múltiplas configurações?*

Partindo do princípio que a aplicação híbrida é a mesma para cada plataforma, a QP1 objetiva investigar como é possível implementar mecanismos de testes automatizados e reaproveitá-los em um conjunto de dispositivos com múltiplas configurações, assim evitando a rescrita do *script* de teste para cada plataforma. A ideia é manter um único *script* de teste para aplicação independentemente de seu ambiente de execução.

*Frameworks* e suas respectivas características de apoio ao teste em aplicações móveis foram identificados e relacionados na Tabela 4. Observa-se que o Appium e o Calabash dentre os *frameworks* relacionados, são os únicos *open source* e com suporte ao teste multiplataforma e em aplicações híbridas. O Appium suporta a realização de teste de aplicações móveis nativas, Web ou híbridas, e com a possibilidade de reutilização do *script* de teste nas múltiplas configurações.

O Appium foi utilizado em uma pesquisa anterior conduzida pelo candidato (Menegassi; Endo, 2016) envolvendo o teste de três aplicações híbridas sob execução

em cinco dispositivos diferentes. Foi observado que a estrutura da GUI é representada por elementos XML que mapeiam cada componente da interface. Esses elementos são acessados programaticamente por meio de expressões de consulta *XPath*[29], possibilitando a realização de testes automatizados. O estudo identificou que a estrutura apresentada é diferente para os ambientes Android e iOS que foram analisados, mesmo entre suas versões, assim impactando na construção do *script* de teste. Em resumo, as três aplicações sob teste apresentaram tais diferenças estruturais. Tomando como exemplo uma das aplicações exploradas *(Fresh Food Finder),* ao comparar uma interface da aplicação sob execução na versão 4.4.4 com a versão 5.1 do Android, foi obtido uma diferença 67 nós de elementos XML, e ao comparar a versão a 4.4.4 com a 6.0.1 a diferença foi de 17 nós. Essa situação também foi observada nos dispositivos com o SO iOS. Para contornar tal situação foi necessário escrever expressões de seleção *XPath* compatíveis entre as diferentes plataformas, como é possível observar nas Linhas 6,7,8,9 e 10 do trecho do *script* de teste apresentado na Figura 13. Essa Figura é um trecho do *script* de teste que automatiza a execução de um caso de teste da aplicação *Fresh Food Finder* e dispara um determinado evento da GUI.

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

**Figura 13: Trecho de um *script* de teste compatibilizado para execução em múltiplas configurações (extraído de Menegassi e Endo (2016)).**

Algumas possíveis abordagens para solucionar o problema de incompatibilidade da seleção de elementos na GUI e tornar o *script* teste compatível são relacionadas a seguir:

[29] *XPath* é uma linguagem de consulta para selecionar elementos (nós) em estruturas computacionais chamadas de árvore representando documentos XML. Site: https://www.w3.org/TR/xpath.

- **Padrão de Projeto**: Padrões de Projetos *(em inglês, Design Patterns)* são soluções gerais para um problema que ocorre com frequência dentro de um determinado contexto no projeto de software. Um padrão identifica e abstrai os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos e reutilizável (Gamma et al., 2000). *PageObject* é um padrão de projeto para organização de testes funcionais (caixa-preta) (Vicente, 2014). A proposta desse padrão é encapsular por meio de uma camada os mecanismos necessários para encontrar e manipular os elementos da GUI a ser testada (Fowler, 2013). Um exemplo dessa estratégia aplicada ao Appium para teste automatizado nas plataformas Android e iOS é encontrada em Carretero (2015)**.** Dadas as definições, essa abordagem propõe a implementação do padrão *PageObject* para apoiar o Appium na construção de um *script* de teste comum entre as configurações.
- **Readequação do *script* de teste em tempo de execução**: Essa abordagem propõe utilizar uma configuração de referência para gerar o *script* de teste da aplicação sob teste. Instrumentar tal *script* para guardar dados que possibilitem a readequação em tempo de execução dos seletores *XPath* durante sua execução em uma outra configuração, objetivando sempre selecionar o elemento de GUI corretamente. É interessante que o emprego dessa abordagem seja implementado em uma ferramenta.
- **Mapeamento de elementos HTML para XML**: A GUI da aplicação híbrida é construída por elementos HTML. Em Menegassi e Endo (2016), foi identificado que o SO móvel representa a GUI da aplicação em uma estrutura de elementos XML, mapeando cada componente da interface. Uma possibilidade seria compreender o processo de mapeamento (ou transformação) de elementos HTML para XML utilizado pelo SO e fornecer subsídios para testador utilizar seletores CSS[30] ao contrário de *XPath*. Para tal, sugere-se a implementação da abordagem em uma ferramenta capaz

[30] Seletores CSS são padrões que utilizam a linguagem *Cascading Style Sheets (CSS)* para selecionar elementos em uma estrutura de árvore (W3C, 2016c). Seu uso é amplamente difundido no desenvolvimento de aplicações que usam HTML e CSS.

de gerar seletores *XPath* adequados para cada configuração a partir de seletores CSS.

***QP2**: Como usar o teste automatizado para encontrar defeitos de inconsistência entre as múltiplas configurações?*

Uma aplicação híbrida multiplataforma pode apresentar defeitos de inconsistência gerados a partir de comportamentos diferentes em cada configuração de dispositivo. Dessa forma, a QP2 objetiva investigar como tais inconsistências se manifestam e se, por meio de execução de teste automatizados, tais defeitos podem ser revelados automaticamente.

Nesta proposta, propõe-se investigar os defeitos de inconsistência em aplicações híbridas multiplataforma, e como identificá-los com testes. Alguns pontos que podem ser explorados são apresentados a seguir:

- **Árvore de componentes da GUI:** Ampliar a investigação do estudo anterior (Menegassi; Endo, 2016) sobre a diferença na representação da árvore de componentes de GUI, fator de incompatibilidade do *script* de teste entre as múltiplas configurações.
- **Aparência diferente da GUI**: Investigar o nível de diferença em relação a aparência da GUI da aplicação sob teste em execução nas múltiplas configurações. Para tal, faz-se necessário a captura de *screenshots* e a utilização de algoritmos diferenciais para estabelecer o nível similaridade entre as GUIs. Alguns elementos visuais característicos do SO móvel devem ser ignorados para não afetar o resultado diferencial, tais como a barra superior e elementos visuais de navegabilidade. A falta de similaridade pode indicar uma inconsistência entre as múltiplas configurações (Choudhary, 2011; Dallmeier et al., 2012).
- **Desempenho:** Investigar de forma exploratória a percepção dos usuários em relação ao desempenho da aplicação. Ao realizar uma determinada função, uma diferença discrepante de desempenho entre as configurações pode ser classificada como um indício de inconsistência.
- ***Plugins*:** No entendimento que um *plugin* de um *framework* como Cordova é um componente de software, defeitos podem ser

introduzidos por uma camada extra com código de terceiros, levando a aplicação a falhar. Diferentes implementações desses *plugins* podem causar inconsistências entre as configurações, tais como um comportamento não esperado da aplicação, uma chamada à camada nativa sem resposta, travamentos ao retornar da camada nativa para a híbrida e interrupção da aplicação por exceções não tratadas no *plugin*. Dessa forma, importante se faz investigar o nível de confiabilidade do *plugin* por meio de testes de compatibilidade entre as configurações.

- **Segmentos de código específico de plataforma**: O *WebView* é codificado com código nativo específico para cada SO e deve fornecer os mesmos recursos independentemente de plataforma. É de sua responsabilidade a compatibilização e mapeamento das diferentes características e recursos para cada SO suportado, tornado seu processo de uso transparente ao desenvolvedor. Inconsistências inerentes da falta de compatibilização ou mapeamento errôneos devem ser investigados.

Para encontrar tais inconsistências, pode-se considerar o seguinte cenário: a execução de um caso de teste como uma sequência de eventos que pode ser mapeada em um modelo de estados. Essa abordagem consiste em investigar as inconsistências por meio da comparação de modelos extraídos da execução do caso de teste em cada configuração.

Em detalhes, a abordagem consiste na extração de modelos que representam uma funcionalidade da aplicação sob teste *(application under test - AUT)* em execução em um conjunto de dispositivos móveis de múltiplas configurações *C* que precisam ser testados. Para cada configuração $c \in C$, um modelo *Mc* é gerado a partir da execução de um caso de teste $CT = \{e_1, e_2 \ldots e_n\}$, composto por uma sequência de *n* eventos implementadas para o teste de sistema da *AUT*. Uma configuração contida em *C* será selecionada com base em sua popularidade entre usuários e identificada como configuração de referência $c_{ref}$. Consequentemente, o seu modelo $M_{ref}$ também será conhecido como modelo de referência. Durante o teste da configuração $c \in C$, os estados de *Mc* devem ser comparados com os estados de $M_{ref}$.

Uma visão geral da proposta dessa abordagem é ilustrada na Figura 14 e o mecanismo para verificação dos estados entre os modelos é apresentado no formato de pseudocódigo no Algoritmo 1.

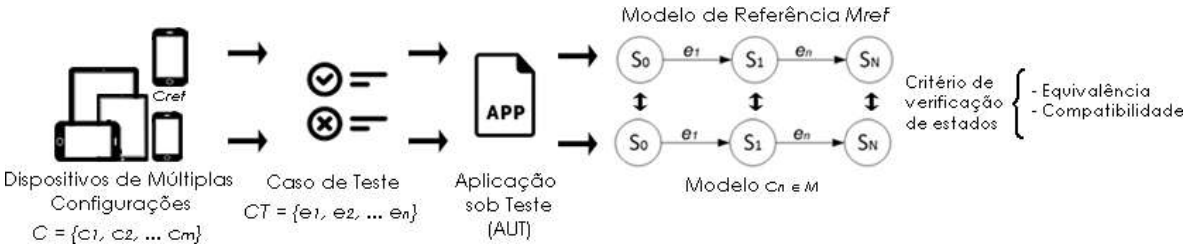


**Figura 14: Visão geral da abordagem para comparação dos modelos entre as múltiplas configurações.**

A verificação entre os estados *S* de $c_i$ e $c_{ref}$ deve considerar alguns critérios: (a) *Equivalência:* estados *S* não equivalentes entre os modelos são indicativos de defeitos de inconsistência; (b) *Compatibilidade:* estados *S* não compatíveis são indicativos de diferença, nesse caso.

**Algoritmo 1: Algoritmo de execução de teste e comparação das configurações**

```
1   input: AUT: Aplicação sob Teste
2          cref: Configuração de referência
3          CT: Caso de Teste
4          ci: Configuração em teste
5          CV: Conjunto de critérios de verificação
6   output: TPDI: Conjunto de Defeitos de Inconsistência
7   {
8       continuar ← true
9       foreach (ei ∈ CT and continuar)
10      {
11          executar(AUT, ci, ei)
12          si ← ObterEstado(AUT, cref, ei)
13          executar(AUT, cref, ei)
14          sref ← ObterEstado(AUT, cref, ei)
15          foreach (cvi ∈ CV and continuar)
16          {
17              if (!AplicarCriterio(cvi, si, sref))
18              {
20                  if (cvi.equivalencia)
21                  {
22                     TPDI.Adicionar("Defeito de inconsistência", si, cvi)
23                     continuar = false
24                  }
25                  else if (cvi.compatibilidade)
26                  {
27                     TPDI.Adicionar("Indicador de incompatibilidade", si, cvi)
28                     continuar = true
29                  }
30              }
31          }
32      }
33      return TPDI
34  }
```

Toda essa investigação será pautada na literatura acadêmica existente com também no meio industrial por meio de consulta à repositórios de projetos de aplicação híbridas, comentários de desenvolvedores e usuários, além de relatos de defeitos mantidos por ferramentas de *BugTracker*. Um rol de 7.167 aplicações híbridas foi identificado na literatura (Georgiev; Jana; Shmatikov, 2014), podendo ser adotado como base para um estudo exploratório.

## 4.1 ETAPAS DO PROJETO E CRONOGRAMA

As etapas que foram e serão desenvolvidas ao longo do trabalho estão listadas a seguir em ordem cronológica e de acordo com a identificação da atividade apresentada no cronograma da Tabela 5.

1. **Cursar disciplinas:** Cursar as disciplinas exigidas pelo programa de mestrado;
2. **Revisão Bibliográfica:** Realizar uma revisão bibliográfica da literatura e participação da elaboração de um MS liderado pelo orientador, objetivando identificar trabalhos relevantes na área de pesquisa e que contribuam para este estudo.
3. **Investigação sobre teste em aplicações híbridas:** Condução de estudo exploratório para avaliar o processo de teste automatizado em aplicações móveis híbridas.
4. **Escrita de artigo:** Com base nos resultados da etapa anterior, foi elaborado um artigo. Os resultados foram publicados no *XLII Latin American Computing Conference (CLEI 2016)* e apresentado pelo candidato em 11 de outubro de 2016 (Menegassi; Endo, 2016);
5. **Elaboração de uma abordagem para teste em aplicações móveis híbridas:** Investigar soluções para responder as QP1 e QP2, oferecendo subsídios para definição de uma abordagem para testes em aplicações móveis híbridas.

6. **Desenvolvimento da ferramenta de apoio:** Empregar a abordagem definada na atividade anterior em uma ferramenta para apoio ao teste de aplicações móveis.
7. **Estudo experimental:** Um estudo de caso será conduzido para avaliar a abordagem e seu emprego na ferramenta de apoio.
8. **Redação do texto da dissertação:** A escrita do trabalho final será iniciada com a ampliação da investigação sobre teste em aplicações híbridas no início do cronograma, passando pelo desenvolvimento e avaliação da abordagem e da ferramenta de apoio, e concluída juntamente com a escrita da coleta e análise de resultados;
9. **Proficiência em inglês:** Estudo da língua inglesa para posterior realização do exame de qualificação exigido pelo programa;
10. **Defesa do mestrado:** Apresentação e defesa do mestrado;
11. **Escrita de artigos:** Paralelo ao desenvolvimento do texto da dissertação, pretende-se escrever novos artigos científicos relatando os achados.

**Tabela 5: Cronograma de atividades**

| Atividades | Meses 2016 | | | | | | | | | | | | Meses 2017 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 |
| 1 | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | |

# REFERÊNCIAS


ABLESON, W. F. et al. **Android em Ação**. 3ª ed. [s.l.] Campus, 2012.

ALASDAIR, A. **Aprendendo Programação - iOS**. [s.l.] Novatec, 2013.

AMALFITANO, D. et al. **Testing Android Mobile Applications: Challenges, Strategies, and Approaches**. [s.l.] Elsevier Inc., 2013. v. 89

AMALFITANO, D. et al. MobiGUITAR Automated Model-Based Testing of Mobile Apps. **Software, IEEE**, v. 32, n. 5, p. 53–59, 2015.

APPIUM. **appium**. Disponível em: <http://appium.io/>. Acesso em: 3 mar. 2016.

ASIKAINEN, T.; MANNISTO, T.; SOININEN, T. A unified conceptual foundation for feature modelling. **10th International Software Product Line Conference (SPLC'06)**, n. August, p. 31–40, 2006.

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.

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.

BOOCH, G. **Software components with ada structures, tools, and subsystems**. 2. ed. [s.l.] Benjamin-Cummings, 1987.

BOUQUET, F. et al. **Extending the Unified Process with Model-Based Testing**. MoDeVa'06, 3rd Int. Workshop on Model Development, Validation and Verification. **Anais**...Genova, Italy: 2006Disponível em: <https://hal.archives-ouvertes.fr/hal-01222822>

BOUQUET, F. et al. **A Subset of Precise UML for Model-based Testing**. Proceedings of the 3rd International Workshop on Advances in Model-based Testing. **Anais**...: A-MOST '07.New York, NY, USA: ACM, 2007Disponível em: <http://doi.acm.org/10.1145/1291535.1291545>

BOUSHEHRINEJADMORADI, N. et al. **Testing Cross-Platform Mobile App Development Frameworks**. Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference. **Anais**...nov. 2015

CALABASH. **Calabash**. Disponível em: <http://calabash.io/>. Acesso em: 29 out. 2016.

CAMDEN, R. K. **Apache Cordova in Action**. [s.l: s.n.].

CARRETERO, J. C. T. **MOBILE TEST AUTOMATION WITH APPIUM + PAGE-OBJECT**. Disponível em: <https://lab.getbase.com/mobile-test-automation-with-appium-page-object/>. Acesso em: 26 nov. 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.

CHOUDHARY, S. R. Detecting cross-browser issues in web applications. **2011 33rd International Conference on Software Engineering (ICSE)**, p. 1146–1148, 2011.

CORDOVA. **Cordova**. Disponível em: <https://cordova.apache.org/>. Acesso em: 7 set. 2016.

CORRAL, L.; SILLITTI, A.; SUCCI, G. Mobile Multiplatform Development: An Experiment for Performance Analysis. **Procedia Computer Science**, v. 10, p. 736–743, 2012.

CORTEZ, D. **MOBILE APP DEVELOPMENT: NATIVE VS. HYBRID VS. MOBILE WEBSITES**. Disponível em:

<http://www.uptopcorp.com/post/mobile-app-development-native-vs-hybrid-vs-mobile-websites>. Acesso em: 22 out. 2016.

COULOURIS, G. et al. **Distributed Systems: Concepts and Design**. 5. ed. [s.l.] Addison-Wesley, 2012.

DALLMEIER, V. et al. **WebMate: A tool for testing Web 2.0 applications**. Proceedings of the Workshop on JavaScript Tools, JSTools'12. **Anais**...2012Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84864030699&partnerID=40&md5=6c8dc2785d472499ff1a73ad96158910>

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: [s.n.].

DEITEL, P. J.; DEITEL, H. M. **Ajax, Rick Internet Applications e desenvolvimento Web para programadores**. São Paulo: [s.n.].

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Introdução ao Teste de Software. In: [s.l.] Elsevier, 2007.

EL-FAR, I. K.; WHITTAKER, J. A. Model-Based Software Testing. In: **Encyclopedia of Software Engineering**. [s.l.] John Wiley & Sons, Inc., 2002.

ENDO, A. T. **Model based testing of service oriented applications**. [s.l.] USP, 2013.

ENDO, A. T. ; et al. **Testing mobile applications: a systematic mapping**. Em elaboração: [s.n.].

FILHO, L. C. Q. **Desenvolvendo seu primeiro aplicativo Android**. 1ª ed. [s.l.] Novatec, 2014.

FLANAGAN, D. **Javascript: O guia definitivo**. 6ª ed. [s.l.] BOOKMAN, 2013.

FOWLER, M. **PageObject**. Disponível em: <http://martinfowler.com/bliki/PageObject.html>. Acesso em: 20 nov. 2016.

GAMMA, E. et al. **Padrões de Projetos: Soluções Reutilizáeis de software orientado a objetos**. [s.l.] BOOKMAN, 2000.

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

GARTNER. **Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016**. Disponível em: <http://www.gartner.com/newsroom/id/3323017>. Acesso em: 26 ago. 2016.

GEORGIEV, M.; JANA, S.; SHMATIKOV, V. Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks. **Ndss**, v. 2014, p. 1–15, 2014.

GRIEBE, T.; GRUHN, V. A model-based approach to test automation for context-aware mobile applications. **Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14**, p. 420–427, 2014.

GRIESKAMP, W. et al. Model-based quality assurance of protocol documentation: tools and methodology. **Software Testing, Verification and Reliability**, v. 21, n. 1, p. 55–71, 2011.

GRONLI, T.-M.; GHINEA, G. **Meeting Quality Standards for Mobile Application Development in Businesses: A Framework for Cross-Platform Testing**. 2016 49th Hawaii International Conference on System Sciences (HICSS). **Anais**...IEEE, jan. 2016Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7427895>

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.

HIERONS, R. M. et al. Using Formal Specifications to Support Testing. **ACM Comput. Surv.**, v. 41, n. 2,

p. 9:1--9:76, 2009.

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

IBM. **Native, web or hybrid mobile-app development**. [s.l: s.n.]. Disponível em: <ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf>.

IDC. **Smartphone OS Market Share, 2015 Q2**. Disponível em: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em: 1 jan. 2016.

IEEE. **IEEE Standard Glossary of Software Engineering Terminology**. [s.l: s.n.]. v. 121990

IONIC. **IONIC**. Disponível em: <http://ionicframework.com/>. Acesso em: 7 set. 2016.

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

JOORABCHI, M. E.; ALI, M.; MESBAH, A. **Detecting inconsistencies in multi-platform mobile apps**. 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). **Anais**...IEEE, nov. 2015Disponível em: <http://ieeexplore.ieee.org/document/7381838/>

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

KANG, K. C. et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. **Distribution**, v. 17, n. November, p. 161, 1990.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. [s.l: s.n.].

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. [s.l.] Novatec, 2015.

LECHETA, R. R. **Desenvolvendo para iPhone e iPad**. 4. ed. 2016: Novatec, 2016.

LEE, S.; DOLBY, J.; RYU, S. HybriDroid : Analysis Framework for Android Hybrid Applications. **ASE '16 (31st IEEE/ACM International Conference on Automated Software Engineering)**, p. 250–261, 2016.

LOPES, S. **Aplicações mobiles híbridas com Cordova e PhoneGap**. [s.l.] Casa do Código, 2016.

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

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, 2015b.

MCKEEMAN, W. M. Differential Testing for Software. **Digital Technical Journal**, v. 10, n. 1, p. 100–107, 1998.

MENEGASSI, A. A.; ENDO, A. T. An evaluation of automated tests for hybrid mobile applications. 2016.

MICROSOFT. **O que é um aplicativo da Plataforma Universal do Windows (UWP)?** Disponível em: <https://msdn.microsoft.com/windows/uwp/get-started/whats-a-uwp>. Acesso em: 31 ago. 2016a.

MICROSOFT. **Desenvolvimento na plataforma Windows universal**. Disponível em: <https://www.visualstudio.com/pt-br/vs/universal-windows-platform/>. Acesso em: 23 out. 2016b.

MILANI, A. **Programando para iPhone e IPad**. 2. ed. [s.l.] Novatec, 2014.

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

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

MUCCINI, H.; DI FRANCESCO, A.; ESPOSITO, P. **Software testing of mobile applications: Challenges and future research directions**. 2012 7th International Workshop on Automation of Software Test (AST). **Anais**...IEEE, jun. 2012Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6228987>

MYERS, G. J. **The Art of Software Testing**. Second Edi ed. [s.l.] John Wiley, 2004.

NAGAPPAN, M.; SHIHAB, E. Future Trends in Software Engineering Research for Mobile Apps. **Saner'15**, 2015.

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.

OLIVEIRA, R. A. P. DE. **Apoio à automatização de oráculos de teste para programas com interfaces gráficas**. [s.l.] USP - São Carlos, 2012.

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

PHONEGAP. **PhoneGap**. Disponível em: <http://phonegap.com/>. Acesso em: 7 set. 2016.

PRETSCHNER, A.; PHILIPPS, J. 10 Methodological Issues in Model-Based Testing. In: BROY, M. et al. (Eds.). . **Model-Based Testing of Reactive Systems: Advanced Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 281–291.

SAAD, N. H.; BAKAR, N. S. A. A. **Automated testing tools for mobile applications**. The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M). **Anais**...IEEE, nov. 2014Disponível em: <http://ieeexplore.ieee.org/document/7020665/>

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.

SENCHA. **Sencha**. Disponível em: <https://www.sencha.com/>. Acesso em: 17 mar. 2016.

SHACKLES, G. **Construindo aplicativos móveis com C#**. [s.l.] Novatec, 2012.

SILVA, E. et al. Computação Ubíqua – Definição e Exemplos. **Revista de Empreendedorismo, Inovação e Tecnologia**, v. 2, n. 1, p. 23–32, 30 jun. 2015.

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.

STAROV, O. et al. Testing-as-a-Service for Mobile Applications: State-of-the-Art Survey. **Dependability Problems of Complex Information Systems**, v. 307, n. April 2016, p. 91–110, 2015.

STATISTA. **Number of apps available in leading app stores as of July 2015**. Disponível em: <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores>.

Acesso em: 1 mar. 2016.

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

STEVEN, A. **What are the best mobile testing tools?** Disponível em: <https://www.quora.com/What-are-the-best-mobile-testing-tools>. Acesso em: 17 out. 2016.

SZYPERSKI, C. **Component software: beyond objectoriented programming**. New York: ACM Press, 1997.

TANENBAUM, A. S.; BOS, H. **Sistemas Operacionais Modernos**. 4ª ed. São Paulo: [s.n.].

TAO, C.; GAO, J. **Modeling Mobile Application Test Platform and Environment: Testing Criteria and Complexity Analysis**. Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing. **Anais**...: JAMAICA 2014.New York, NY, USA: ACM, 2014Disponível em: <http://doi.acm.org/10.1145/2631890.2631896>

TELERIK. **AppBuilder**. Disponível em: <http://www.telerik.com/platform/appbuilder>. Acesso em: 7 set. 2016.

UTTING, M.; LEGEARD, B. **Practical Model-Based Testing: A Tools Approach**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

VICENTE, G. **PAGE OBJECTS – PADRÃO DE PROJETO PARA ORGANIZAÇÃO DE TESTES FUNCIONAIS**. Disponível em: <http://dextra.com.br/pt/page-objects-padrao-de-projeto-para-organizacao-de-testes-funcionais/>. Acesso em: 20 nov. 2016.

W3C. **W3C**. Disponível em: <https://www.w3.org/>. Acesso em: 17 mar. 2016a.

W3C. **What is the Document Object Model?** Disponível em: <http://www.w3.org/TR/DOM-Level-3-Core/introduction.html>. Acesso em: 25 set. 2016b.

W3C. **CSS3**. Disponível em: <https://www.w3.org/TR/CSS/>. Acesso em: 17 jun. 2016c.

WANG, Y.; ALSHBOUL, Y. Mobile security testing approaches and challenges. **2015 First Conference on Mobile and Secure Services (MOBISECSERV)**, n. February 2015, p. 1–5, 2015.

WARGO, J. M. **Apache Cordova 3 - Programming**. [s.l.] Pearson Education, 2013.

WASSERMAN, A. I.; FOSSER. Software Engineering Issues for Mobile Application Development. **FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research**, p. 397–400, 2010.

WEISER, M. **The Computer for the 21st CenturyScientific American**, 1991.

WIERINGA, R. et al. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. **Requirements Engineering**, v. 11, n. 1, p. 102–107, 2006.

WOHLIN, C. et al. **Experimentation in Software Engineering**. [s.l.] Springer, 2012.

XANTHOPOULOS, S.; XINOGALOS, S. **A comparative analysis of cross-platform development approaches for mobile applications**. Proceedings of the 6th Balkan Conference in Informatics on - BCI '13. **Anais**...New York, New York, USA: ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2490257.2490292>

XDK, I. **Intel XDK**. Disponível em: <https://software.intel.com>. Acesso em: 7 set. 2016.

YAN, D.; YANG, S.; ROUNTEV, A. **Systematic testing for resource leaks in Android applications**. 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE). **Anais**...nov. 2013

ZEIN, S.; SALLEH, N.; GRUNDY, J. A systematic mapping study of mobile application testing techniques. **Journal of Systems and Software**, v. 117, p. 334–356, abr. 2016.

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