Universidade Federal de Pernambuco

Centro de Informática

Departamento de Engenharia da Computação

Bacharelado em Engenharia da Computação

Estudo exploratório sobre ferramentas de testes para aplicações Android

Raissa do Rego Barros Xavier de Morais

Trabalho de Graduação

Recife

2018

Universidade Federal de Pernambuco
Centro de Informática
Departamento de Engenharia da Computação

Raissa do Rego Barros Xavier de Morais

Estudo exploratório sobre ferramentas de testes para aplicações Android

Trabalho apresentado ao Programa de Bacharelado em Engenharia da Computação do Departamento de Engenharia da Computação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Adriano de Augusto Moraes Sarmento*

Recife
2018

Estudo exploratório sobre ferramentas de testes para aplicações Android

Raissa do Rego Barros Xavier de Morais

Aprovada em ____/____/_____.

BANCA EXAMINADORA

_____

Adriano Augusto de Moraes Sarmento

_____

Juliano Manabu Iyoda

CONCEITO FINAL: _____

Agradecimentos

Agradeço a minha família por todo apoio ao longo dos anos. Agradeço também ao meu orientador Adriano Sarmento pelo apoio durante o curso e desenvolvimento deste trabalho.

Resumo

A prática de testes durante o desenvolvimento de um aplicativo é algo comum entre programadores e essencial para manter a qualidade do software que está sendo desenvolvido [1]. A diferença entre usuários de software para Desktop ou Web, e aqueles que fazem uso de aplicativos para dispositivos móveis está na rápida interação do usuário com o software, resultando assim em um tempo ainda menor para que o aplicativo capture a atenção do usuário [2]. Este fator reforça a importância de um aplicativo que possua uma GUI (*Graphical User Interface*) e UX (*User Experience*) capaz de prover uma boa interação entre o usuário e o aplicativo. A necessidade de realizar testes se torna ainda maior para aplicativos Android devido ao vasto número de dispositivos disponíveis no mercado [3].

Este trabalho tem como objetivo investigar abordagens de testes disponíveis no mercado para plataforma Android e realizar uma análise das ferramentas de automação de testes existentes para testes caixa-preta. Adicionalmente, através de mineração de repositórios, observaremos como aplicações open source existentes fazem uso (ou não) destas ferramentas. Com base nestes dois estudos, espera-se extrair informações relevantes sobre o uso de tais ferramentas no desenvolvimento de aplicações móveis.

Palavras chaves: Teste de software, Teste de aplicativos móveis, aplicativos Android.

Abstract

Testing during an app development is a common behaviour amongst programmers and essential to maintaining the quality of the software being developed [1]. The difference between users of Desktop or Web software, and those who use mobile apps is in the fast user-software interaction, resulting in a considerably less time for the app to capture the user's attention [2]. This point emphasizes the importance of an application that possesses a GUI (Graphical User Interface) and UX (User Experience) capable of providing a good interaction between user and app. The need for testing is even greater for Android applications due to the large amount of devices available on the market [3].

The purpose of this monograph is to investigate testing approaches available on the market for the Android platform and analyse existing automated black-box testing frameworks. Furthermore, through repository mining we will observe how open source applications use or not those applications. Based on these two studies, we hope to extract relevant information about the use of these tools in mobile app development.

Keywords: Software Testing, Mobile Testing, Android applications.

Lista de Figuras

Lista de Tabelas

Sumário

1. Introdução

Este capítulo tem como propósito expor as motivações que levaram ao desenvolvimento do projeto, os objetivos esperados e uma descrição de como será estruturado o restante deste documento.

1.1 Motivação

No final da década de 1950 e início da década de 1960 programadores não interagiam diretamente com os dispositivos responsáveis por rodar seus programas. Eles entregavam seus programas para técnicos que então os processavam junto de diversos outros, tendo os resultados apenas horas depois. Devido ao fato dessas tarefas necessitarem de poucas interações a cada repetição, seu uso era mais voltado para computações matemáticas. Dois exemplos seriam a primeira linguagem de programação amplamente utilizada FORTRAN (*Formula Translation*) [31], desenvolvido pela IBM e COBOL [32] (*Common Business Oriented Language*), desenvolvido pela CODASYL. Com a evolução do modelo de processamento desses programas para sistemas interativos, houve um rápido crescimento no desenvolvimento e uso de aplicações computacionais [4].

Com os avanços da tecnologia nas últimas décadas, o uso de dispositivos computacionais se tornou bastante comum. Dentre eles, o número de dispositivos móveis, em específico smartphones se tornou considerável. Tais dispositivos trouxeram novos sistemas operacionais como *Android da Google* [33], *iOS* da *Apple* [34] e *Windows Phone* [35] sendo *Android* a primeira plataforma *open source* [5].

Em 2008 o primeiro celular comercial com sistema operacional Android, desenvolvido pela Google, o T-Mobile G1, foi colocado no mercado. Com o passar dos anos a plataforma Android continuou a crescer, tendo aplicativos desenvolvidos para relógios, TVs, carros, entre outros dispositivos [7]. Usuários que fazem uso de aplicativos para dispositivos móveis estão interessados na rápida interação com o software, resultando assim em um tempo ainda menor, se comparado a usuários Desktop e Web, para que o aplicativo capture a atenção do usuário [2]. Este fator reforça a importância de um aplicativo que possua uma GUI (Graphical User

Interface) e UX (User Experience) capaz de prover uma boa interação entre o usuário e o aplicativo.

A prática de testes durante o processo de desenvolvimento de um programa é essencial para manter a qualidade do software que está sendo desenvolvido e garantir que o resultado do código está de acordo com o que foi projetado para fazer [1]. Além disso, falhas de *software* podem causar impactos econômicos negativos. Um exemplo foi o *bug* do milênio (Y2K), como ficou conhecido, que ocorreu devido a uma falha no código de sistemas de computadores que armazenavam datas com apenas 2 dígitos para o ano. De acordo com [6] "Estimados $300 bilhões foram gastos (metade apenas nos Estados Unidos) para atualizar computadores e programas para se tornarem compatíveis com o *bug* Y2K".

Para aplicativos *Android* que possuem um mercado bastante amplo de dispositivos móveis, surge uma necessidade ainda maior de realizar testes abrangentes que cobrem não apenas diferentes versões deste sistema operacional, mas também uma grande variedade de dispositivos disponíveis, o que pode tornar esses aplicativos mais propensos a falhas [8].

Há diversos desafios associados ao desenvolvimento de aplicativos móveis. O processo deve ser ágil, porém testes manuais requerem tempo [9] pois *testers* precisam realizar manualmente ações para cobrir o maior número de casos de teste possíveis tendo a desvantagem de precisar de interação humana durante todo o período no qual o aplicativo está sendo testado [10]. Para combater esse problema a utilização de testes automatizados é uma solução rentável para desenvolver aplicativos de alta qualidade, robustos e confiáveis além de trazer benefícios como o aumento da eficiência e eficácia dos testes, e um *time to market* mais rápido [9] como pode ser visto na Figura 1.

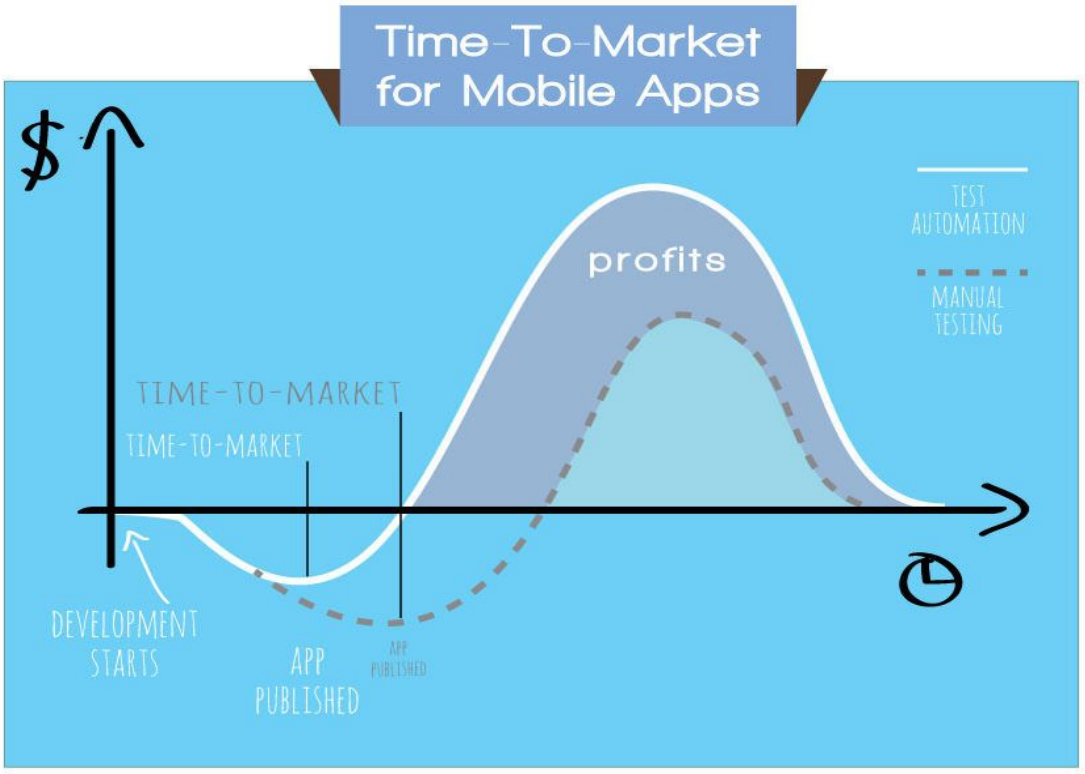

Figura 1. Comparação de *time-to-market* com testes manuais e automatizados [9]

Neste trabalho será realizado um estudo das abordagens de testes disponíveis assim como uma análise de alguns dos *frameworks* de testes mais utilizados para a plataforma *Android,* com objetivo de extrair informações sobre uso dessas ferramentas em projetos *open source.*

1.2 Objetivos

O objetivo deste trabalho é realizar um estudo das abordagens de testes disponíveis para a plataforma *Android* identificando os *frameworks* de testes de automação disponíveis no mercado. Além disso, será feita uma análise de repositórios contendo aplicações *open source* com o propósito de identificar se há ou não implementações de testes utilizando alguma dessas ferramentas e quais são as mais utilizadas.

Como contribuição deste trabalho, temos:

- Uma estudo sobre as abordagens de testes caixa-preta automatizados disponíveis para a plataforma *Android.*
- Análise do uso dos frameworks estudados durante o processo de desenvolvimento de aplicativos *Android*.
- Identificar quais *frameworks* são mais utilizados na comunidade *open source*.

Esperamos com esse estudo mostrar as abordagens de testes disponíveis para a plataforma Android e analisar como projetos open source disponíveis no mercado utilizam testes de UI, mais especificamente, qual importância é dada a esse tipo de testes no processo de desenvolvimento desses projetos.

1.3 Estrutura do documento

Este trabalho está estruturado da seguinte forma: O Capítulo 2 aborda os conceitos básicos de testes de *software* e testes automatizados. No Capítulo 3 é feito um estudo das abordagens de teste disponíveis para aplicativos *Android,* identificando os *frameworks* relacionados a cada uma dessas abordagens, e também é descrita a metodologia utilizada na análise de repositórios *open source* com objetivo de expor os *frameworks* mais utilizados. No Capítulo 4 temos a análise do uso dos *frameworks* de testes de UI em repositórios *open source*. Por fim, no Capítulo 5 se encontram as conclusões obtidas através dos estudos realizados ao longo deste trabalho, assim como possíveis trabalhos futuros.

2. Fundamentação Teórica

2.1 Teste de Software

"Teste de software é um processo, ou série de processos, projetados para garantir que o código faz o que foi projetado para fazer e, de forma recíproca, não realiza nada de maneira não intencional." [1]. Testes são realizados com objetivos de expor os erros de um programa antes que ele chegue na sua etapa final de desenvolvimento. Um conjunto de entradas e saídas esperadas de um programa é conhecido como caso de teste. Mesmo sendo impossível encontrar todos os erros de um programa [1] é imprescindível a elaboração de diversos casos de teste que abrangem o maior número possível de cenários de uso de um programa dentro do tempo disponível para desenvolvimento.

Segundo Myers et al. [1], devido ao tempo e recursos limitados durante o desenvolvimento de software, deve-se estabelecer estratégias de teste antes de dar início ao processo. Duas das estratégias mais utilizadas são:

- ***Black-Box testing:*** Essa estratégia consiste em realizar testes sem considerar a implementação de determinado programa. Entradas e saídas esperadas são o foco principal dessa estratégia, e os dados utilizados nos testes são retirados apenas das especificações do projeto, sem conhecimento sobre a estrutura interna do *software.* Essa estratégia exige o uso de *exhaustive input testing* (teste exaustivo) para que não haja erros, pois não há como garantir que o funcionamento correto sem ter conhecimento sobre a maneira como o programa foi estruturado. Porém o uso de teste exaustivo é impossível pois para testar todos os valores possíveis de entrada e saída de um programa seria necessário um número de casos de teste muito grande.
- ***White-box testing*:** Também conhecido como *logic-driven testing*, essa estratégia permite que o desenvolvedor dos testes tenha acesso a estrutura

interna do programa. Não apenas isso, mas os dados de entrada e saída dos casos de teste são extraídos a partir de uma análise da lógica do programa. Nessa estratégia, para garantir que o programa não contenha erros, os testes devem executar pelo menos uma vez todos os fluxos possíveis do código. Porém, garantir que todos os caminhos de um programa foram executados se torna quase impossível, especialmente para programas muito complexos.

Outro conceito importante é o de **Teste unitário**. Esse tipo de teste consiste na validação de componentes de código para garantir que eles funcionam como esperado [23].

2.2 Aplicativos móveis

"Um Aplicativo móvel é um *software* projetado para rodar em *smartphones, tablets e* outros dispositivos móveis…" [14].

O desenvolvimento de dispositivos móveis contém desafios além daqueles apresentados na implementação de *softwares* convencionais, seja o lançamento de novas versões de sistemas operacionais, novas tecnologias sendo desenvolvidas exclusivamente para tais dispositivos, limitações de *hardware* ou necessidade de liberar um produto em um curto *time-to-market* [14]. Esses fatores mostram a complexidade existente no processo de produção de tais aplicativos.

Há diversos tipos de aplicativos móveis disponíveis no mercado. Segundo Ahmad et al.[11] e Ali et al. [12] , eles podem ser categorizados da seguinte forma:

- **Aplicações Nativas:** Essas aplicações rodam diretamente no sistema operacional do dispositivo. Dessa forma, conseguem utilizar toda a capacidade computacional do sistema no qual estão instalados. No entanto, esses aplicativos precisam ser compatíveis com um grande número de dispositivos disponíveis e necessitam de maior conhecimento técnico e recursos durante seu desenvolvimento. Caso seja necessário desenvolver para sistemas operacionais diferentes, como *iOS* e *Android,* será preciso implementar o programa separadamente para ambas as plataformas.

- **Aplicações Web:** Diferente de aplicações nativas, esse tipo de aplicação é desenvolvido utilizando tecnologias *Web*, podendo ser acessado utilizando qualquer *Web Browser* disponível no dispositivo do usuário. Sendo assim, essas aplicações requerem menor uso de recursos para seu desenvolvimento por poderem ser utilizadas em múltiplas plataformas (*cross-platform*) visto que é necessário garantir a compatibilidade com diferentes *Web Browsers* ao invés de diversos dispositivos e sistemas operacionais. No entanto, essas aplicações não possuem acesso ao *hardware* do dispositivo, o que de certa maneira limita o número de funcionalidades disponíveis, como por exemplo o uso da câmera do dispositivo.
- **Aplicações Híbridas:** Com aplicações híbridas, encontra-se um modelo que inclui os dois citados anteriormente. Esse aplicativos são implementados usando *Cross Platform Tools* (CTP), podendo ser utilizados em diferentes sistemas operacionais, porém sendo instaladas no próprio dispositivo. Dessa forma essas aplicações são consideradas *Web Native-wrapped*. Assim como aplicativos nativos eles podem acessar funcionalidades de *hardware* disponíveis no dispositivo.

2.3 Testes de Aplicativos móveis

Alguns fatores diferenciam os testes de *software* convencionais daqueles realizados em aplicativos móveis. Esses aplicativos devem funcionar em diversos dispositivos diferentes apesar de serem testados apenas em alguns dispositivos específicos. Dessa maneira, fatores como versão do sistema operacional, poder de processamento, *Graphical User Interface* (GUI), consumo de bateria, tamanho de telas, entre outros devem ser levados em consideração não apenas durante o desenvolvimento mas também durante a execução de testes [2] [8].

De acordo com a Smartbear [2], 50% dos usuários apagam um aplicativo se encontrarem pelo menos uma falha, enfatizando a necessidade de produzir um produto que não contenha erros. Além disso, é preciso garantir que o usuário tenha uma boa experiência ao usar o aplicativo, fatores como *design* e facilidade de uso também são importantes.

No contexto de testes de aplicativos móveis, além das estratégias de teste descritas anteriormente (*black-box testing e white-box testing*) pode-se citar: *Usability testing,* que avalia a experiência do usuário durante o uso do aplicativo e *Quality of service testing*, que inclui desempenho, disponibilidade, teste de carga e escalabilidade [8].

De acordo com Rogers and Miles[13], testes exploratórios e *session-based,* sendo este último uma forma estruturada de realizar testes exploratórios em "sessões" onde se espera que os testes sejam executados e reportados de uma maneira específica [21], fazem parte de 85% dos testes realizados por pessoas, e apenas 26% dessas pessoas podem afirmar que 25% de seus testes são automatizados. Apesar de não ser possível substituir completamente testes manuais com testes automatizados, esse último ainda possui papel bastante importante no desenvolvimento de *software.*

Testes automatizados podem ser bastante úteis na realização de testes de regressão, que têm como propósito garantir que funcionalidades previamente implementadas não deixaram de funcionar em etapas futuras do desenvolvimento, além de poder abranger um maior número de dispositivos se comparado a testes manuais.

As diferenças entre os tipos de aplicações móveis também devem ser consideradas ao escrever e realizar testes pois algumas ferramentas de testes automatizados não suportam os 3 tipos de aplicações.

3. Metodologia

O objetivo deste trabalho é investigar abordagens de testes automatizados voltados para plataforma *Android* disponíveis no mercado, além de extrair informações sobre a utilização de ferramentas de teste automatizados em projetos *open source*. Após estudo das abordagens e ferramentas disponíveis, foi decidido realizar a análise dos projetos com foco em testes automatizados de UI (*User Interface*) pois identificar e diferenciar *frameworks de* testes unitários de *frameworks* de UI que utilizam notações de testes unitários traria uma maior dificuldade.

A análise de repositórios foi realizada utilizando as seguinte etapas: Escolha dos *frameworks* que serão incluídos, definição dos critérios de avaliação sobre uso desses *frameworks* durante análise, seleção dos repositórios, extração e análise dos resultados.

3.1 Trabalhos relacionados

O artigo *An analysis of automated tests for mobile Android applications* por Vinícius H.S Durelli [15] realiza uma estudo exploratório do uso de testes automatizados em 25 aplicativos móveis para dispositivos *Android.* A partir desta análise, Durelli identifica quais frameworks estão sendo utilizados, a proporção entre linhas de código de produto e teste, e como esses projetos tratam desafios provenientes de aplicações móveis.

No trabalho *Comparison of GUI testing tools for Android applications* por Tomi Lämsä [22], é desenvolvido um *benchmark* de ferramentas de testes de UI para *Android* a partir da comparação e análise extensiva das ferramentas *Appium, Espresso, Robotium, UIAutomator* e *TAU* seguindo os critérios: velocidade de execução de teste, manutenabilidade do código de teste, confiabilidade e outros. Uma das contribuições do trabalho anterior é ter identificado quais os *frameworks* mais utilizados e de ter proposto os critérios de comparação.

Podemos citar também *A comparative study on automated Android application testing tools*, de Gülçin Hökelekli [28]. Neste trabalho é realizado uma análise comparativa entre as ferramentas *Robotium*, *Appium* e *UiAutomator*.

Diferente do trabalho de Durelli [15] que avalia os repositórios a fim de descobrir quais *frameworks* de teste automatizados estão sendo utilizados, incluindo os de testes unitários, iremos verificar quais projetos utilizam pelo menos um dos *frameworks* de testes automatizados, considerando apenas os utilizados para testes de UI, e extrairemos informações sobre o uso dessas ferramentas. Apesar de focar apenas nos ferramentas de testes UI, iremos avaliar um número maior de repositórios: 1163 em comparação aos 25 avaliados no projeto anterior. Os trabalhos de Lamsa [22] e Hokelekli [28] realizam comparações bastante aprofundadas das ferramentas de teste estudadas em cada projeto, porém não analisam como elas são utilizadas em projetos disponíveis no mercado.

Neste trabalho iremos realizar um estudo sobre os frameworks de testes automatizados disponíveis para a plataforma *Android*, e através da análise de repositórios, extrair informações sobre o uso dessas ferramentas, como quais as mais utilizadas e quantos projetos implementam testes de UI. Para isso serão utilizados 1163 projetos retirados do catálogo de aplicativos *Free and Open Source Software (*FOSS*) Android,* F-droid [16].

3.3 Critérios de análise

Esse estudo foi realizado com os seguinte objetivos:

- Identificar abordagens e ferramentas de testes caixa-preta disponíveis para plataforma *Android*.
- Verificar se e quais entre essas ferramentas estão sendo utilizadas em projetos *open source.*
- Extrair informações sobre o uso dessas ferramentas.

Após identificar as abordagens de testes disponíveis, decidimos realizar a análise de repositórios *open source* com foco no uso de ferramentas de testes UI .

As seguinte questões serão respondidas durante a análise de repositórios:

- **Q1:** Qual a porcentagem de projetos implementam teste de UI?
- **Q2:** Qual porcentagem de utilização de cada framework?
- **Q3:** Quantos projetos utilizam as dependências (adicionadas no arquivo *build.gradle*) de uma ferramenta porém não implementam nenhum teste?

Para responder a **Q1,** iremos avaliar a presença de testes de pelo menos um dos *frameworks Espresso, Robotium* ou *UIAutomator*, selecionados após a identificação de abordagens e ferramentas disponíveis. As Questões **Q2** e **Q3** serão avaliadas para cada framework.

3.4 Análise das abordagens de testes e ferramentas disponíveis para *Android*

As informações sobre abordagens de testes disponíveis foram extraídas a partir da leitura do trabalho de Lämsä [22], dos blogs Saucelabs [18] Bitbar [19] e Tycoon Story [20] e do site oficial da plataforma *Android* [24] e das ferramentas de teste.

Entre as principais abordagens de testes identificadas estão: testes unitários, testes de integração e testes de UI.

Testes unitários tem o propósito de validar o funcionamento de partes menores de código, podendo ser executados isoladamente, sem necessidade de instalar o aplicativo ou integrar com outras partes do código. Estes testes geralmente utilizam *mocks*, objetos que substituem de forma limitada e temporária os objetos reais (pois estes ainda não estão implementados), possibilitando a realização dos testes de maneira isolada e antecipada. Para *Android*, testes unitários podem ser divididos em duas categorias, testes locais e testes de instrumentação [27]. Testes locais são independentes do *framework Android* e estão relacionados ao funcionamento do código Java. Testes de instrumentação no entanto, precisam ser executados em um emulador ou dispositivo físico, pois utilizam componentes específicos da aplicação *Android.* Testes unitários geralmente são mais rápidos se comparado aos outros dois (integração e UI), porém por serem executados separadamente, não podem garantir sozinhos o funcionamento correto da aplicação. Como principais ferramentas de testes unitários, podemos citar:

- **JUnit:** *Framework* de testes unitários para aplicações Java. A plataforma *Android* disponibiliza a classe *AndroidJUnitRunner* que permitir executar testes de

instrumentação nos dispositivos *Android* utilizando JUnit. Esse *framework* é usado como base para diversos outros.

- **Roboletric:** Este *framework* permite que aplicações *Android* sejam executadas sem precisar utilizar um emulador ou dispositivo. Além disso, *Roboletric* usa como base o framework *JUnit,* e consegue gerenciar componentes *Android* como *Views* e *Activities* [25].
- **Mockito:** Um framework com propósito de gerar *mocks.* É compatível com testes unitários em *Android.* Essa ferramenta permite configurar objetos *mock* para facilitar o gerenciamento de dependências externas, garantindo que a interação com tais dependências ocorra da maneira esperada [26].

Testes de integração são responsáveis por validar a integração de componentes da plataforma *Android* como por exemplo *Services* e *ContentProviders.* Esses testes são executados em emuladores ou dispositivos físicos.

A terceira e última abordagem de testes identificada foi a de testes de UI. Após validar o comportamento de pequenas seções/unidades de código individualmente com testes unitários, assim como a integração com os componentes *Android*, o próximo passo é a validação da interface gráfica, ou seja, da interação do usuário com componentes de tela, através de ações como cliques, *scrolls* e *swipes.* Entre as principais ferramentas de teste de UI estão:

- **Appium:** Essa ferramenta *open source* de testes automatizados oferece suporte para aplicações nativas, web e híbridas. Os testes podem ser escritos em diversas linguagens, incluindo Java, Objective-C, Ruby, Python, entre outros. É uma ferramenta *cross-platform*. É um *framework* que utiliza *black-box testing*, sendo assim, não tem acesso ao código da aplicação que está sendo testada. Entre as ferramentas citadas é a mais versátil, pois é compatível com diversas versões da API (*Application Programming Interface*) *Android* e os testes podem ser escritos em várias linguagens. *Appium* utiliza frameworks das próprias plataformas para executar os códigos, ou seja, ao executar uma aplicação *Android,* é utilizado *UI Automator* ou *Selendroid* para interagir com UI [29].

- **Calabash:** Assim como *Appium*, Calabash é um *framework cross-platform,* com suporte para aplicações nativas, web e híbridas, e também utiliza *black-box testing*. Calabash tem suporte para *Cucumber*, uma ferramenta que possibilita a escrita de testes de maneira mais próxima da linguagem natural. Os testes desse *framework* podem ser escritos em *Ruby*.
- **Espresso:** Construído sobre *Android Instrumentation Framework,* utilizado para testes de instrumentação, Espresso é o mais novo entre os três frameworks disponibilizados pela *Google* para testes *Android*, sendo as outras duas *Robotium* e *UI Automator*. Por ser um *framework* nativo, seus testes são escritos na linguagem da aplicação (Java ou Kotlin). Utiliza *white-box testing* e não é *cross-platform*. Não é preciso se preocupar com sincronização e espera de componentes.
- **UI Automator:** Assim como *Espresso*, é uma ferramenta que conta com suporte da *Google* e pode ser utilizada para testes de aplicações nativas. Pode ser utilizada para testar várias aplicações. Os testes são realizados através da interação com componentes de UI disponíveis. Para capturar tais componentes, é utilizada a ferramenta *UI Automator viewer* para análise da hierarquia de layout e acesso a propriedades dos componentes de UI.
- **Robotium:** É uma ferramenta construída com base em *JUnit*, disponível para testes de aplicações nativas, web e híbridas. Pode ser utilizada para testar aplicações onde o código fonte está disponível ou projetos onde se tem o arquivo executável (.apk).

3.5 Seleção das ferramentas de teste

Devido ao número de *frameworks* de testes disponíveis, foram selecionados apenas alguns dos mais utilizados entre as ferramentas de teste de GUI, a fim de cobrir maior parte do uso dessas ferramentas, pela comunidade *open source* [18][19][20][22]. Não foi considerado neste estudo o uso de testes unitários pois alguns *frameworks* de UI utilizam notações de testes unitários (*JUnit*) o que tornaria a análise de arquivos para identificar tais *frameworks* mais complexa. *Espresso, Robotium e UI Automator* serão as ferramentas avaliadas durante a análise de repositórios. As ferramentas *Appium e Calabash* não serão

consideradas pois não são nativas e por isso não se encontram dentro do código fonte da aplicação.

3.6 Extração de dados dos repositórios

O primeiro passo para extração dos dados foi definir os critérios para identificar o uso das ferramentas selecionadas em um projeto *Android.* Em seguida foram extraídos do catálogo *F-Droid* [16] o endereço dos repositórios a serem avaliados. Para isso foi extraído um arquivo XML com os dados de todos os aplicativos armazenados no catálogo. A partir desse arquivo, foram removidos repositórios duplicados e não-existentes restando 1163 repositórios. A análise de repositórios foi realizada utilizando a ferramenta repoDriller [17]. Com essa ferramenta foram baixados os metadados dos 1163 repositórios selecionados. A etapa seguinte consiste em verificar no arquivo ***build.gradle*** a existência da dependência necessária para cada framework, essa informação é utilizada posteriormente para responder a pergunta **Q3 da Seção 3.3.**. Essas dependências podem ser visualizadas na **Tabela 1**.

Espresso	Robotium	UI Automator
com.android.support.test.espresso:espresso-core:*%VersionNumber*	com.jayway.android.robotium:robotium:*%VersionNumber*	com.android.support.test.uiautomator:uiautomator-*%VersionNumber*

Tabela 1. Dependências necessárias para uso das ferramentas no arquivo **build.gradle**

Os repositórios com dependências são avaliados nas etapas seguintes. Esse projetos são analisados para identificar quais contém arquivos **.java** ou .**kt** com indicações de implementações de teste. A **Tabela 2** mostra os critérios utilizados na análise desses arquivos. Esses critérios foram definidos com base na documentação de cada framework, identificando quais requisitos devem estar presentes para implementação de testes. O resultado foi armazenado em um arquivo .csv para extrair as informações que serão apresentadas no capítulo seguinte, assim como as respostas das questões apresentadas.

Espresso	Robotium	UI Automator
Utilização de **OnView** ou **OnData**	Inicialização ou declaração do objeto **Solo**	Inicialização ou declaração dos objetos **UiDevice** ou **UiObject**

Tabela 2. Critérios de identificação da presença de testes em arquivos de código **.java** e **.kt**

4. Análise de Repositórios Open-Source

Neste capítulo serão apresentados os resultados obtidos a partir da análise dos repositórios retirados do catálogo *F-Droid*.

Foi realizada uma análise de 1163 repositórios *open source*, guiada pelas questões apresentadas no capítulo anterior, investigando o uso das ferramentas de testes *Espresso, Robotium* e *UI Automator*.

Rótulo de linha	Número de repositórios
Não utiliza nenhuma das 3 ferramentas de testes UI	1084
Utiliza dependência de pelo menos 1 ferramenta	236
Utiliza pelo menos 1 ferramenta de teste UI	79
Espresso (Não exclusivo)	71
Robotium (Não exclusivo)	15
UiAutomator (Não exclusivo)	13
Totais	1163

Tabela 3. Uso de ferramentas de testes UI em repositórios *open source*.

Na **Tabela 3** temos os dados de repositórios que utilizam dependências de pelo menos 1 ferramenta, porém não implementam nenhum teste de UI, com 236 repositórios e temos 79

repositórios que adicionam as dependências de pelo menos 1 ferramenta e contém testes de UI.

4.1 Implementação de testes automatizados de UI

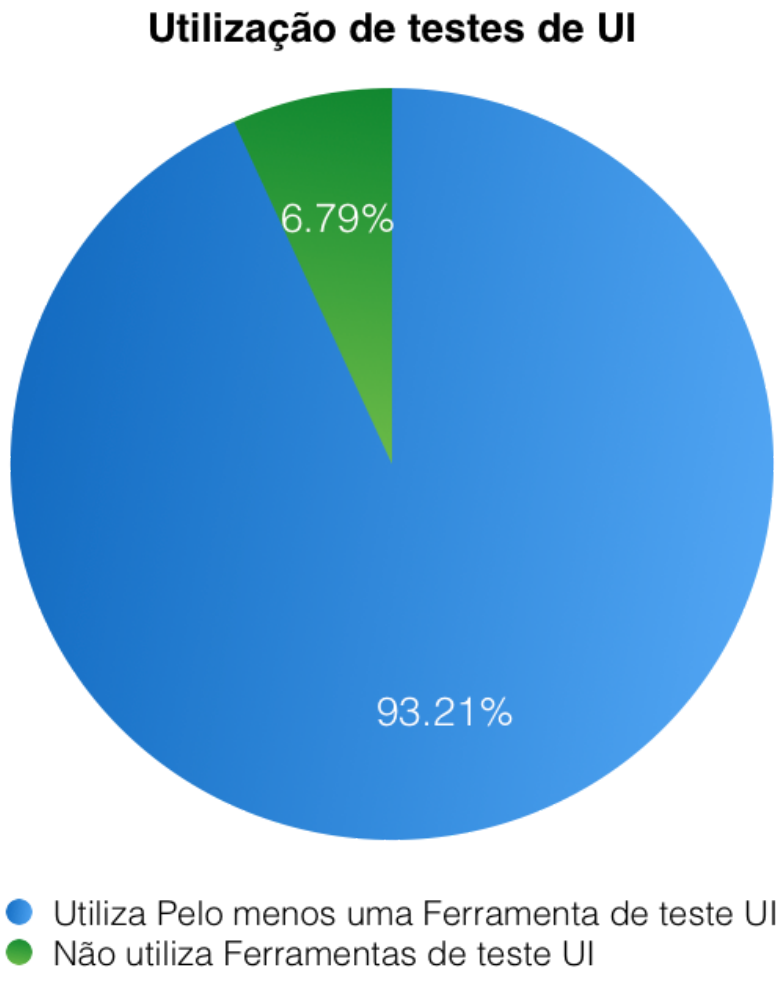

Figura 2. Gráfico de utilização de testes UI

Analisando os dados da **Tabela 3** em forma de gráfico (**Figura 2)** com as porcentagens de uso de ferramentas de testes de UI, podemos verificar que apenas 79 ou 6,79% dos projetos apresentam implementação de testes de UI.

Desta maneira, pode-se dizer que projetos *open source Android* ainda possuem um número baixo de implementação de testes de UI. Sendo *Android* uma plataforma com um mercado bastante abrangente, tendo dispositivos com diversas resoluções, tamanhos e

fabricantes, é importante realizar testes de UI para garantir que o aplicativo projetado funcione no maior número de dispositivos possíveis.

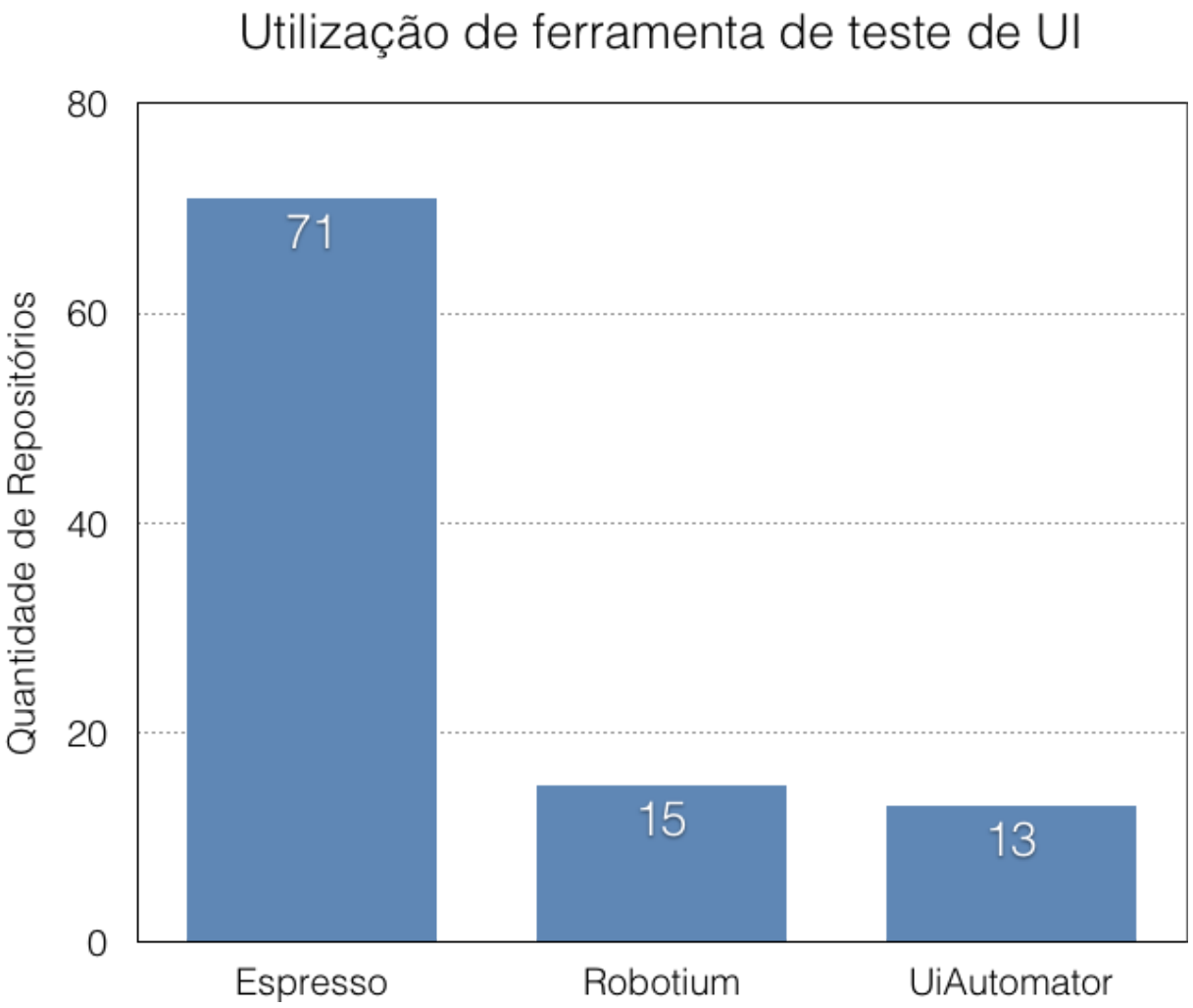

Figura 3. Gráfico de utilização de ferramentas de testes UI.

Na **Figura 3** podemos observar que entre os 79 projetos que utilizam testes de UI, a ferramenta *Espresso* é a mais popular, estando presente em 71 dos projetos. É importante informar que o uso dessas ferramentas apresentado na figura acima não é exclusivo, sendo assim, um projeto que utiliza a ferramenta *Espresso* pode também estar utilizando as ferramentas *Robotium* ou *UI Automator*.

Analisando os resultados obtidos na **Figura 3**, podemos observar que entre as ferramentas de testes de UI mais populares, *Espresso* já se estabeleceu como a mais utilizada, sendo 4 ou 5 vezes mais comum que *Robotium* e *UI Automator*.

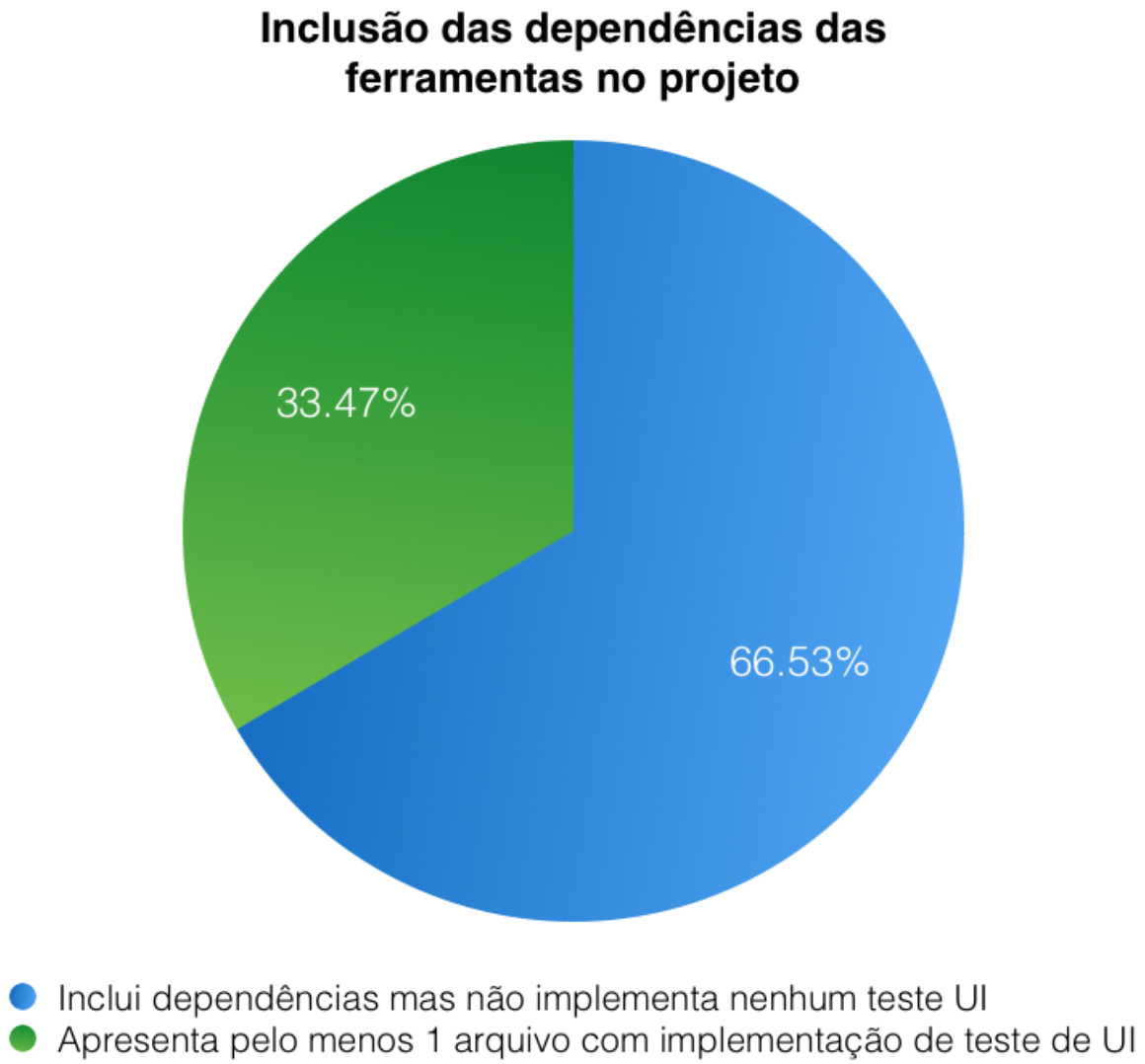

Figura 4. Gráfico de utilização ferramentas de testes UI e inclusão de suas dependências

Ao analisar a **Figura 4** podemos verificar que dos 236 projetos que incluíram as dependências de uma ou mais das ferramentas analisadas, ou seja adicionaram uma das ferramentas no arquivo **build.gradle** do projeto, pelo menos 66.53% não implementaram nenhum teste de UI. Com base nesses dados podemos verificar que alguns fatores podem ter influenciado a não implementação de testes de UI, como por exemplo o curto *time-to-market*, ou dificuldade de uso dos frameworks analisados neste estudo. Outra possibilidade é ter sido utilizada outra ferramenta de testes que não foi incluída neste estudo como *Appium ou Calabash.*

5. Considerações Finais

Este trabalho foi realizado nas seguinte etapas: Estudo de abordagens de testes disponíveis para a plataforma *Android,* identificação e análise dos *frameworks* de testes de automação disponíveis e, por último, uma análise do uso dessas ferramentas em projetos open source. Três tipos de abordagens foram identificadas, sendo elas: testes unitários, testes de integração e testes de UI. Para testes unitários, analisamos as ferramentas *JUnit, Roboletric e Mockito*, e para testes de UI temos *Appium, Calabash, Espresso, UI Automator e Robotium.*

Durante a etapa de análise de repositórios, foram selecionados os *frameworks* de testes de UI *Espresso, UI Automator e Robotium*. *Appium* e *Calabash* não foram incluídos por não serem nativos da plataforma *Android* e não se encontrarem dentro do código do projeto. Em seguida foram definidos os critérios de análise dos repositórios e como seria feita a coleta dos dados.

A partir dos dados obtidos, pudemos perceber ao final da coleta de dados nos projetos *open source,* incluindo 1163 repositórios, que o uso das ferramentas de testes de UI estão presente em apenas 6.79% dos projetos analisados, um valor consideravelmente baixo. Entre as três ferramentas estudadas, *Espresso* se mostrou a mais utilizada, aparecendo em torno de 5 vezes mais do que *Robotium* e *UiAutomator*. Outro critério de avaliação foi o uso de dependências de pelo menos um dos *frameworks*, porém sem de fato implementar testes automatizados. Entre os 236 projetos que incluíram dependências de um dos *frameworks,* apenas 33.47% de fato, mostraram indícios de utilizarem testes de UI. A inclusão dessas dependências podem ser um fator indicativo do planejamento de utilização dessas ferramentas.

Com base no estudo realizado ao longo deste trabalho, podemos concluir que apesar da importância da utilização de testes automatizados, em específico testes de UI, no desenvolvimento de aplicações *Android,* o uso de ferramentas de testes de UI é

consideravelmente baixo. Mesmo a ferramenta *Espresso* que é considerada uma entre as duas ferramentas de teste de UI mais populares não alcançou ao menos 7% de uso. Como foi realizado apenas análise do código e não foi possível incluir as ferramentas *Appium e Calabash*, é possível que este número seja maior do que o obtido se considerarmos outras ferramentas.

Como sugestão para trabalhos futuros podemos citar: Análise de repositórios considerando *frameworks* de testes unitários ou outras ferramentas de testes de UI.

Referências

[1] Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.

[2] SmartBear, **What is Mobile Testing?**, disponível em https://smartbear.com/learn/software-testing/what-is-mobile-testing/, visitado em 08/06/2018.

[3] Milano, Diego Torres. Android application testing guide. Packt Publishing Ltd, 2011.

[4] Viking Code School, **A Brief History of Software Engineering**, disponível em https://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering, visitado em 08/06/2018.

[5] D. Bernardo Silva, A. T. Endo, M. M. Eler and V. H. S. Durelli, "An analysis of automated tests for mobile Android applications," *2016 XLII Latin American Computing Conference (CLEI)*, Valparaiso, 2016, pp. 1-9.

[6] The Editors of Encyclopaedia Britannica, **Y2K Bug**, disponível em https://www.britannica.com/technology/Y2K-bug, visitado em 08/06/2018.

[7] Android Central, **Android History**, disponível em https://www.androidcentral.com/android-history, visitado em 08/06/2018.

[8] Gao, J., Bai, X., Tsai, W. T., & Uehara, T. (2014). Mobile application testing: A tutorial. Computer, 47(2), 46-55. [6693676]. DOI: 10.1109/MC.2013.445

[9] Ville-Veikko Helppi, **The Basics Of Test Automation For Apps**, Games And The Mobile Web, Smashing Magazine, disponível em

https://www.smashingmagazine.com/2015/01/basic-test-automation-for-apps-games-and-mobile-web/, visitado em 08/06/2018.

[10] P. Liu, X. Zhang, M. Pistoia, Y. Zheng, M. Marques and L. Zeng, "Automatic Text Input Generation for Mobile Testing," 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, 2017, pp. 643-653.

[11] A. Ahmad, C. Feng, M. Tao, A. Yousif and S. Ge, "Challenges of mobile applications development: Initial results," *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, 2017, pp. 464-469.

[12] M. Ali and M. Ali, "Mining and Characterizing Hybrid Apps," presented at the WAMA'16, Seattle, WA, USA, 2016.

[13] G. P. Rogers and P. Miles, "Manual testing's newfound place in the automated testing world," *2015 IEEE AUTOTESTCON*, National Harbor, MD, 2015, pp. 199-202.
doi: 10.1109/AUTEST.2015.7356489.

[14] A. Santos and I. Correia, "Mobile Testing in Software Industry Using Agile: Challenges and Opportunities," *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Graz, 2015, pp. 1-2.

[15] D. Bernardo Silva, A. T. Endo, M. M. Eler and V. H. S. Durelli, "An analysis of automated tests for mobile Android applications," *2016 XLII Latin American Computing Conference (CLEI)*, Valparaiso, 2016, pp. 1-9.

[16] F-Droid, **Catalog of FOSS applications for the Android platform**, disponível em https://f-droid.org/ , visitado em 03/06/2018.

[17] Github, **RepoDriller,** disponível em https://github.com/mauricioaniche/repodriller. , visitado em 03/06/2018.

[18] SauceLabs, **The Top 5 Android UI Frameworks for Automated Testing**, disponível em https://saucelabs.com/blog/the-top-5-android-ui-frameworks-for-automated-testing, visitado em 08/06/2018.

[19] Bitbar, **Top 5 Android testing frameworks with code examples,** disponível em https://bitbar.com/top-5-android-testing-frameworks-with-examples/, visitado em 08/06/2018.

[20] Tycoon Story, **Top 10 testing platforms for Android app developers,** disponível em https://android.jlelse.eu/top-10-testing-platforms-for-android-app-developers-f0ca9993517e, visitado em 08/06/2018.

[21] Satisfice Inc., **Session-Based Test Management**, disponível em http://www.satisfice.com/sbtm/, visitado em 12/06/2018.

[22] Lämsä, T. , "Comparison of GUI testing tools for Android applications", (Master's Thesis), (2017).

[23] Software testing fundamentas, **Unit Testing** , disponível em http://softwaretestingfundamentals.com/unit-testing/, visitado em 12/06/2018.

[24] Android, **Test your app**, disponível em https://developer.android.com/studio/test/, visitado em 12/06/2018.

[25] Roboletric, **Roboletric** , disponível em http://robolectric.org/, visitado em 12/06/2018.

[26] Mockito, **Mockito**, disponível em http://site.mockito.org/, visitado em 12/06/2018.

[27] Stfalcon, **Simple unit tests for Android** , disponível em https://stfalcon.com/en/blog/post/simple-unit-tests-for-android, visitado em 12/06/2018.

[28] Hökelekli, G. ,"A comparative study on automated android application testing tools" (Graduate school), (2016).

[29] Appium, **Appium**, disponível em http://appium.io/, visitado em 12/06/2018.

[30] Calabash, **Calabash**, disponível em https://calaba.sh/, visitado em 12/06/2018 .

[31] Chivers I., Sleightholme J., Introduction to programming with Fortran: With coverage of Fortran 90, 95, 2003, 2008 and 77 . Springer Publishing Company, 2015.

[32] Chivers I., Sleightholme J., Visual COBOL: A developer's guide to modern COBOL. Box Twelve Press, 2017.

[33] Android, **Android**, disponível em https://www.android.com/, visitado em 29/06/2018.

[34] Apple, **iOS 11**, disponível em https://www.apple.com/lae/ios/ios-11/, visitado em 29/06/2018.

[35] Microsoft, **Fim do suporte do Windows Phone 8.1**, disponível em https://www.microsoft.com/pt-br/windows/windows-10-mobile-upgrade, visitado em 29/06/2018.