FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Android Testing

André Rodrigues Barros

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Ana Cristina Ramada Paiva

26 de Julho de 2018

Android Testing

André Rodrigues Barros

Mestrado Integrado em Engenharia Informática e Computação

26 de Julho de 2018

Resumo

Atualmente, a indústria dos *smartphones* é uma das maiores e mais competitivas no mercado e o desenvolvimento de aplicações móveis, naturalmente, acompanham o seu crescimento, especialmente quando se trata da plataforma Android. Como tal, essas aplicações tornaram-se uma parte importante da vida diária da maioria das pessoas. Com o aumento de aplicações disponíveis no Google Play, tornou-se extremamente importante desenvolver aplicações bem organizadas e estáveis. Para desenvolver aplicações móveis com boa qualidade, uma das tarefas mais importantes que os *developers* têm de realizar é o teste de aplicações, que muitas vezes podem ser negligenciados pelas empresas. Embora já existam opções para testes automatizados, o que é a maneira mais barata de realizar esses testes, a maioria dessas opções não é satisfatória.

O objetivo deste trabalho de pesquisa é retomar o desenvolvimento de uma abordagem já iniciada, a ferramenta iMPAcT. Esta ferramenta combina engenharia reversa, comparação de padrões e teste. O objetivo principal é testar o comportamento recorrente (padrões de UI) em aplicações Android, verificando se as diretrizes de desenvolvimento estão a ser seguidas. Para testar os padrões de UI identificados, a ferramenta iMPAcT executa um conjunto de estratégias de teste correspondentes na aplicação de Android em teste. O comportamento que a ferramenta iMPAcT pode testar está guardada num catálogo.

O principal objetivo desta pesquisa é adicionar mais comportamentos a serem testados, estendendo as estratégias de teste da ferramenta iMPAcT.

Em conclusão, com a ferramenta iMPAcT podemos verificar se boas práticas de programação estão a ser seguidas para o desenvolvimento de aplicações Android e, com ela, ajuda as empresas e as empresas focadas em apresentar produtos produtos de qualidade que continuarão a melhorar a nossa vida quotidiana.

Abstract

Nowadays the smartphone industry is one of the biggest and most competitive ones in the market and the development of mobile apps naturally goes side-by-side with it, specially when it comes to the Android platform. As such these apps have become a very important part of most people's daily life. With the increasing of available apps on Google Play, it has become exceedingly important to develop well organized and stable apps. To develop mobile apps with good quality, one of the most crucial tasks developers have to undertake is the testing of said apps, which can often be neglected by the companies. Although there are already options for automated testing, which may be the easiest and cheapest way to perform these tests, most of these options are not enough. Knowing this, the goal of this research work is to resume the development of an approach started and improved before, the iMPAcT tool. This tool combines reverse engineering, pattern matching and testing. The main goal is to test recurrent behavior (UI patterns) on Android applications checking if the development guidelines are being followed. To test the identied UI patterns, iMPAcT tool runs a set of corresponding test strategies over the Android application under test. The behavior that iMPAcT tool is able to test is in a catalog. The main goal of this research is to add more behavior to be tested by extending the test strategies of the iMPAcT tool. In conclusion, with the iMPAcT tool we can check if good programming practices are being followed for the development of Android apps, and, with it help Android companies and developers to deliver quality products that will continue to improve our day-to-day life.

Agradecimentos

Concluída a última fase da vida de estudante não posso deixar de deixar alguns agradecimentos a quem me acompanhou e marcou. Primeiro vem sempre a família e, aqui, quero dar o mais sentido obrigado aos meus pais e avós que me educaram e tornaram a pessoa que hoje sou, nunca conseguiria chegar aqui se não fossem quem são. Á minha irmã, que não sei como, sempre me conseguiu aturar e é a pessoa mais irritante que conheço (com todo o carinho do mundo).

Não seriam agradecimentos se não mencionasse quem me acompanhou lado-a-lado este tempo todo. A quem partilhou comigo estes últimos 5 anos, juro que não existem palavras para descrever os momentos que passámos juntos e tudo que aprendi convosco. Aqui, quero começar por deixar o meu obrigado ao pessoal que me acompanhou na realização desta dissertação: o Nogueira, Poncho e Tikinho.

Deixo, também, um grande obrigado à malta da grande Républica, que, por favor, nunca mudem (excepto o Botelho)!

Um grande abraço para os condutores da Maia que estão sempre à minha procura, um dia ainda me encontram.

Ao meu estimadíssimo Pádrinho Eskilo, uma das pessoas que mais me ensinou, quero dizer que é um prazer enorme chamar-te um grande amigo.

Não me podia esquecer de dar uma palavra especial a quem me acompanhou desde a infância, as melhores pessoas com que alguma vez podia ter passado grande parte da minha vida. A eles quero deixar um enorme obrigado e sei que, onde estivermos, vai sempre haver um tempinho para um café!

Por fim, quero agradecer à pessoa que me orientou nesta jornada que foi a dissertação, a professora Ana Paiva, por todos os conselhos e avisos que guiaram o barco a bom porto.

André Rodrigues Barros

"Don't cry because it's over, smile because it happened"

Dr.Seuss

Conteúdo

1 Introdução **1**
- 1.1 Problema . . . 1
- 1.2 Motivação e Objetivos . . . 2
- 1.3 Estrutura da Dissertação . . . 3

2 Estado da Arte **5**
- 2.1 Evolução e teste de aplicações móveis . . . 5
- 2.2 Estado atual do teste de GUI em aplicações móveis . . . 7
  - 2.2.1 Ferramentas mais utilizadas . . . 7
- 2.3 Testes baseados em modelos . . . 9
- 2.4 Engenharia reversa . . . 9
- 2.5 Testes automáticos baseados em padrões . . . 10
  - 2.5.1 Padrões . . . 10
  - 2.5.2 PBGT - projeto de teste de interface gráfica baseado em padrões . . . 12
- 2.6 Conclusões . . . 13

3 iMPAcT Tool **15**
- 3.1 Abordagem . . . 15
  - 3.1.1 Implementação . . . 15
  - 3.1.2 Configuração de testes . . . 16
  - 3.1.3 Ciclo de deteção de padrões . . . 17
  - 3.1.4 Condições de paragem . . . 18
- 3.2 Catálogo de padrões . . . 18
  - 3.2.1 Padrão *Side Drawer* . . . 18
  - 3.2.2 Padrão de Orientação . . . 19
  - 3.2.3 Padrão de Abas (*Tabs*) . . . 20
  - 3.2.4 Padrão de *Background* . . . 22
  - 3.2.5 Padrão de *Action Bar* . . . 22
  - 3.2.6 Padrão do *Up* . . . 23
  - 3.2.7 Padrão do *Back* . . . 24
- 3.3 Artefactos . . . 24
  - 3.3.1 Relatório . . . 24
  - 3.3.2 Modelo do comportamento observado . . . 26
- 3.4 Conclusões . . . 27

4 Implementação **29**
- 4.1 Melhorias implementadas na *iMPAcT Tool* . . . 29
  - 4.1.1 Padrão *Back* . . . 29

4.1.2 Padrão de Orientação 31
4.2 Padrão de chamadas 32
4.2.1 Estrutura do padrão 32
4.2.2 Comportamento do padrão 33
4.3 Resumo ou Conclusões 35

5 Validação **37**
5.1 Questões de pesquisa 37
5.2 Especificação técnica 38
5.3 Metodologia de teste 38
5.4 Deteção de falhas 40
5.5 Melhorias Implementadas 41
5.5.1 Padrão de Orientação 41
5.5.2 Padrão *Back* 42
5.6 Qualidade dos resultados 43
5.7 Conclusões 44

6 Conclusões e Trabalho Futuro **45**
6.1 Discussão 45
6.1.1 QP1: A *iMPAcT Tool* é capaz de identificar falhas em aplicações Android, tendo em conta o novo padrão acrescentado à ferramenta? 45
6.1.2 QP2: Os padrões melhorados aumentaram a capacidade de identificar falhas nas aplicações sob teste? 45
6.1.3 QP3: As falhas detetadas pela ferramenta são, realmente, falhas da aplicação? 46
6.2 Trabalho Futuro 46
6.3 Conclusões 46

Referências **49**

Lista de Figuras

2.1 Evolução no número de aplicações disponíveis na plataforma Google Play desde Dezembro de 2009 até Dezembro de 2017 6
2.2 Principais desafios para o teste de apps 6
2.3 *Meta-model* da linguagem PARADIGM 13

3.1 Processo da *iMPAcT Tool* [MPF14] 16

4.1 Processo do comportamento do padrão de chamadas 34

Lista de Tabelas

5.1 Aplicações que falharam o teste do padrão de chamadas 41
5.2 Resultados finais do padrão de chamadas 43

Abreviaturas e Símbolos

API	Application
DSL	Domain Specific Language
GUI	Graphical User Interface
PBGT	Pattern-Based GUI Testing
UI	User Interface
MBT	Model-Based Testing

Capítulo 1

Introdução

Nos dias de hoje, os *smartphones* são cada vez mais parte do dia-a-dia de cada individuo, sendo que as suas funções são sendo cada vez mais abrangentes, estendendo-se, por exemplo, até a bancos e a opção de fazer transferências através de aplicações. Considerando o facto de que os *smartphones* se tornaram quase indispensáveis não é surpresa esta industria ser uma das que mais crescimento apresenta, sendo que em 2014 os utilizadores de *smartphones* ficava-se pelos 1.57 biliões com uma previsão aumento de 1.3 biliões, ficando o número final em 2.87 biliões em 2020 [Stab].

Este aumento em utilizadores afeta, naturalmente, o valor do mercado, mostrando um crescimento de cerca de 45% ao longo do período de 2013 até 2017, situando-se, neste momento, em 478.7 biliões de dólares. Neste mercado o Android é a plataforma que se destaca, possuindo uma quota de mercado de 81.7%, mostrando-se, assim, a plataforma mais popular entre os utilizadores [Staa].

Estes números demonstram a crescida importância que os *smartphones* têm na nossa sociedade. Com isso, torna-se fulcral garantir a funcionalidade e segurança das suas aplicações e, para isso, uma das maneiras de o fazer é investindo em testes.

1.1 Problema

De modo a garantir que as suas aplicações são bem sucedidas num mundo cada vez mais povoado por um grande número de aplicações, os *developers* e empresas têm de garantir a qualidade do seu produto. Com as constantes mudanças no mercado e surgimento de novos conceitos de desenvolvimento, um dos maiores desafios neste momento é o teste a essas aplicações. No entanto, empresas presentes neste ramo sentem várias dificuldades neste aspeto, sendo que o tempo é um dos principais fatores destas dificuldades [Stac].

A acrescentar a estas dificuldades, existem também outros fatores que tornam o teste a aplicações móveis uma tarefa complicada. Destes fatores destaca-se a panóplia de diferentes versões

do sistema operativo, que traz, em regra geral, novas funcionalidades e outras deixam de existir, o que torna o ambiente de teste em Android um ambiente volátil. A contribuir para isto, existe, também, uma grande variedade de dispositivos no mercado, com diferentes características, como, por exemplo, o tamanho do ecrã, aos quais os *developers* são obrigados a adaptar-se de modo, a garantir que as suas aplicações têm o mesmo desempenho através dos múltiplos dispositivos e sistemas operativos.

Com esta dificuldade, podemos concluir que a melhor abordagem é a automação dos testes às aplicações, no entanto, um dos problemas apontados como sendo dos mais difíceis de ultrapassar é a falta de uma ferramenta satisfatória para a realização dos testes.

1.2 Motivação e Objetivos

Com a crescente preocupação em desenvolver testes para aplicações de *smartphones* começaram a surgir algumas ferramentas apresentadas como solução. No entanto, apesar de existir esta evolução, em que até a Google criou *frameworks* auxiliares, como o *UIAutomator* que acaba por ser uma das bases que muitas das ferramentas usam, as soluções apresentadas até à data não conseguem cumprir as altas expectativas existentes no que toca ao teste de aplicações da plataforma Android.

Apesar de estas aplicações conseguirem a automação dos testes, estes ainda precisam de ser escritos manualmente por parte dos programadores.

Aqui entra a ferramenta-alvo desta dissertação: a ***iMPAcT tool***. Esta ferramenta foca-se na automatização de testes de interface gráfica para aplicações de Android. O que separa esta ferramenta das demais é o facto de todo o processo de teste ser feito de forma automática, ou seja, quer a geração de testes, quer a execução dos mesmos, é inteiramente feita pela *iMPAcT tool*. Para isto, combina engenharia reversa e identificação de padrões de interface gráfica, de forma a testar comportamentos recorrentes em aplicações.

Neste contexto, os padrões são representações de vários comportamentos habitualmente presentes em aplicações Android. Estes padrões encontram-se, maioritariamente, definidos na documentação da Google sobre boas práticas a desenvolver aplicações para a sua plataforma [Gooa].

Com isto, o objetivo desta dissertação passa por estender esta ferramenta, aumentando o número de padrões presente no seu catálogo de modo a conseguir abranger mais casos de teste.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais quatro capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3, é apresentada a ferramenta que esta dissertação aborda, tal como as metodologias e soluções ao problema. No capítulo 4 são descritos os detalhes dos novos padrões adicionados, tal como o catálogo da ferramenta e as estratégias de teste. No capítulo 6 são apresentados os resultados dos testes desenvolvidos ao longo desta dissertação, assim como a conclusão e trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo é descrito o estado da arte no que diz respeito ao teste de aplicações para Android.

O primeiro assunto a ser abordado é a evolução destas aplicações nos anos mais recentes como visto na secção 2.1.

De seguida, na secção 2.2 é feita uma revisão do estado atual de testes a partir de interfaces gráficas em aplicações móveis, fornecendo uma análise das ferramentas mais utilizadas neste momento.

Na secção 2.4 é explicado a abordagem com recurso a engenharia reversa com o objetivo de simular interações com a aplicação por parte do utilizador.

São descritas as várias soluções que deram origem à automação de testes baseados em padrões na secção 2.5. É analisada a evolução da utilização de padrões como forma de teste, bem como os elementos que os constituem, seguido por uma representação mais formal dos mesmos. A técnica de testes baseados em modelos também é mencionada nesta secção como o primeiro passo na automação de testes. Esta secção termina com a descrição do projeto de teste de interface gráfica baseado em padrões (*PBGT*).

2.1 Evolução e teste de aplicações móveis

Tendo em conta o facto de aplicações para plataformas móveis, como o Android, serem mais vantajosas a nível de preço e dificuldade torna-se mais fácil perceber o grande aumento verificado na figura 2.1.

Com este crescimento quase exponencial, aumenta também a necessidade de produzir aplicações com maior qualidade e, neste âmbito, um dos maiores desafios é o teste às mesmas. Isto é justificado pela grande dificuldade que testar cada elemento do sistema. Com este grande aumento de desenvolvimento de apps, a exigência em relação às ferramentas está por sua vez a aumentar

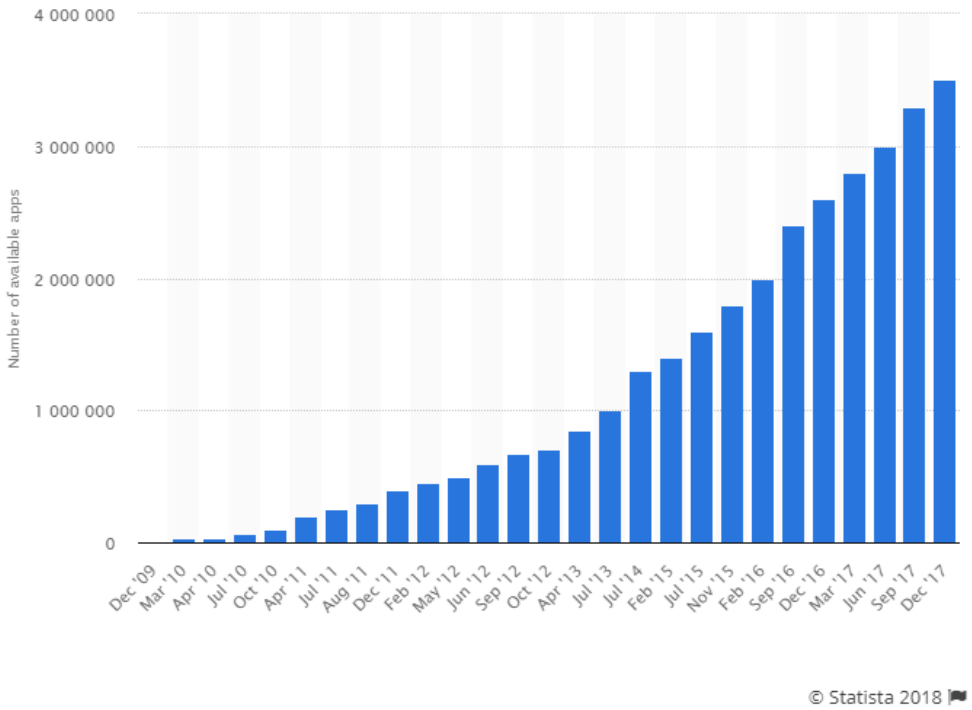

Figura 2.1: Evolução no número de aplicações disponíveis na plataforma Google Play desde Dezembro de 2009 até Dezembro de 2017

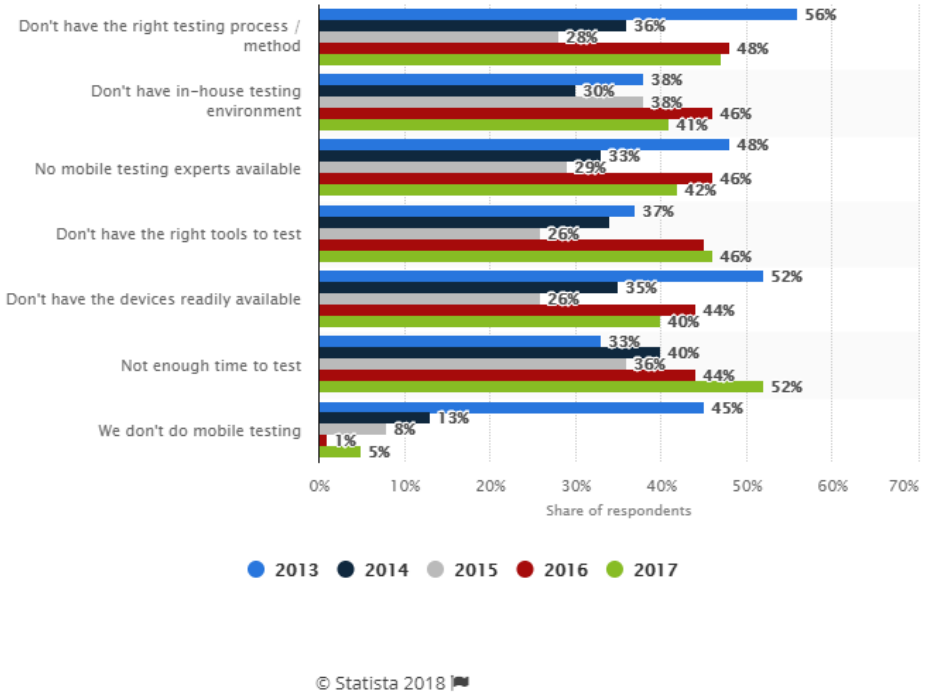

Figura 2.2: Principais desafios para o teste de apps

também, visto que em 2017 46% das respostas apontam para a falta de uma ferramenta satisfatória, o que corresponde a um aumento de 9% comparado a 2013, como podemos ver na figura 2.2,

segundo uma pesquisa feita entre executivos seniores de *IT-management*. [Stac]

2.2 Estado atual do teste de GUI em aplicações móveis

Neste momento, maior parte dos testes efetuados em aplicações requerem que os *developers* escrevam os seus próprios testes, ou seja, não são totalmente automáticos. No geral, a automação de testes através interface gráfica é uma tarefa bastante complexa. A automação tem de simular a interação humana com a interface, usando as *widgets* e analisando a informação apresentada pela GUI. [TKH11]. Na secção 2.2.1 é apresentada uma lista descritiva das frameworks mais utilizadas e populares atualmente.

2.2.1 Ferramentas mais utilizadas

Esta lista apresenta uma descrição das frameworks e ferramentas de teste da interface gráfica mais usadas:

UIAutomator

O UIAutomator é uma framework oficial destinada a testar a interface gráfica fornecida pela Google para a plataforma Android. Esta framework disponibiliza um conjunto de APIs para o desenvolvimento de testes à interface gráfica que tenham como intenção a interação com qualquer aplicação, seja ela de sistema ou não. É a framework indicada para testes de caixa negra, onde o código a ser testado não depende nos pormenores de implementação internos da aplicação.

Os seus principais aspetos são:

- Um visualizador capaz de inspecionar a hierarquia do "layout", proporcionando uma GUI para analisar os elementos da interface gráfica atualmente ativos.
- Uma API para o acesso e realização de operações no dispositivo em que a aplicação-alvo está a ser executada. Com estas operações pode simular a interação de um utilizador com o dispositivo, como carregar no botão "Home".
- Um conjunto de APIs que permitem a escrita de testes mais robustos. Este conjunto de APIs consegue dar à framework a sua capacidade de realizar testes *cross-app* à interface gráfica.

.

Appium

A Appium é uma ferramenta *open-source* que pode ser usada para testar qualquer tipo de aplicação móvel, quer seja nativa, web ou híbrida. Esta ferramenta por sua vez utiliza a framework *UIAutomator* para testes na plataforma Android. Isto permite, com a utilização de *Selenium WebDriver*, uma API de teste web, o desenvolvimento de testes em várias linguagens de programação, como Java, Objective-C, JavaScript, PHP e outras mais.

As suas vantagens são:

- sendo que é multi-plataforma, os mesmos testes resultam sendo realizados em plataformas diferentes.
- o suporte a várias linguagens de *scripting*, aumentando a facilidade de utilização
- o código-original permanece a todo o intacto, ou seja, o código submetido é sempre o código que é testado.

No entanto, também apresenta algumas desvantagens:

- limitações técnicas no que diz respeito à utilização de várias instâncias
- Como utiliza *UIAutomator* para automação em Android, apenas suporta versões mais recentes do que a API 18 da plataforma Android SDK

Espresso

A ferramenta Espresso, disponível desde 2014 na sua versão 2.0 é uma ferramenta oficial disponibilizada pela Google. Esta ferramenta oferece um conjunto de APIs para construir teste à interface gráfica de modo a testar os casos de uso da aplicação. O estilo de teste pode ser caracterizado como *white box testing*, pois tem acesso ao código que, efetivamente, corre a aplicação, o que resulta num teste mais completo a cada elemento.

> "Os principais recursos da estrutura de teste Espresso são:
>
> - APIs flexíveis para ver e adaptar correspondências em aplicativos-alvo. Para obter mais informações, consulte Ver correspondências.
> - Um conjunto extenso de APIs de ação para automatizar interações da IU. Para obter mais informações, consulte APIs de ação.
> - Sincronização de encadeamento de UI para melhorar a confiabilidade dos testes. Para obter mais informações, consulte Sincronização de encadeamento de UI." [Goob]

Robotium

Uma framework *open-source* para automação de testes em Android, usada para desenvolver, principalmente, testes de caixa-negra robustos e poderosos.As suas principais características são:

- Permite ao utilizador escrever casos de teste mesmo tendo conhecimento mínimo da aplicação em teste
- Conjunto de APIs capaz de interagir diretamente com os controlos de interface gráfica contidos dentro da aplicação, como um *TextView*.
- A plataforma Android não sofre qualquer modificação por parte da ferramenta.
- Pode ser integrado facilmente com *Maven* e *Ant*, de modo a ajudar na automação de testes.

- O projeto da aplicação de teste e o projeto da aplicação são executados na mesma JVM.
- Consegue detetar mensagens *Toast* presentes no ecrã do dispositivo.
- É capaz de gerir automaticamente várias atividades numa aplicação.
- A legibilidade dos casos de teste é bastante melhorada comparada com os testes padrão.

2.3 Testes baseados em modelos

A técnica de testes baseados em modelos (MBT) tem como principal objetivo a automação da geração de testes, usando modelos de requerimentos de sistema. Nesta técnica, os casos de teste são gerados a partir de um modelo ou especificação. Neste caso, o modelo representaria o comportamento duma aplicação que seria utilizado para gerar uma série de testes. Apesar de ser uma técnica bastante útil, apresenta duas desvantagens:

- A necessidade de providenciar um modelo da aplicação. que por sua vez requer construção manual, é um processo propício a erros e demorado
- a explosão combinatória de casos de teste

Relativamente ao primeiro problema, a solução pode passar por:

- submeter a aplicação sob teste a um processo de engenharia reversa, que será o foco da próxima secção
- aumentar o nível de abstração do modelo construído

Para além disso o projeto PBGT (revisto mais adiante) diminuí consideravelmente o esforço necessário para construir um modelo.

No que diz respeito ao segundo obstáculo, o foco no teste de comportamentos recorrentes (padrões de comportamento) irá auxiliar a resolução deste problema.

2.4 Engenharia reversa

Engenharia reversa é uma técnica que tem como objetivo obter representações de alto nível de um dado programa. O processo, normalmente, é iniciado com uma representação de baixo nível (ficheiros binários, código-fonte ou traços de execução) e tenta obter representações com uma maior abstração a partir destas (como, para os exemplos já dados, código-fonte, vistas da arquitetura ou casos de uso, respetivamente) [DB05].

No âmbito desta dissertação, esta secção irá abordar apenas engenharia reversa aplicada à plataforma Android.

Ao longo dos últimos anos, têm sido apresentadas várias ferramentas e técnicas que visam tirar o máximo proveito da engenharia reversa, sendo que maior parte destas se foca em obter uma representação da aplicação sob teste (AuT) [Imp, MBN03, AFM12] .

Neste contexto, podemos dividir em três as abordagens da engenharia reversa:

- estática, que visa extrair informação analisando o código-fonte ou o *bytecode* da aplicação;
- dinâmica, que extrai a informação analisando a própria aplicação enquanto é executada;
- híbrida, que exibe o comportamento de ambas as abordagens anteriores.

Sendo que, no que diz respeito à engenharia reversa da interface gráfica, o objetivo é conseguir analisar e simular a interação do utilizador com a aplicação, obtendo um modelo descritivo, é possível a conclusão de que a abordagem estática não é melhor.

Com esta técnica, o principal objetivo passa por obter o código-fonte da aplicação através do *APK (Android Application Package)*, que se trata de um ficheiro executável de Android. Este código-fonte será utilizado para gerar vistas da arquitetura da aplicação sob teste, processo o qual será mais aprofundado no capítulo seguinte.

2.5 Testes automáticos baseados em padrões

Nesta secção são descritas as várias soluções que deram origem aos testes automáticos baseados em padrões, que, por sua vez, deu origem a projetos como a ferramenta relacionada com esta dissertação. Inicialmente, é aprofundada a noção de padrões e os seus vários tipos, também como a relevância no contexto deste trabalho. De seguida, são indicadas as principais abordagens da técnica de testes baseados em modelos. Por fim, é descrito o projeto que levou à ferramenta desta dissertação, o projeto de teste de interface gráfica baseado em padrões.

2.5.1 Padrões

O termo *padrão* é usado para descrever algo que é, ao mesmo tempo, parte de um sistema e uma descrição de como construir essa mesma parte. Padrões normalmente descrevem abstrações de software usados por *designers* e programadores no seu sistema [Ris98].

Este é um tema que, ao longo dos anos, tem sido explorado por vários autores nos seus trabalhos, focando-se, principalmente, na arquitetura de sistemas [GH04, Sha94, DG08].

Segundo o livro escrito por Eric Gamma *et al.* [EGV02], os padrões são composto por quatro elementos:

- **Nome do padrão:** usado para descrever o problema de design, as suas soluções e consequências numa palavra ou duas. Permite uma melhor catalogação dos mesmos.
- **O problema:** descrição de quando utilizar o padrão. Por vezes pode incluir um conjunto de condições que necessitam de se cumprir para a utilização do padrão fazer sentido.

- **A solução:** descreve os elementos que compõem o *design*, as suas relações, responsabilidades e colaborações. Como um padrão pode ser aplicado em várias situações, a solução não se trata de um *design* concreto ou uma implementação, em vez disso, providencia um agrupamento de elementos que, geralmente, resolve o problema.

- **As consequências:** são o resultado da utilização do padrão. Normalmente relacionam-se com questões de tempo ou espaço. Como a reutilização é um fator crítico, no que a padrões diz respeito, as consequências de um padrão incluem o seu impacto na generalidade do sistema. Estas consequências podem ser de uma importante ajuda quando se trata da avaliação do padrão.

Apesar de existir vários tipos de padrões, nesta dissertação serão focados dois tipos: padrões de *user interface* e padrões de teste de *user interface*. Ora, considerando que a solução apresentada passa pela expansão de um catálogo destes mesmo padrões, torna-se importante uma representação mais formal de um padrão. Deste modo, podemos representar um padrão como um conjunto de tuplos:

{<Objetivo, V, A, C, P>}, em que:

- O **Objetivo** é o ID do padrão

- **V** é um conjunto de pares constituídos por uma variável e um valor, que relaciona os dados fornecidos com as variáveis envolvidas

- **A** é a sequência de ações a executar

- **C** é o conjunto de verificações a serem efetuadas

- **P** é a pré-condição que define as condições em que o padrão é aplicado.

Tendo em conta os dois tipos de padrão acima referidos e a definição do tuplo de padrão, é possível concluir que: para os padrões de interface gráfica, **P** define quando efetuar a verificação da existência do padrão, **A** define as ações a serem executadas de modo a verificar a presença do padrão e **C** valida a presença do padrão . No que diz respeito aos padrões de teste: **P** tem como fim definir quando um teste é aplicado, **A** é a ação executada pelo teste e **C** indica o sucesso ou insucesso do teste.

[MP15a, MP15b]

2.5.2 PBGT - projeto de teste de interface gráfica baseado em padrões

Como referido anteriormente, o projeto de teste de interface gráfica baseado em padrões ([MP14c, MPNM17, MP14a, CPFA10, **?**]) almeja diminuir o esforço necessário para a construção de modelos para MBT.

Considerando que padrões de interface gráfica demonstram comportamento comum, que pode ser implementado de maneiras ligeiramente diferente, o objetivo deste projeto passa por criar estratégias de teste com diferentes configurações possíveis a fim de permitir o teste a diferentes implementações do mesmo padrão de interface gráfica.

O PBGT tem como objetivo permitir a construção de modelos cujo foco é modelar o comportamento a testar (perspetiva de quem testa) e não o comportamento de uma GUI (perspetiva de utilizador/*developer*). De forma a alcançar estes objetivos, foi desenvolvida uma *DSL (Domain-Specific Language* denominada **PARADIGM** [MP14b].

O objetivo desta linguagem passa por:

- colecionar abstrações de domínio aplicáveis, ou seja, padrões de teste
- permitir relações específicas entre estas abstrações
- fornecer uma maneira de estruturar os modelos em diferentes níveis de abstração, de modo a lidar com a complexidade do processo

O modelo da linguagem encontra-se representada na figura 2.3

De maneira a promover a reutilização ao construir modelos de interface gráfica para teste, a linguagem PARADIGM pode ser estendida. Isto permite a definição de padrões de teste adicionais (elementos de comportamento) no topo de padrões-base já existente (por exemplo, *Input, Sort, Find*. Os novos padrões são denominados de padrões de alta-ordem (**HOP**). O principal benefício desta habilidade de extensão de linguagem, é o facto de cada *tester* poder possuir o seu próprio conjunto de padrões de teste de UI que formam uma espécie de biblioteca capaz de evoluir conforme o desejado, adaptando-se a características especificas da GUI a ser testada [MPM13].

Foi, também, realizado uma experiência ([CPN14]) em aplicações móveis, no sentido de averiguar se esta abordagem poderia ser aplicada de igual modo a estas aplicações. Com o sucesso desta experiência, ficou provada a necessidade de desenvolver estratégias de teste apropriadas para aplicações móveis.

Este projeto é baseado na assunção de que interface gráficas baseadas nos mesmos padrões de interface gráfica devem partilhar as mesmas estratégias de testes de UI.

Este projeta utiliza, também, um processo de engenharia reversa de modo a gerar e extrair automaticamente o modelo de uma aplicação web.

Apesar de este projeto se focar, inicialmente, em plataformas web, verificou-se que a mesma abordagem, ainda que com algumas alterações devido à arquitetura e conceitos próprios do Android (atividades, interação do utilizador) [MP15a], a *iMPAcT tool*, que será descrita no capítulo 3.

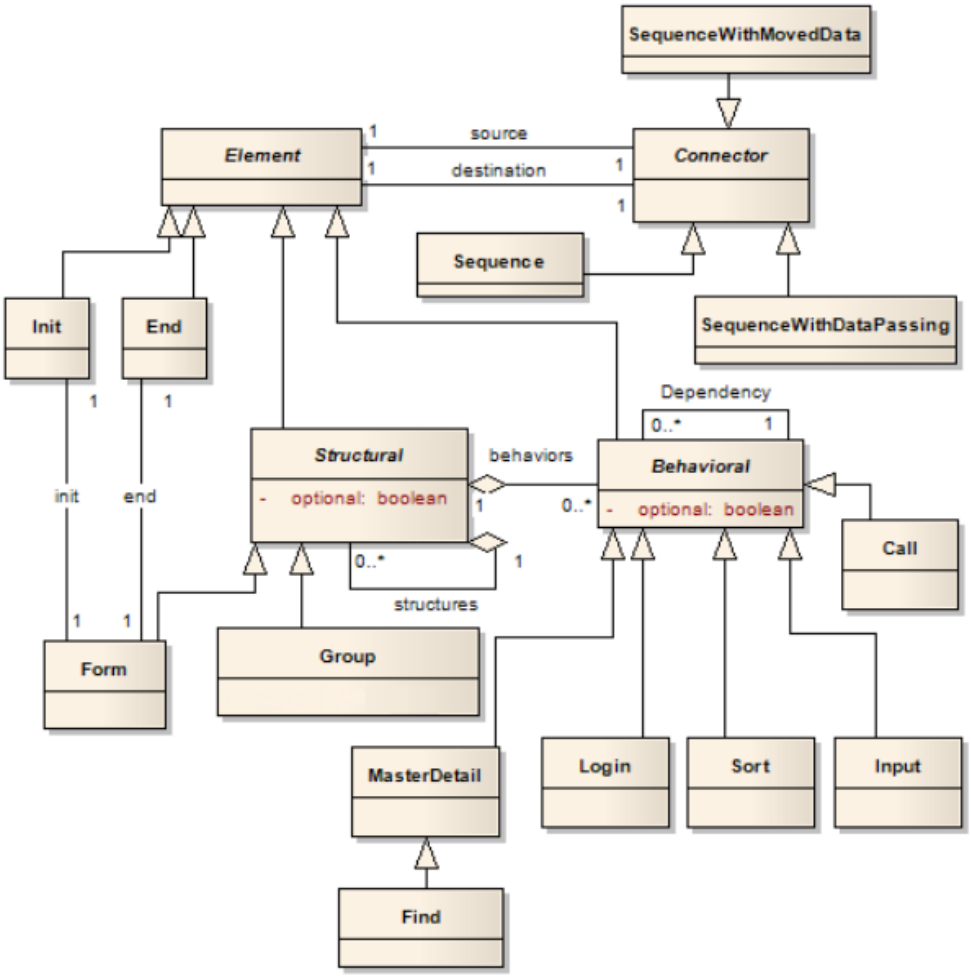

Figura 2.3: *Meta-model* da linguagem PARADIGM

2.6 Conclusões

Após esta revisão literária ao estado da arte podemos concluir que esta é uma área com margem para evoluir, como mostram os vários estudo realizados, tal como a opinião de profissionais com experiência na matéria. Efetivamente, o mercado dos *smartphones* tem apresentado um crescimento quase exponencial de aplicações disponíveis no mercado.

Com este acentuado crescimento foi surgindo a necessidade de garantir a qualidade das aplicações e para isso é necessário testar estas de uma forma que seja eficaz em termos de recursos e tempo. Apesar de existir essa necessidade e terem sido apresentadas várias soluções como resultado, o seu desenvolvimento parece estar, neste momento, aquém das necessidades mostradas. Para além disso, as ferramentas mais utilizadas que são revistas neste capítulo têm como defeito o facto de não serem capazes de disponibilizar um serviço de automação de testes em que a geração destes também seja feita de forma automática.

Com o intuito de resolver o problema, foi criada a ferramenta *iMPAcT tool*, focada em testar as

melhores práticas no desenvolvimento de aplicações móveis. Esta é uma ferramenta que visa ajudar ajudar todos os envolvidos no desenvolvimento de aplicações, apresentando uma combinação entre engenharia reversa e padrões de modo a fornecer um serviço de teste às boas práticas de desenvolvimento Android, de uma maneira completamente automática. Esta ferramenta é analisada em maior detalhe no capítulo 3.

Capítulo 3

iMPAcT Tool

Com o intuito de ajudar quem desenvolve aplicações móveis a cumprir com as melhores práticas destinadas à plataforma Android, é apresentada aqui como uma solução a *iMPAcT tool*. Tem como principal objetivo automatizar o processo de testes à interface gráfica, identificando e testando padrões recorrentes presentes em aplicações.

3.1 Abordagem

A abordagem desta ferramenta consiste em, como anteriormente referido, testar comportamentos recorrentes no que diz respeito à interface gráfica de aplicações Android. Inicialmente, é feita a exploração da aplicação sob teste, identificando a presença de padrões de interface gráfica. Após a confirmação destes padrões, é-lhes aplicado o padrão de teste correspondente.

Com isto, a abordagem apresenta dois tipos de resultados diferentes:

- Padrões identificados - padrões de interface gráfica presentes na aplicação
- Falhas - os padrões incorretamente implementados na aplicação.

[MP15b]

3.1.1 Implementação

A implementação do processo iterativo desta ferramenta é dividida em três fases, como demonstrado na figura 3.1:

- **Exploração**
- **Identificação de padrões**
- **Teste**

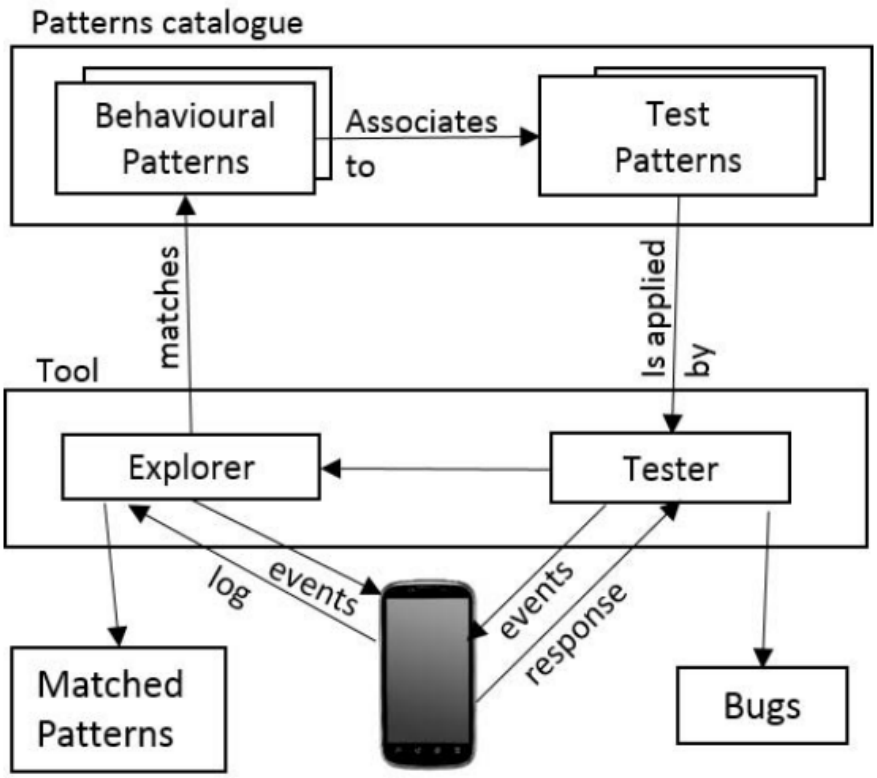

Figura 3.1: Processo da *iMPAcT Tool* [MPF14]

Na fase inicial, a exploração começa por analisar o estado atual da aplicação, fazendo uma representação hierárquica dos elementos presentes no ecrã do dispositivo. Esta representação é feita na forma de árvore, sendo que cada nó diz respeito a um elemento, como por exemplo um botão. Após construir esta representação, cada evento passível de ser acionado em cada elemento é identificado e escolhido de forma aleatório para ser simulado.

Na próxima fase, após o evento da fase anterior ser identificado, a ferramenta irá tentar identificar quais os padrões de interface gráfica presentes no momento. Para fazer esta identificação, são analisados todos os padrões presentes no catálogo da ferramenta.

A fase de teste é iniciada quando é encontrado o padrão de interface gráfica, ou padrões, da fase anterior. Nesta fase, são aplicados os padrões de testes, presentes no catálogo, associados aos padrões de interface gráfica identificados. O processo desta fase consiste, à semelhança da fase anterior, em verificar se as pré-condições dos testes se verificam, executando as ações necessárias, neste caso o próprio teste, e, no final, analisar se as verificações finais são cumpridas [MP15b, MP15c].

3.1.2 Configuração de testes

A ferramenta *iMPAcT tool* está dividida em duas partes principais [MP17], desenvolvidas em *Java*.

- **Configurador**, que é responsável pela configuração do processo de testes, recebendo informação dada pelo utilizador sobre a aplicação a ser testada.

- ***Tester***, que consiste na implementação da abordagem falada na sub-secção anterior.

O primeiro passo para usar a aplicação é providenciar a informação necessária para a realização do teste. Estas informações podem ser vistas na figura x.

Nesta janela, o utilizador poderá:

- indicar a aplicação a ser testada, usando o caminho para o APK ou o nome do *package*.

- escolher o tipo de exploração

- onde correr o teste: num dispositivo ou num emulador

- definir os padrões a ser testados, escolhendo-os a partir dos disponibilizados no catálogo da ferrramenta.

3.1.3 Ciclo de deteção de padrões

Tendo o utilizador fornecido a aplicação a ser testada e quais os padrões a serem identificados e testados, resta, ainda, a escolha de como o ciclo de eventos é gerido. Para isso, o utilizador terá de escolher um dos seguintes modos:

- ***Execute Once***: Apenas aciona os eventos que ainda não tenham sido ainda realizados, Ou seja, só aciona cada evento disponível uma única vez.

- ***Prioritize Not Executed***: À semelhança do anterior, este modo é focado em acionar eventos que ainda não o tenham sido. No entanto, neste modo, é possível existir repetição de eventos, mas apenas caso todos os outros eventos associados a um ecrã tenham sido lançados e, para aceder ao próximo ecrã esse evento seja necessário.

- ***Prioritize List Items and Not Executed***: Este modo apresenta duas grande diferenças no que toca ao previamente descrito: a primeira é o facto de eventos associados a elementos presentes em listas terem prioridade sobre os restantes; o evento de "pressionar no botão da aplicação"só é acionado quando não for detetado mais nenhum evento a ser realizado. Assim, este modo tem como objetivo percorrer todos os eventos de um ecrã antes de prosseguir para o seguinte.

- ***All Events***: Neste modo, os eventos não são diferenciados, pelo que os eventos serão acionados aleatoriamente. Este evento pode levar a que a aplicação sobre teste sofra várias transições pelos diferentes ecrãs da app sem que os eventos respetivos a cada um tenham sido lançados.

3.1.4 Condições de paragem

A *iMPAcT Tool* irá fazer o ciclo de testes até uma de duas condições:

- Disparou todos os eventos: Quando deixam de existir eventos num ecrã, a ferramenta irá lançar o evento correspondente ao botão do *back* o que irá fazer com que a aplicação navegue para ecrãs anteriores. Se, durante esta navegação a aplicação sobre teste deixar de estar em *foreground* o ciclo é dado com terminado.
- O utilizador pressionou no botão *"Home"*: De modo a permitir que o utilizador tenha a decisão de quando acabar o ciclo, a ferramenta fornece a opção de terminar o ciclo de testes e exploração aquando do toque do utilizador no botão de sistema *"Home"*.

3.2 Catálogo de padrões

A ferramenta visada nesta dissertação possui um catálogo de padrões de interface gráfica descritos no capítulo 2. Este catálogo é instanciado apenas uma vez, quando a aplicação de teste é inicializado, e contém toda a informação, quer das condições para detetar cada padrão, quer como as regras que têm de obedecer para serem considerados corretamente implementados. Relembrando a definição de um padrão [MP]:

{<Objetivo, V, A, C, P>}, em que:

- O **Objetivo** é o ID do padrão
- **V** é um conjunto de pares constituídos por uma variável e um valor, que relaciona os dados fornecidos com as variáveis envolvidas
- **A** é a sequência de ações a executar
- **C** é o conjunto de verificações a serem efetuadas
- **P** é a pré-condição que define as condições em que o padrão é aplicado.

Os padrões descritos nesta secção são, maioritariamente baseados em especificações para o bom desenvolvimento de aplicações, fornecidas pela Google para Android.

3.2.1 Padrão *Side Drawer*

Este padrão testa uma das formas de navegação hierárquica possíveis em aplicações na plataforma Android. O *side drawer* é um menu que é aberto de duas maneiras: um "swipe"do utilizador na borda esquerda do ecrã, ou através do clique no botão da app (botão normalmente localizado na parte esquerda da *Action Bar* no topo da aplicação.

De acordo com as guias de desenvolvimento, um *side srawer* é considerado corretamente implementado quando: obedece às regras de como deve ser aberto ("swipe"ou botão da app) e se ocupar todo o espaço vertical destinado à aplicação.

Este padrão deteta se existe algum *side drawer* não visível e é definido como:

- **Objetivo**: "Existe um *Side Drawer*"
- **P**: {"verdadeiro"}
- **V**: {}
- **A**: [observação]
- **C**: {Existe uma *side srawer* e não está visível }

De seguida, o padrão de teste correspondente abre o *side drawer* e faz a verificação do espaço vertical ocupado pelo mesmo. Este padrão é definido como:

- **Objetivo**: "*Side Drawer* tem a altura total"
- **P**: {"Padrão de interface identificado, *side drawer* disponível e padrão de teste ainda não aplicado à atividade atual"}
- **V**: {}
- **A**: [abrir *side drawer*, observação]
- **C**: {cobre o ecrã na altura total }

3.2.2 Padrão de Orientação

Quando existe rotação do dispositivo, o ecrã da aplicação acompanha o movimento e atualiza a sua disposição. Com isto, ao desenvolver aplicações é preciso ter o cuidado de verificar que o comportamento das guias de Android são seguidas:

- nenhum do *input* do utilizador pode ser perdido, ou seja, todo o conteúdo introduzido, por exemplo em *textedit*, deverá estar presente após a rotação.
- os *widgets* não devem desaparecer do ecrã

O padrão de interface gráfica verifica se é possível realizar a rotação do ecrã e é definido da seguinte maneira:

- **Objetivo**: "Rotação é possível"
- **P**: {"verdadeiro"}
- **V**: {}
- **A**: [observação]
- **C**: {é possível rodar o ecrã}

Para este padrão existem dois padrões de teste correspondentes. O primeiro, é aplicado quando o padrão de interface é detetado e o teste ainda não foi aplicado previamente na atividade em questão. A sua ação passa por rodar o ecrã e verificar que não houve perda de *widgets*. Se se verificar que estes não estão inicialmente presentes, é efetuado um *scroll* do ecrã para verificar os elementos restantes e fazer uma verificação final. Este padrão tem como definição:

- **Objetivo**: "Principais componentes da interface gráfica continuam presentes"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado à atividade atual"}
- **V**: {}
- **A**: [observação, rotação do ecrã, observação, *scroll* do ecrã, observação]
- **C**: {*widgets* ainda presentes}

O segundo padrão de teste é aplicado quando existe dados inseridos pelo utilizador e o padrão ainda não foi executado a um elemento que tenha sofrido alteração. O ecrã é rodado e é verificado se os dados que foram inseridos se mantêm. A sua definição é:

- **Objetivo**: "Dados mantêm-se aquando da rotação"
- **P**: {"Padrão de interface identificado, foi inserido algum tipo de dado, e padrão de teste ainda não aplicado ao elemento modificado"}
- **V**: {}
- **A**: [observação, rotação do ecrã, observação, *scroll* do ecrã, observação]
- **C**: {a totalidade dos dados inseridos é mantida}

3.2.3 Padrão de Abas (*Tabs*)

Em muitas aplicações Android são usadas abas para organização de conteúdo e facilidade de exploração para o utilizador. Existem algumas guias no que toca ao uso e comportamento de abas [BN]:

- Deve existir apenas um conjunto de abas, de modo a que seja claro qual o conteúdo a ser navegado.
- Em aplicações Android as abas devem ser mostradas na parte superior do ecrã, ao contrário de iOS (mostradas em baixo).
- deslizar o ecrã horizontalmente deve apenas mudar de aba, ou seja, não deve existir nenhum elemento que permita a mesma ação

O padrão de interface gráfica correspondente verifica se existe alguma aba e é definido como:

- **Objetivo**: "Presença de abas"
- **P**: {"verdadeiro"}
- **V**: {}
- **A**: [observação]
- **C**: {"Existem abas presentes"}

Existem três padrões de teste correspondentes ao teste de abas. O primeiro verifica se existe apenas um conjunto de abas presentes e a sua definição é:

- **Objetivo**: "Apenas um conjunto de abas"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação]
- **C**: {"Existe apenas um conjunto de abas a ser mostrado"}

O segundo padrão verifica se as abas estão posicionadas corretamente na parte superior do ecrã e é definido como:

- **Objetivo**: "Posição das abas"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação]
- **C**: {"As abas encontram-se na secção superior do ecrã"}

Por fim, o terceiro padrão de teste executa um *swipe* horizontal de modo a verificar se existe uma mudança da aba ativa. É definido como:

- **Objetivo**: "Fazer um *swipe* horizontal deve mudar a aba ativa"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, *swipe* horizontal do ecrã]
- **C**: {"A aba ativa é trocada"}

3.2.4 Padrão de *Background*

Em aplicações Android é comum existirem tarefas a serem executadas em paralelo a sua própria *framework* da plataforma oferece várias opções para realizar tarefas no plano de fundo do dispositivo. Para além disso, também é dada a opção ao utilizador de enviar uma aplicação para o *background*. Tal é possível pressionando o botão "Home"do dispositivo. Esta ação leva a que a aplicação deixe de estar a ser usada e é esperado que esta continue a executar no fundo e que a informação presente no momento da transição se mantenha.

Tendo em conta que para o teste deste padrão a única condição necessária é que a aplicação sob teste esteja aberta, não existe uma formalização do padrão de interface gráfica.

O padrão de teste correspondente é definido como:

- **Objetivo**: "A aplicação é enviada para o fundo e os dados mantêm-se intactos"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, lança a aplicação para *background*, observação, retornar a aplicação, observação]
- **C**: {"O estado da aplicação é igual antes e depois de estar em *background*"}

3.2.5 Padrão de *Action Bar*

De acordo com as guias de desenvolvimento para Android, o uso de uma barra de ação, ou barra da aplicação, promove a consistência das aplicações, fazendo com que estas se tornem mais similares, permitindo, assim, o utilizador tornar-se familiar com diferentes aplicação em menos tempo. As principais características de uma *action bar* são:

- É um espaço dedicado a atribuir uma identidade à aplicação e situar o utilizador nela.
- Fornece acesso a funções importantes ao utilizador, como, por exemplo, pesquisa ou definições.
- Suporte à navegação e a mudanças na *view* da aplicação.

Para além das características enumeradas, esta barra tem, também, certos elementos que devem estar presentes. No seu modo mais simples, a *action bar* é composta pelo nome da aplicação no seu lado esquerdo e por um menu flutuante à direita. Para além destes dois elementos, também deve ser incluída a seta da ação "Up"à esquerda do título quando a aplicação não se encontra no seu ecrã principal.

À semelhança do que acontece com o padrão de *background*, não existe uma formalização do padrão de teste, assumindo, portanto, que a *action bar* estará sempre presente numa aplicação. Portanto, o padrão de teste correspondente apresenta a sua definição como:

- **Objetivo**: "Existe uma *action bar* no ecrã da aplicação"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação]
- **C**: {"Existe uma barra no ecrã e esta tem os elementos necessários"}

3.2.6 Padrão do *Up*

Em Android, é visto como boa prática existir uma seta na *action bar* de cada ecrã, que não seja o principal de uma aplicação, cuja função é oferecer navegação para o ecrã que é o seu parente lógico na hierarquia da aplicação. Este padrão almeja verificar se, em cada ecrã diferente do ecrã principal existe a seta de navegação, com a exceção de haver um *side drawer*. Para além disso, testa, também, se a navegação aquando do clique do botão é realizada tendo em conta a hierarquia de ecrãs da aplicação, ou seja, é verificado se o ecrã que resultou do evento "Up"é o parente lógico do ecrã prévio.

O padrão de interface gráfica é definido como:

- **Objetivo**: "Existe o padrão *Up*"
- **P**: {"Aplicação não se encontra no ecrã principal"}
- **V**: {}
- **A**: []
- **C**: {"Tem *action bar* e não existe *side drawer*"}

O padrão de testes correspondente é:

- **Objetivo**: "Existe uma seta *Up* e navega para o ecrã hierarquicamente superior"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, clique na seta *Up*, observação]
- **C**: {"O estado da aplicação é igual antes e depois de estar em *background*"}

3.2.7 Padrão do *Back*

Em todos os dispositivos Android existe um botão que permite um tipo de navegação diferente do visto no padrão do *Up*. Esta navegação permite ao utilizador percorrer os ecrãs previamente visitados (a *back stack* do sistema Android), independentemente do estado em que se encontrem. Normalmente, este botão *back* é tratado pelo próprio sistema, no entanto, é possível alterar este comportamento e especificar manualmente qual o comportamento do *back* será no contexto de uma dada aplicação. Este padrão verifica se:

- A aplicação sob teste usa o comportamento definido pelo sistema e não um personalizado
- a aplicação sob teste não está no ecrã principal, pois o evento *back* quando usado no ecrã principal retorna o utilizador para o ecrã principal do **dispositivo** e ativaria a condição de paragem da ferramenta
- O clique no botão *back* provoca a navegação para o ecrã anterior da aplicação

Como este botão é uma parte integrada no próprio sistema, é seguro assumir que estará sempre presente, logo a formalização de um padrão de interface gráfica para ser identificado perde o seu propósito. Assim, o padrão de teste correspondente é definido como:

- **Objetivo**: "Botão **Back** navega para o ecrã anterior"
- **P**: {"Não é o primeiro ecrã visitado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, clique no botão *back*, observação]
- **C**: {"O ecrã foi alterado para o imediatamente anterior"}

3.3 Artefactos

Quando a *iMPAcT Tool* acaba o seu ciclo de exploração são produzidos dois artefactos que possibilitam o utilizador a ganhar uma maior compreensão quer da aplicação sob teste, quer das ações e resultados dos testes.

3.3.1 Relatório

O relatório final da ferramenta está dividida em duas partes:

- Registo da exploração
- Resultado dos testes

3.3.1.1 Registo da exploração

O registo da exploração está, por sua vez, também dividido em duas partes: 1) identificação da aplicação sob teste e dos parâmetros da exploração e 2) sequência de eventos lançados durante a exploração

Identificação da aplicação sob teste e dos parâmetros da exploração

A aplicação sob teste é identificada pelo nome do seu pacote e os parâmetros da exploração são:

- a versão do sistema Android do dispositivo/emulador em que está a ser executada a aplicação sob teste
- altura e comprimento do ecrã
- a coordenada *y* em que a aplicação começa a ser desenhada: a não ser que seja iniciada em *fullscreen*, uma aplicação nunca começará a ser desenhada pelo topo do ecrã.

Sequência de eventos lançados durante a exploração

Cada um dos eventos realizados pela ferramenta é registado como um tuplo <*Evento, Elemento, Execuções*>.

O *Evento* identifica o evento que é lançado e é, por seu lado, também um tuplo entre o tipo de evento que se trata (clique, editar) e o valor de entrada usado nesse evento. O valor de entrada é opcional dependendo do evento que é lançado.

O *Elemento* é utilizado apenas quando o evento é lançado num elemento do ecrã da aplicação, como por exemplo, um clique num botão. É formado por um tuplo <*class, id do recurso, texto, descrição do contéudo, margens*>, em que:

- *class*: identifica o tipo de elemento (botão, text, etc...)
- *id do recurso*: identifica o elemento. Esta propriedade não é considerada nem única nem mandatória, estando definido assim pela própria plataforma Android.
- *texto*: texto a ser mostrado no ecrã, dentro do limites do elemento. Não é uma propriedade mandatória e está, normalmente, associada a botões, ou caixas de texto. Pode ser uma *string* vazia.
- *descrição do conteúdo*: texto que é associado a um elemento com o objetivo de descrever o resultado de um clique nesse elemento, mas que, no entanto, não é representado no ecrã. Esta propriedade não é mandatória.
- *margens*: indica a posição do elemento dentro do ecrã como um retângulo e é mandatória.

Execuções mantém o número de vezes que um certo evento foi lançado num dado elemento.

3.3.1.2 Resultado dos testes

O relatório contém os seguintes resultados:

- número de eventos realizados e eventos identificados, tal como a percentagem correspondente
- se cada um dos padrões a ser testado foi detetado e, caso tenha sido, indica se o teste conclui que o padrão está corretamente implementado ou não
- a duração da exploração

Este relatório é útil para o utilizador da ferramenta poder consultar todos os passos feitos pelo ciclo de exploração, tal como se e quando é detetado um padrão e o resultado do seu padrão de teste correspondente.

3.3.2 Modelo do comportamento observado

O processo de engenharia reversa é utilizado na ferramenta para obter um modelo da interface gráfica da aplicação. Este modelo segue a forma de uma máquina de estados finita e representa o comportamento da aplicação em que os distintos estados do grafo correspondem a diferentes atividades da aplicação sob teste e as suas transições representam os eventos que provocam a transição de um ecrã para outro.

Neste modelo podem ser representados três tipos de estados:

- uma atividade da aplicação sob teste. O estado encarregue da representação desta atividade é identificado como *Ecrã<i>* em que *i* corresponde ao id da atividade. Este id é atribuído consoante as atividades são alcançadas pelo ciclo, ou seja o *Ecrã 0* será o ecrã principal da atividade.
- uma falha da aplicação (*crash*). Quando existe uma falha da aplicação e esta termina inesperadamente tal é registado e é adicionado um estado representativo à maquina de estados. Apenas pode existir um estado com este tipo, todos os eventos que despoletarem uma próxima falha serão ligados a este estado.
- a exploração abandonou a aplicação. Quando um evento leva a que a exploração abandone a aplicação é adicionado um estado que o represente. À semelhança com o estado anterior, apenas pode existir um estado deste tipo.

3.4 Conclusões

Após tendo analisado a ferramenta em questão nesta dissertação, podemos concluir que se trata de uma ferramenta inovadora, misturando engenharia reversa com padrões de interface gráfica com uma estrutura que incentiva à sua extensão. Para além disso, é uma ferramenta de fácil usabilidade, visto que não é necessário o utilizador possuir qualquer tipo de informação ou conhecimento em relação quer aos padrões implementados no catálogo, quer da aplicação a ser submetida aos testes.

Capítulo 4

Implementação

Neste capítulo será discutido o trabalho desenvolvido para esta dissertação. Na primeira secção irá incidir nas correções feitas a padrões anteriormente presentes na *iMPAcT Tool*. Na secção seguinte é descrito o novo padrão adicionado ao catálogo da ferramenta, bem como as técnicas e ferramentas utilizadas no seu desenvolvimento

4.1 Melhorias implementadas na *iMPAcT Tool*

Nesta secção serão revistas algumas das definições formais dos padrões previamente implementados no catálogo da ferramenta. Esta necessidade surgiu no âmbito de refinamento da *iMPAcT Tool*, tendo em conta guias mais específicas do bom desenvolvimento em Android. Essa revisão, permitiu desenvolver alguns melhoramentos que irão aumentar a cobertura dada pelos testes.

4.1.1 Padrão *Back*

4.1.1.1 Revisão

No que diz respeito ao padrão de teste de interface gráfica correspondente à ação de retrocesso no Android, este foi, inicialmente, definido como:

- **Objetivo**: "Botão *Back* navega para o ecrã anterior"
- **P**: {"Não é o primeiro ecrã visitado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, clique no botão *back*, observação]
- **C**: {"O ecrã foi alterado para o imediatamente anterior"}

De facto, tendo em conta as guias de desenvolvimento para Android, e, no que diz respeito à navegação, este é o comportamento correto. Porém, o comportamento do back não se limita à navegação, existindo algumas condicionantes que o afetam:

- um elemento do ecrã que esteja em destaque (*highlight*), é retirado desse estado quer por um clique noutra secção do ecrã, ou pressionando o butão *back*.
- quando existem janelas flutuantes no ecrã da aplicação, existe a opção de estas serem fechadas com um clique no botão *back*
- ao introduzir dados numa aplicação, como, por exemplo, texto, o teclado virtual é aberto e a maneira de o fechar passa por invocar o *back*.

4.1.1.2 Melhoria implementada

Terminada a análise feita a este padrão, é seguro assumir que existe espaço para melhoramentos, os quais foram desenvolvidos no âmbito desta dissertação. Para o melhoramento do "back"foi necessária uma reformulação formal do seu padrão de teste. Para facilidade de leitura a nova definição formal será dividida em quatro definições mais pequenas:

A primeira definição é igual à já implementada:

- **Objetivo**: "Botão *Back* navega para o ecrã anterior"
- **P**: {"Não é o primeiro ecrã visitado e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, clique no botão *back*, observação]
- **C**: {"O ecrã foi alterado para o imediatamente anterior"}

Na segunda, é verificado se um elemento que esteja em destaque no ecrã anterior a ser invocado o evento deixa de o estar aquando do clique no botão *back*:

- **Objetivo**: "Botão *Back* retira o destaque ao elemento em questão"
- **P**: {"padrão de teste ainda não aplicado ao elemento"}
- **V**: {}
- **A**: [observação, clique no botão *back*, observação]
- **C**: {"O elemento perdeu o destaque"}

É de referir que, neste caso, o ecrã da aplicação se mantém, essencialmente, o mesmo, pelo que, também é verificado se os elementos originais permanecem no ecrã resultante.

A terceira parte desta definição diz respeito à verificação do comportamento do *back* em relação a janelas flutuantes:

- **Objetivo**: "Botão *Back* fecha a janela flutuante"
- **P**: {"existe uma janela flutuante e padrão de teste ainda não aplicado ao elemento"}

- **V**: {}
- **A**: [observação, clique no botão *back*, observação]
- **C**: {"A janela flutuante é fechada"}

A última verificação deste padrão passa por testar se, quando existe um teclado virtual presente no ecrã, este é fechado quando o botão *back* é pressionado:

- **Objetivo**: "Botão *Back* fecha o teclado virtual"
- **P**: {"existe um teclado virtual e padrão de teste ainda não aplicado na atividade"}
- **V**: {}
- **A**: [observação, clique no botão *back*, observação]
- **C**: {"O teclado virtual é fechado"}

É importante frisar que, na parte prática, estas definições juntam-se num só padrão de testes para ser usado pela ferramenta.

4.1.2 Padrão de Orientação

4.1.2.1 Revisão

No padrão de teste original referente à orientação, a sua definição formal passava por:

- **Objetivo**: "Principais componentes da interface gráfica continuam presentes"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado à atividade atual"}
- **V**: {}
- **A**: [observação, rotação do ecrã, observação, *scroll* do ecrã, observação]
- **C**: {*widgets* ainda presentes}

Apesar do padrão estar, considerando as guias de desenvolvimento, correto, não se encontra completo. Para este padrão se encontrar completo foi proposto o melhoramento a seguir descrito.

4.1.2.2 Melhoria implementada

Como tal, foi feita uma nova definição formal deste padrão:

- **Objetivo**: "Principais componentes da interface gráfica continuam presentes"
- **P**: {"Padrão de interface identificado e padrão de teste ainda não aplicado à atividade atual"}

- **V**: {}
- **A**: [observação, rotação do ecrã, observação, *scroll* do ecrã, observação, rotação do ecrã, observação, *scroll* do ecrã, observação]
- **C**: {Componentes da interface gráfica do ecrã original ainda presentes}

Atendendo, agora, à nova definição, é possível verificar que é adicionado mais um ciclo de rotação ao ecrã e consequente observação. Com isto, após a primeira rotação e posteriores verificações, o ecrã é rodado mais uma vez, para a sua posição original. Assim, é feita a verificação integral da estrutura dos elementos da aplicação sob teste, verificando se quer a rotação de modo de retrato para modo de paisagem, quer o contrário, mantém os elementos do ecrã original. Foi, de igual modo, acrescentado mais um evento de *scroll* de modo a obter todos os elementos que não estejam inicialmente no ecrã após a segunda rotação.

4.2 Padrão de chamadas

Nos dias que correm, o mercado de aplicações para a plataforma Android tem apresentado um grande crescimento, ou seja, existem cada vez mais aplicações disponibilizadas ao utilizador. Estas aplicações podem apresentar todo o tipo de diferentes funcionalidades, desde simples organizadores pessoais até aplicações complexas que envolvam transferências bancárias. No entanto, é importante não esquecer do que um dispositivo Android é na sua génese: um telemóvel. Sendo um telemóvel, é percetível que este irá estar encarregado de receber chamadas/mensagens e, tendo em conta que no dispositivo serão utilizadas variadas aplicações, estas têm de garantir que a sua estrutura de dados e elementos se mantém e que esta não falha aquando de uma chamada telefónica. Para testar esse comportamento, foi desenvolvido o padrão de chamadas.

4.2.1 Estrutura do padrão

Tendo em conta a estrutura já definida da ferramenta e algumas limitações de segurança do Android, foi necessário criar uma estrutura ligeiramente diferente das já vistas. Assim este padrão é composto por:

- **Atividade de chamadas**: encarregue de injetar chamadas no dispositivo que realiza os testes
- **Detetor de chamadas**: encarregue de detetar chamadas recebidas

4.2.1.1 Atividade de chamadas

Como o nome indica, este componente trata-se de uma atividade Android, passível de ser utilizada por grande parte das versões da plataforma. Esta atividade é inicializada num dispositivo (emulador) diferente daquele em que os testes estão a ser executados e é o único processo relativo à ferramenta que está presente nesse dispositivo. A atividade é composta por um ciclo que é responsável por efetuar uma chamada a cada 10 segundos para o dispositivo encarregue pelo teste. O intervalo de tempo foi definido de modo a que o dispositivo a executar a *iMPAcT Tool* não seja inundado por chamadas e facilita a compreensão do utilizador no que diz respeito às ações do padrão.

4.2.1.2 Gestor de chamadas

Este componente toma partido da capacidade que o Android possui de executar várias tarefas em paralelo para, ao contrário do componente anterior, poder ser executado no mesmo dispostivo que se encontra a realizar os testes. Isto permite que a iMPAcT tool seja capaz de obter informação sobre o estado de chamada do telémovel. Assim, é possível detetar chamadas recebidas no dispositivo desejado com o intuito de terminar as mesmas e prosseguir com a fase de teste ao padrão.

4.2.2 Comportamento do padrão

Estando a estrutura definida, é preciso analisar mais aprofundadamente os diferentes componentes, tal como as suas interações com os componentes da ferramenta previamente desenvolvidos. Devido ao facto de não ser possível efetuar chamadas entre dois dispositivos reais sem ter um prestador de serviços móveis, este padrão requer que sejam inicializados dois emuladores: o primeiro para ser o dispositivo encarregue por executar a ferramenta e receber chamadas; e o segundo em que estará a ser executada a atividade de chamadas. Para não sobrecarregar a interface gráfica da ferramenta e tornar o processo o mais autónomo, a *iMPAcT Tool* assume que o primeiro emulador a ser instanciado é o principal, ou seja, o que corre os testes, e o segundo como o auxiliar.

Após a inicialização dos emuladores, ao escolher este padrão para ser testado, são postos a correr os componentes respetivos e a ferramenta irá executar os testes de outros padrões eventualmente escolhidos enquanto escuta por chamadas recebidas. Quando é detetada a chamada é usada uma técnica denominada *Java Reflection* para recusar a chamada.

Este comportamento é resumido na figura 4.1.

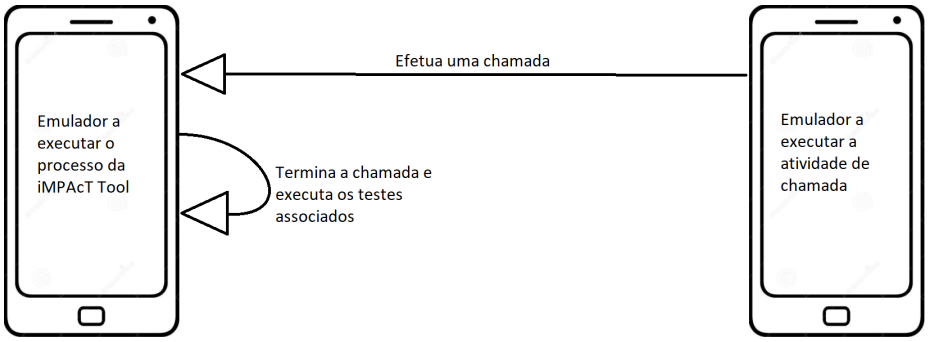

Figura 4.1: Processo do comportamento do padrão de chamadas

4.2.2.1 Java Reflection

Reflection é a habilidade de correr um programa de modo a fazer uma análise a si próprio e ao seu ambiente , para além de mudar o seu comportamento mediante essas análises. De modo a fazer a auto-análise, o programa necessita de possuir uma representação de si mesmo (*metadata*). No caso de ambiente de programação orientada a objetos (como Java), a *metadata* é organizada em objetos denominados ***metaobjects***. No geral, existem três técnicas que uma API de *reflection* pode oferecer de modo a mudar o comportamento do programa [FFI+04]:

- modificação direta de *metaobjects*
- operações usando *metadata*, como invocação dinâmica de funções
- intersecção, em que o código pode interceder em várias fases da execução do programa

No âmbito desta dissertação é utilizada uma API de *reflection* de modo a ser possível aceder a métodos da API interna *Telephony* do Android. Tal é feito, definindo a representação da API como um ficheiro **AIDL** (*Android Interface Definition Language*) que, em termos práticos, é uma interface em Java definindo quais os métodos de *Telephony* a serem usados. Tendo a representação da API é possível, agora, invocar os métodos necessários para terminar a chamada recebida programaticamente a partir da *iMPAcT Tool*.

4.2.2.2 Definição

Após a chamada de funções da API interna de Android, será aplicado o padrão de testes referente às chamadas. Este irá verificar se: 1) a chamada foi, efetivamente, desligada, 2) se a aplicação não falhou com a interação do sistema e 3) caso não falhe, compara o ecrã anterior à chamada com o ecrã após a rejeição da mesma. Para este padrão é proposta a seguinte definição:

- **Objetivo**: "Aplicação não falha e mantém a coerência dos dados após receber uma chamada"
- **P**: {"Chamada recebida"}
- **V**: {}
- **A**: [observação, rejeição da chamada, observação]
- **C**: {"aplicação não apresenta falha e os dados e elementos mantêm-se"}

4.3 Resumo ou Conclusões

Revendo o trabalho desenvolvido nesta dissertação, foi possível estender a ferramenta com um novo padrão, tal como melhorar dois dos padrões previamente implementados. Acreditamos que este novo padrão traz um bom contributo para aumentar os comportamentos capazes de ser testados pela *iMPAcT Tool*. Isto é reforçado pelo facto de este padrão estender não só, como previamente referido, os comportamentos, mas as próprias funcionalidades da ferramenta, nomeadamente, a capacidade de fazer/lidar com chamadas. Os melhoramentos efetuados no âmbito desta dissertação trazem, também, acrescido valor à ferramenta, na medida em que consolidam os casos de testes dos padrões prévios, o que leva a que a cobertura de testes aumente consideravelmente.

Capítulo 5

Validação

5.1 Questões de pesquisa

Nesta secção, são apresentadas questões originadas ao longo do desenvolvimento desta dissertação. Estas questões irão servir de linha orientadora às conclusões a serem retiradas após a obtenção de resultados. O seu objetivo passa, por também, clarificar se os resultados obtidos são satisfatórios e se o trabalho desenvolvido contribuí, de facto, para o melhoramento da *iMPAcT Tool*. As respostas a estas perguntas de pesquisa serão dadas no capítulo 6. As questões são as seguintes:

- **QP1**: A *iMPAcT Tool* é capaz de identificar falhas em aplicações Android, tendo em conta o novo padrão acrescentado à ferramenta?

- **QP2**: Os padrões melhorados aumentaram a capacidade de identificar falhas nas aplicações sob teste?

- **QP3**: As falhas detetadas pela ferramenta são, realmente, falhas da aplicação?

É de notar que, apesar do principal objetivo da *iMPAcT Tool* é detetar falhas em aplicações, a não deteção das mesmas não implica que as aplicações sob teste não as demonstrem.

5.2 Especificação técnica

Para as experiências conduzidas no âmbito desta dissertação, a ferramenta foi utilizada em vários dispositivos. Para esse facto, contribui a necessidade do padrão desenvolvido em ser executado num dispositivo emulado. Os dispositivos apresentam as seguintes especificações técnicas:

Samsung Galaxy S8:

- Sistema operativo: Android Oreo (8.0.0)
- CPU: Octa-core (4x2.3 GHz Mongoose M2 and 4x1.7 GHz Cortex-A53)
- Chipset: Exynos 8895 Octa
- GPU: Mali-G71 MP20
- RAM: 4 GB

Asus ROG Strix GL553VD:

- Sistema operativo: Windows 10 x64
- CPU: Intel® Core™ i7 7700HQ
- Chipset: Intel® HM175 Express Chipset
- GPU: NVIDIA GeForce GTX 1050 , com 4GB GDDR5 VRAM
- RAM: 16 GB
- Emulador Android: Nexus 5X API 26
- Versão do emulador: Android Oreo (8.0.0)

Para o dispositivo emulado é dada a especificação do hardware em que este se assenta, ou seja, as especificações do computador.

5.3 Metodologia de teste

Nesta secção é descrita a metodologia de testes, de modo a avaliar os melhores dados possíveis. Para alcançar tal feito, é imperativo definir um plano de teste juntamente a uma seleção de aplicações a serem testadas.

Assim, o plano de testes referido apresenta os seguintes passos:

- Selecionar as aplicações a serem submetidas aos testes da *iMPAcT Tool*.

- Realizar uma inspeção manual das aplicações de modo a determinar a existência de falhas.
- Executar a ferramenta para cada uma destas aplicações.
- Recolher a informação dos resultados dos testes, ou seja, registar as falhas detetadas.
- Comparar os resultados obtidos na ferramenta com os resultados da inspeção manual.
- Retirar conclusões acerca dos resultados obtidos com este plano.

Iniciando o plano de teste, o primeiro passo consistiu em fazer uma seleção que seja consistente e abrangente. Para isso, foram selecionadas aplicações de várias categorias diferentes disponíveis na *Play Store*. A variedade de categorias confere uma panóplia de diferentes funcionalidades e aspetos a serem testados. Para além das categorias associadas a cada aplicação foram considerados mais alguns requerimentos a cumprir pelas aplicações a serem escolhidas:

- Têm de ser capazes de executar nos dispositivos especificados
- São grátis na *Play Store*
- Serem aplicações nativas de Android
- Não dependentes de outras aplicações
- Não precisar de autenticação para interagir
- Não necessitar de autorizações para aceder à câmara, contactos ou documentos do dispositivo
- Ter mais de cinco mil transferências na *Play Store*
- Passíveis de serem lidas pelo *UiAutomator*
- Possuir uma das seguintes características:
  - Funciona em *background*
  - Responde ao botão *Back* do dispositivo

De forma aos resultados obtidos serem o mais fiáveis possível, é necessário ter em conta que os eventos acionados durante o ciclo de exploração são aleatórios e, consequentemente, a ordem pela qual os eventos são executados pode alterar o resultado do teste. Assim, os resultados das experiências conduzidas nesta dissertação foram obtidos após correr, para cada aplicação, os testes da *iMPAcT Tool* três vezes e registando o resultado que se verifica mais vezes. Desta forma o grau de aleatoriedade do ciclo da ferramenta consegue ser reduzido.

É importante realçar que, de modo a obter a maior cobertura de testes possível para cada aplicação, ou seja, o maior número de eventos lançados, o algoritmo de exploração escolhido foi **Prioridade aos não executados**, devido à sua capacidade de repetir alguns eventos que possam já ter sido executados.

5.4 Deteção de falhas

O objetivo desta secção passa pela realização de uma experiência que visa responder à primeira das questões de pesquisa (QP1 - A *iMPAcT Tool* é capaz de identificar falhas em aplicações Android, tendo em conta o novo padrão acrescentado à ferramenta?). Para tal, foi, para cada aplicação da lista das escolhidas, aplicado o novo padrão desenvolvido.

No final desta experiência, deverá ser possível determinar se a ferramenta é capaz de detetar falhas quando estas estão presentes.

Tendo em conta o tamanho da lista de aplicações a serem testadas, os resultados completos referentes a esta experiência foram colocados em anexo. No entanto, após a sua analise podemos concluir que:

Para 61 aplicações:

- em 50 aplicações não foi detetado erro no padrão
- em 11 aplicações foi detetado erro no padrão

As aplicações que não cumpriram os requisitos do teste do padrão são apresentados na tabela 5.1. Após uma análise a esta tabela, é possível verificar que grande parte das falhas no padrão se encontram em aplicações que utilizem GPS ou edição de imagens/ficheiros. Tal facto pode ser explicado pelo facto de estas aplicações não possuírem mecanismos implementados que lidem com a repentina interrupção do seu trabalho. Ou seja, estas aplicações ao receberem a interação de chamada telefónica não armazenam o seu estado ou, simplesmente, falham.

Finalizados os primeiros resultados, a secção seguinte descreve a próxima experiência, com o objetivo de responder à segunda questão de pesquisa.

Tabela 5.1: Aplicações que falharam o teste do padrão de chamadas

Name	Calls
uCrop	Incorrect
Loop - Habit Tracker	Incorrect
OneBusAway	Incorrect
aMetro	Incorrect
MIFARE Classic Tool	Incorrect
GPS Logger for Android	Incorrect
Vlille Checker	Incorrect
CycleStreet	Incorrect
OpenBikeSharing	Incorrect
Amaze File Manager	Incorrect
Omni Notes	Incorrect

5.5 Melhorias Implementadas

Nesta secção será realizada mais uma experiência, esta com o intuito de responder à segunda questão de pesquisa (QP2 - Os padrões melhorados aumentaram a capacidade de identificar falhas nas aplicações sob teste?). Para esta experiência foram usadas as mesmas 61 aplicações de modo a obter os resultados mais fidedignos possíveis.

Tratando-se, esta secção, de uma análise comparativa a padrões realizados anteriormente é necessário saber qual a taxa de deteção no momento em que nenhuma alteração foi efetuada em termos de implementação. Para este levantamento, foi utilizada uma versão da plataforma prévia à atribuição desta dissertação.

5.5.1 Padrão de Orientação

Para este padrão foram obtidos os seguintes resultados (prévios à melhoria):

Para 61 aplicações:

- em 43 aplicações não foi detetado erro no padrão
- em 18 aplicações foi detetado erro no padrão

Enquanto que, depois de implementada a melhoria, foram registados os seguintes resultados:

Para 61 aplicações:

- em 39 aplicações não foi detetado erro no padrão
- em 22 aplicações foi detetado erro no padrão

Fazendo, agora, uma comparação entre os dois conjuntos de dados obtidos podemos verificar que, após a melhoria ser implementada o número de aplicações em que foram detetados erros do padrão aumentou. Ora, tendo em conta que o objetivo desta ferramenta é a deteção de erros, é possível verificar que a cobertura providenciada pela mesma aumentou. Isto deve-se ao facto de, como referido anteriormente, ter sido adicionado mais um ciclo de verificações a este padrão, ao qual corresponde, neste caso, a mais uma rotação de ecrã e a sua posterior análise.

5.5.2 Padrão *Back*

Para 61 aplicações:

- em 8 aplicações não foi detetado erro no padrão
- em 53 aplicações foi detetado erro no padrão

Enquanto que, depois de implementada a melhoria, foram registados os seguintes resultados:

Para 61 aplicações:

- em 45 aplicações não foi detetado erro no padrão
- em 16 aplicações foi detetado erro no padrão

Analisando os resultados prévios à implementação desta melhoria é verificável que a ferramenta tinha um alto grau de deteção de erro, porém este deve-se ao facto de no padrão não melhorado ser apenas verificado se o ecrã resultante seria igual ao anterior daquele em que era invocado o *back*.

Relembrando as ações passíveis de serem tomadas pelo *back*:

- um elemento do ecrã que esteja em destaque (*highlight*), é retirado desse estado quer por um clique noutra secção do ecrã, ou pressionando o butão *back*.
- quando existem janelas flutuantes no ecrã da aplicação, existe a opção de estas serem fechadas com um clique no botão *back*
- ao introduzir dados numa aplicação, como, por exemplo, texto, o teclado virtual é aberto e a maneira de o fechar passa por invocar o *back*.

Atendendo, agora, aos resultados obtidos após a implementação dos pontos acima referidos, verificou-se que o número de aplicações nas quais o padrão está corretamente implementado aumenta substancialmente. Isto é justificado pelo facto de a ação do *back* não se restringir apenas à navegação dentro da aplicação, mas, também, à gestão de certos elementos dentro de um ecrã.

Estando as experiências referentes às melhorias implementadas na ferramenta concluídas, resta apenas discutir, na próxima secção, a experiência que tratará de responder à terceira questão de pesquisa.

5.6 Qualidade dos resultados

Nesta secção, após saber que a *iMPAcT Tool* é capaz de detetar erros nas aplicações, o objetivo passa por concluir se as falhas detetadas são realmente falhas da aplicação (QP3 - As falhas detetadas pela ferramenta são, realmente, falhas da aplicação?).

Para esta experiência, foi, em primeira instância, realizado um levantamento manual das aplicações a serem testadas de modo a existir um ponto de comparação com a avaliação efetuada através da ferramenta.

Tendo isto em conta, a experiência conduzida nesta secção trata-se, em suma, de uma análise dos positivos reais e falsos , tal como dos negativos reais e falsos . Isso significa que:

- Um positivo real significa que uma falha detetada pela ferramenta, é, de facto, uma falha da aplicação (funciona corretamente);
- Um falso positivo trata-se de uma falha detetada pela ferramenta não é realmente uma falha da aplicação;
- Um falso negativo acontece quando a ferramenta falha na identificação de um erro presente na aplicação;
- Um negativo real significa que a ferramenta não deteta uma falha que não existe na aplicação (funciona corretamente).

No âmbito desta dissertação, e, tendo em conta o facto de os padrões do *back* e orientação já terem sido submetidos a este tipo de testes anteriormente, o único padrão visado nesta experiência será o padrão de chamadas. Como tal, é apresentada de seguida a tabela referente aos resultados verificados:

Tabela 5.2: Resultados finais do padrão de chamadas

Inspeção Manual	iMPAcT Tool		
		Deteta falha	Não deteta falha
	É falha	8	0
	Não é falha	3	-

Tendo em conta que o objetivo é encontrar falhas, a contagem de negativos reais não é considerada nesta experiência.

5.7 Conclusões

Neste capítulo foram apresentadas três experiências distintas. A primeira destas tinha como objetivo verificar se a *iMPAcT Tool* foi capaz de detetar falhas para o novo padrão adicionado, respondendo, assim, à primeira questão de pesquisa, colocada no início deste capítulo. Posteriormente, foi descrita a segunda experiência, responsável por responder à segunda questão de pesquisa. Para tal, foi realizada uma comparação entre dois conjuntos de resultados, o primeiro obtido com uma versão anterior da ferramenta e o segundo com a versão com as melhorias implementadas. Por fim, foi demonstrada a terceira experiência, em que foi efetuado um levantamento manual das falhas das aplicações sob teste para, depois, ser comparado com os resultados obtidos usando a *iMPAcT Tool*. Esta experiência visava responder à terceira e última das perguntas colocadas na primeira secção deste capítulo. Os resultados destas experiências encontram-se, na integra, nos anexos no final do documento.

As respostas para as questões levantadas podem ser encontradas no capítulo 6, tal como uma breve discussão sobre cada uma destas e as conclusões relativas ao trabalho desenvolvido para esta dissertação.

Capítulo 6

Conclusões e Trabalho Futuro

No capítulo anterior, capítulo 5, foram levantadas algumas questões, cujas respostas visam validar o trabalho desenvolvido nesta dissertação.

Na secção 6.1 são apresentadas as respostas às perguntas referidas.

Por fim, na secção 6.3 é feito um sumário dos itens expostos ao longo deste trabalho, de modo a dar a dissertação como terminada.

6.1 Discussão

6.1.1 QP1: A *iMPAcT Tool* é capaz de identificar falhas em aplicações Android, tendo em conta o novo padrão acrescentado à ferramenta?

De modo a dar resposta a esta questão foi realizada uma experiência capaz de demonstrar a capacidade da *iMPAcT Tool* de testar o novo padrão adicionado. Pela experiência conduzida na secção 5.4 do capítulo 5 pode ser concluído que o novo padrão foi implementado na ferramenta com sucesso. Ou seja, a *iMPAcT Tool* é capaz de detetar erros para o padrão das chamadas.

6.1.2 QP2: Os padrões melhorados aumentaram a capacidade de identificar falhas nas aplicações sob teste?

Esta questão é respondida pela experiência conduzida na secção 5.5. Analisando os resultados obtidos, é possível verificar que o número de erros detetados pela ferramenta, influenciado pelas novas verificações nos padrões *back* e de orientação, aumentou, verificando-se, assim, um acréscimo à cobertura de testes fornecida pela *iMPAcT Tool*.

6.1.3 QP3: As falhas detetadas pela ferramenta são, realmente, falhas da aplicação?

Para fornecer uma resposta a esta última questão foi realizada a experiência analisada na secção 5.6. Tendo em conta os resultados obtidos, podemos afirmar que a ferramenta, apesar de apresentar um pequeno número de falsos positivos, é, na maior parte dos casos estudados, bem sucedida.

6.2 Trabalho Futuro

De modo a estender, futuramente, a ferramenta, podem ser tomadas duas abordagens:

- **Aumentar o comportamento a ser testado**: Adicionar novos padrões ao catálogo, de maneira a aumentar a cobertura de testes a aplicações e melhorar os resultados finais. Apesar, de o catálogo já se encontrar num estado aceitável, é importante aumentar o número de padrões presente neste, especialmente, tendo em conta a grande capacidade de mudança do mercado.

- **Melhorar a visualização dos resultados de teste**: Neste momento, os resultados dos teste são demonstrados num ficheiro *log* em que são registados os padrões e as falhas encontrados. Esta visualização pode ser melhorada, sendo apresentada, por exemplo, num ficheiro *pdf*, de modo a apresentar a informação de uma maneira mais estrutura e de mais fácil leitura por parte do utilizador.

6.3 Conclusões

No trabalho realizado nesta dissertação, foi tornado óbvio que a área de testes e mercado para aplicações móveis, especialmente Android, tornou-se enorme. Este é um mercado que, de momento, gera milhões e que continua a apresentar um crescimento exponencial.

Tendo isto em conta, torna-se difícil de desvendar tudo o que já existe sobre o tema. Como estamos a falar de um mercado que apresenta tantas mudanças e inovação, é necessária uma constante pesquisa. Quase diariamente, somos deparados com novas informações, abordagens, ferramentas, dispositivos, etc...

Assim, a solução apresentada, continuada nesta dissertação, tenta satisfazer uma necessidade que, de momento, não se encontra facilitada. Esta necessidade é a de automação dos testes com a maior cobertura possível, reduzindo o tempo e custo que este processo acarreta para empresas e *developers*.

Como o esperado com o fim desta dissertação, a *iMPAcT Tool* melhorou a sua qualidade de testes a aplicações Android, sendo adicionado um novo padrão ao seu catálogo, tal como duas melhorias a padrões, aumentando a sua cobertura de testes, assim como o comportamento capaz de ser testado.

O trabalho alcançado tem potencial para ser ainda mais estendido, sendo melhorado com ainda mais padrões adicionados ao catálogo. Pode ser, também, melhorada a sua visualização e, eventualmente, dar espaço ao aparecimento de novas abordagens capazes de garantir que as aplicações desenvolvidas são apresentadas ao utilizador na sua melhor versão possível: Rápida, limpa e, acima de tudo, sem falhas.

Referências

[AFM12] Domenico Amalfitano, Anna Ritta Fasolino e Atif M. Memon. Using gui ripping for automated testing of android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012*, page 258, 2012.

[BN] N. Butcher e R Nurik. Android design in action: Navigation anti-patterns. Disponível em http://www.allreadable.com/ff647Z8N, acedido a última vez em 5 de Junho de 2018.

[CPFA10] Marco Cunha, Ana C.R. Paiva, Hugo Sereno Ferreira e Rui Abreu. PETTool: A pattern-based GUI testing tool. *ICSTE 2010 - 2010 2nd International Conference on Software Technology and Engineering, Proceedings*, 1(November), 2010. doi:10.1109/ICSTE.2010.5608882.

[CPN14] Pedro Costa, Ana C. R. Paiva e Miguel Nabuco. Pattern based gui testing for mobile applications. In *Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014*, pages 66–74, 2014.

[DB05] Arie Van Deursen e Elizabeth Burd. Software reverse engineering. *Journal of Systems and Software*, pages 209–211, Setembro 2005.

[DG08] Christoph Denzler e Dominik Gruntz. Design patterns. In *Proceedings of the 13th international conference on Software engineering - ICSE '08*, page 801. ACM Press, 2008.

[EGV02] Ralph Johnson Erich Gamma, Richard Helm e John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. 2002.

[FFI+04] Ira R. Forman, Nate Forman, Dr. John Vlissides Ibm, Ira R. Forman e Nate Forman. Java reflection in action, 2004.

[GH04] Hassan Gomaa e Mohamed Hussein. Software reconfiguration patterns for dynamic evolution of software architectures. In *Proceedings - Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pages 79–88, 2004.

[Gooa] Google.

[Goob] Google. Testing support library. Disponível em https://developer.android.com/topic/libraries/testing-support-library/index.html#Espresso, acedido a última vez em 21 de Janeiro de 2018.

[Imp] Genarro Imparato. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 83:284–291. IEEE.

[MBN03] Atif Memon, Ishan Banerjee e Adithya Nagarajan. Gui ripping: Reverse engineering of graphical user interfaces for testing. In *Proceedings - Working Conference on Reverse Engineering, WCRE*, pages 260–269, 2003.

[MP] Inês Coimbra Morgado e Ana C.R. Paiva. In *The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)*, pages 760–762. ANT.

[MP14a] M. L. M. Rodrigo Moreira e Ana C. R. Paiva. Towards a Pattern Language for Model-Based GUI Testing. *19th European Conference on Pattern Languages of Programs (EuroPLoP 2014)*, 2014. doi:10.1145/2721956.2721972.

[MP14b] R.M.L.M. Moreira e A C R Paiva. A GUI modeling DSL for pattern-based GUI testing PARADIGM. *ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2014.

[MP14c] Rodrigo M.L.M. Moreira e Ana C.R. Paiva. PBGT tool: an integrated modeling and testing environment for pattern-based GUI testing. *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering - ASE '14*, 2014.

[MP15a] Ines Morgado e Ana C. R. Paiva. Test patterns for android mobile applications. *Proceedings of the 20th European Conference on Pattern Languages of Programs*, pages 1–7, 2015.

[MP15b] Ines Coimbra Morgado e Ana C. R. Paiva. The impact tool: Testing ui patterns on mobile applications. In *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pages 876–881, 2015.

[MP15c] Ines Coimbra Morgado e Ana C. R. Paiva. Testing approach for mobile applications through reverse engineering of ui patterns. *Sixth International Workshop on Testing Techniques for Event BasED Software*, pages 42–49, 2015.

[MP17] Inês Coimbra Morgado e Ana C.R. Paiva. Mobile GUI testing. *Software Quality Journal*, Setembro 2017.

[MPF14] Inês Coimbra Morgado, Ana C.R. Paiva e João Pascoal Faria. Automated pattern-based testing of mobile applications. In *Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014*, 2014.

[MPM13] Rodrigo M. L. M. Moreira, Ana C. R. Paiva e Atif Memon. A pattern-based approach for gui modeling and testing. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 288–297. IEEE, 2013.

[MPNM17] Rodrigo M. L. M. Moreira, Ana Cristina Paiva, Miguel Nabuco e Atif Memon. Pattern-based GUI testing: Bridging the gap between design and quality assurance. *Software Testing, Verification and Reliability*, 27(3):e1629, may 2017. URL: http://doi.wiley.com/10.1002/stvr.1629, doi:10.1002/stvr.1629.

[Ris98] Linda Rising. *The Patterns Handbook: Techniques, Strategies and Applications*. Cambridge University Press, Fourth edition, 1998.

[Sha94] Mary Shaw. Patterns for software architectures. *First Annual Conference on the Pattern Languages of Programming*, (August 1994):453–462, 1994.

[Staa] Statista. Global revenue from smartphone sales from 2013 to 2017 (in billion u.s. dollars). Disponível em https://www.statista.com/statistics/237505/global-revenue-from-smartphones-since-2008/, acedido a última vez em 7 de Fevereiro de 2018.

[Stab] Statista. Number of smartphone users worldwide from 2014 to 2020 (in billions). Disponível em https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, acedido a última vez em 7 de Fevereiro de 2018.

[Stac] Statista. Principais desafios para o teste de apps. Disponível em https://www.statista.com/statistics/500630/worldwide-mobile-and-multichannel-application-testing-challenges/, acedido a última vez em 21 de Janeiro de 2018.

[TKH11] Tommi Takala, Mika Katara e Julian Harty. Experiences of system-level model-based GUI testing of an android application. In *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*, pages 377–386, 2011. doi:10.1109/ICST.2011.11.

REFERÊNCIAS

Anexos

Testes manuais

Name	Orientation	Back
Lottie	Correct	Incorrect
Material Dialogs Library Demo	Incorrect	Incorrect
Advanced RecyclerView Examples	Correct	Incorrect
uCrop	Correct	Incorrect
k-9 mail	Correct	Incorrect
Timber	Incorrect	Incorrect
Forkhub for Github	Correct	Correct
My Diary (unoficial)	Correct	Incorrect
Materialistic - Hacker News With Account	Correct	Correct
Loop - Habit Tracker	Correct	Incorrect
ConnectBot	Correct	Incorrect
RedReader	Correct	Incorrect
LibreTorrent	Correct	Incorrect
Simple Calendar	Incorrect	Incorrect
OneBusAway	Correct	Incorrect
Simple Gallery	Incorrect	Correct
MHGen Database	Correct	Incorrect
And Bible	Correct	Incorrect
AntennaPod	Correct	Incorrect
Debatekeeper	Correct	Incorrect
Simple Draw	Correct	Incorrect
AnkiDroid Flashcards	Incorrect	Incorrect
Chroma Doze	Incorrect	Incorrect
Clementine Remote	Correct	Incorrect
CPU Stats	Correct	Incorrect
aMetro	Incorrect	Incorrect
Calendar Notifications	Correct	Incorrect

Simple Flashlight	Correct	Correct
Flym News Reader	Correct	Incorrect
MIFARE Classic Tool	Correct	Incorrect
Anuto TD	Correct	Incorrect
Blokish	Correct	Incorrect
Simple File Manager	Incorrect	Correct
Glucosio	Incorrect	Incorrect
Lightning	Correct	Incorrect
BankDroid	Incorrect	Incorrect
GPS Logger for Android	Correct	Incorrect
OpenTasks	Correct	Incorrect
OSRS Helper	Incorrect	Incorrect
Password Store	Correct	Incorrect
pMetro for Android	Correct	Incorrect
Poet Assistant (English)	Correct	Correct
Primitive FTPd	Correct	Incorrect
qBittorent Controler	Correct	Incorrect
RGB Tool	Correct	Incorrect
Shader Editor	Incorrect	Incorrect
Simple Last fm Scrobbler	Correct	Incorrect
Unit Converter Ultimate	Incorrect	Incorrect
Vlille Checker	Correct	Incorrect
Web Opac: 1,000+ libraries	Incorrect	Incorrect
Weechat Android	Incorrect	Incorrect
WiFi Automatic	Correct	Incorrect
WifiAnalyser (open-source)	Correct	Incorrect
WiGLE Wifi Wardriving	Correct	Incorrect
ShutUp!	Incorrect	Incorrect
Swiftnotes	Incorrect	Incorrect
CycleStreet	Correct	Incorrect
OpenBikeSharing	Correct	Incorrect
Amaze File Manager	Correct	Correct
Photo Affix	Incorrect	Incorrect
Omni Notes	Correct	Correct

Teste padrão de chamadas

Name	Calls
Lottie	Correct
Material Dialogs Library Demo	Correct
Advanced RecyclerView Examples	Correct
uCrop	Incorrect
k-9 mail	Correct
Timber	Correct
Forkhub for Github	Correct
My Diary (unoficial)	Correct
Materialistic - Hacker News With Account	Correct
Loop - Habit Tracker	Incorrect
ConnectBot	Correct
RedReader	Correct
LibreTorrent	Correct
Simple Calendar	Correct
OneBusAway	Incorrect
Simple Gallery	Correct
MHGen Database	Correct
And Bible	Correct
AntennaPod	Correct
Debatekeeper	Correct
Simple Draw	Correct
AnkiDroid Flashcards	Correct
Chroma Doze	Correct
Clementine Remote	Correct
CPU Stats	Correct
aMetro	Incorrect
Calendar Notifications	Correct
Simple Flashlight	Correct
Flym News Reader	Correct
MIFARE Classic Tool	Incorrect
Anuto TD	Correct
Blokish	Correct
Simple File Manager	Correct
Glucosio	Correct
Lightning	Correct
BankDroid	Correct

GPS Logger for Android	Incorrect
OpenTasks	Correct
OSRS Helper	Correct
Password Store	Correct
pMetro for Android	Correct
Poet Assistant (English)	Correct
Primitive FTPd	Correct
qBittorent Controler	Correct
RGB Tool	Correct
Shader Editor	Correct
Simple Last fm Scrobbler	Correct
Unit Converter Ultimate	Correct
Vlille Checker	Incorrect
Web Opac: 1,000+ libraries	Correct
Weechat Android	Correct
WiFi Automatic	Correct
WifiAnalyser (open-source)	Correct
WiGLE Wifi Wardriving	Correct
ShutUp!	Correct
Swiftnotes	Correct
CycleStreet	Incorrect
OpenBikeSharing	Incorrect
Amaze File Manager	Incorrect
Photo Affix	Correct
Omni Notes	Incorrect

Teste a melhorias implementadas

Name	Back	Orientation
Lottie	Correct	Correct
Material Dialogs Library Demo	Correct	Incorrect
Advanced RecyclerView Examples	Correct	Incorrect
uCrop	Incorrect	Incorrect
k-9 mail	Correct	Correct
Timber	Correct	Incorrect
Forkhub for Github	Correct	Correct
My Diary (unoficial)	Incorrect	Correct
Materialistic - Hacker News With Account	Correct	Correct

Loop - Habit Tracker	Incorrect	Correct
ConnectBot	Correct	Correct
RedReader	Correct	Correct
LibreTorrent	Correct	Correct
Simple Calendar	Correct	Incorrect
OneBusAway	Incorrect	Correct
Simple Gallery	Correct	Incorrect
MHGen Database	Correct	Correct
And Bible	Incorrect	Correct
AntennaPod	Correct	Correct
Debatekeeper	Correct	Correct
Simple Draw	Correct	Correct
AnkiDroid Flashcards	Incorrect	Incorrect
Chroma Doze	Correct	Incorrect
Clementine Remote	Correct	Correct
CPU Stats	Correct	Correct
aMetro	Incorrect	Incorrect
Calendar Notifications	Correct	Correct
Simple Flashlight	Correct	Correct
Flym News Reader	Correct	Correct
MIFARE Classic Tool	Incorrect	Correct
Anuto TD	Correct	Correct
Blokish	Correct	Correct
Simple File Manager	Incorrect	Incorrect
Glucosio	Incorrect	Incorrect
Lightning	Correct	Correct
BankDroid	Correct	Incorrect
GPS Logger for Android	Correct	Correct
OpenTasks	Correct	Correct
OSRS Helper	Correct	Incorrect
Password Store	Correct	Incorrect
pMetro for Android	Correct	Correct
Poet Assistant (English)	Correct	Incorrect
Primitive FTPd	Correct	Correct
qBittorent Controler	Correct	Correct
RGB Tool	Correct	Correct
Shader Editor	Correct	Incorrect
Simple Last fm Scrobbler	Correct	Correct

Unit Converter Ultimate	Correct	Incorrect
Vlille Checker	Correct	Correct
Web Opac: 1,000+ libraries	Incorrect	Incorrect
Weechat Android	Correct	Incorrect
WiFi Automatic	Correct	Correct
WifiAnalyser (open-source)	Correct	Correct
WiGLE Wifi Wardriving	Correct	Correct
ShutUp!	Incorrect	Incorrect
Swiftnotes	Correct	Incorrect
CycleStreet	Incorrect	Correct
OpenBikeSharing	Incorrect	Correct
Amaze File Manager	Incorrect	Correct
Photo Affix	Correct	Incorrect
Omni Notes	Incorrect	Correct