PODER EXECUTIVO

MINISTÉRIO DA EDUCAÇÃO

UNIVERSIDADE FEDERAL DO AMAZONAS

INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Jonathas Silva dos Santos

INSUMOS PARA A UTILIZAÇÃO DO CRITÉRIO DE TESTE BASEADO EM ERROS PARA APLICAÇÕES MÓVEIS

MANAUS

2016

JONATHAS SILVA DOS SANTOS

INSUMOS PARA A UTILIZAÇÃO DO CRITÉRIO DE TESTE BASEADO EM ERROS PARA APLICAÇÕES MÓVEIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, PPGI, da Universidade Federal do Amazonas como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Arilo Claudio Dias Neto

MANAUS

2016

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Santos, Jonathas Silva dos
S237i Insumos para a utilização do critério de teste baseado em erros para aplicações móveis / Jonathas Silva dos Santos. 2016
111 f.: il.; 31 cm.

Orientador: Arilo Claudio Dias Neto
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. modelo de falhas. 2. aplicações móveis. 3. teste de aplicações móveis. 4. teste baseado em defeitos. I. Dias Neto, Arilo Claudio II. Universidade Federal do Amazonas III. Título

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UFAM

FOLHA DE APROVAÇÃO

"Insumos para a utilização de teste baseado em erro em aplicações móveis"

JONATHAS SILVA DOS SANTOS

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Arilo Claudio Dias Neto
Prof. Arilo Claudio Dias Neto - PRESIDENTE

Prof. Lucas Carvalho Cordeiro - MEMBRO INTERNO

Prof. Auri Marcelo Rizzo Vincenzi - MEMBRO EXTERNO

Manaus, 23 de Fevereiro de 2016

À minha família, Aldener, Cláudia e Stephany.

À Larissa Mahall.

“Elevo os meus olhos para os montes; de onde me vem o socorro?

O meu socorro vem do Senhor, que fez os céus e a terra.

Não deixará vacilar o teu pé; aquele que te guarda não dormitará.

Eis que não dormitará nem dormirá aquele que guarda a Israel.

O Senhor é quem te guarda; o Senhor é a tua sombra à tua mão direita.

De dia o sol não te ferirá, nem a lua de noite.

O Senhor te guardará de todo o mal; ele guardará a tua vida.

O Senhor guardará a tua saída e a tua entrada, desde agora e para sempre”

Salmos 121:1-8

AGRADECIMENTOS

A DEUS, pelo dom da vida e por me permitir concluir essa etapa.

Ao amigo e orientador, Prof. Dr. Arilo Claudio Dias Neto, pelo apoio, sugestões e profissionalismo exemplar na orientação deste trabalho. Pelo qual minha admiração é cada dia maior, e que em muitos momentos, sua importância extrapolou a de orientador.

Aos meus familiares, em especial, aos meus pais Aldener e Cláudia e minha irmã Stephany, pelo amor, compreensão, incentivo, e por entender as constantes ausências.

À Larissa Mahall pelo amor incondicional e pelo companheirismo, e por dizer o sim mais importante da minha vida até o momento. Obrigado por nunca soltar da minha mão, e por me permitir faze-la feliz.

A todos aqueles que, de alguma forma, contribuíram e colaboraram para a realização deste trabalho, em especial: Awdren, Silvia, Rodrigo, Renata e Karina. Contribuindo sempre com excelentes *feedbacks* e com um ambiente agradável para a condução da pesquisa. Além do possível título de mestre, afirmo com todas as evidências empíricas que ganhei cinco amigos para toda a vida.

Aos demais companheiros do laboratório Experts, todos os professores e funcionários do Instituto de Computação da UFAM que tanto fazem em nosso favor.

Ao CNPq, pelo apoio financeiro. E ao INDT, por também colaborar financeiramente e com a condução da pesquisa.

Resumo da Dissertação de Mestrado apresentada à UFAM/AM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática (M.Sc.)

INSUMOS PARA A UTILIZAÇÃO DO CRITÉRIO DE TESTE BASEADO EM FALHAS PARA APLICAÇÕES MÓVEIS

Jonathas Silva dos Santos

Fevereiro / 2016

Orientador: Arilo Claudio Dias Neto

No contexto da computação atual, plataformas móveis têm se apresentado como uma das plataformas de maior crescimento, e isso se deve principalmente pelo aumento da popularidade dos dispositivos móveis. Com isso, tais aplicações estão se tornando indispensáveis na rotina dos seus usuários, assumindo muitas vezes papéis críticos, por conta dos dados que manipulam.

Diante desse cenário, garantir a qualidade de tais aplicações é uma busca constante em trabalhos acadêmicos mais recentes. Para isso, a Engenharia de Software fornece métodos, técnicas, critérios e atividades que buscam garantir a qualidade de software de um modo geral. Tais atividades são denominadas Verificação e Validação (V&V), sendo que Teste de Software está inserido nesse contexto. Porém algumas atividades necessitam ser adaptadas, por conta das características que envolvem a plataforma móvel.

No contexto dessa adaptação, o conhecimento prévio de possíveis falhas pode fornecer informações importantes para fornecer insumos para aplicação de atividades de teste especificamente para o contexto de aplicações móveis.

Esta dissertação apresenta a definição do modelo de falhas MOBPI-FM (*MOBile Platform Independent – Fault Model*) específico para aplicações construídas para plataforma móvel. Esse modelo foi gerado de forma sistemática utilizando a técnica HAZOP (*Hazard and Operability Study*) e é composto por três níveis de utilização (Falha, Erro e Defeito). O nível de falhas proposto no modelo MOBPI-FM foi avaliado em relação a cobertura de falhas reais por meio de um estudo de viabilidade realizado com 58 aplicações móveis da plataforma Android. Os resultados indicaram que 86% das falhas avaliadas foram cobertas pelo modelo.

Palavras-chave: Modelo de falhas; aplicações móveis; teste de aplicações móveis; teste baseado em defeitos.

Abstract of Master's thesis presented to UFAM/AM as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SUPPLIES TO USAGE OF FAULT-BASED TESTING CRITERIA TO MOBILE APPLICATIONS

Jonathas Silva dos Santos

February / 2016

Advisor: Arilo Claudio Dias Neto

In the context of current computing, mobile platforms have been presented as one of the most growth platforms, and this is mainly by the increased popularity of mobile devices. Furthermore, such applications are becoming indispensable in the routine of their users, assuming often critical roles, due to the data they handle.

In this scenario, ensuring the quality of such applications is a usual purpose in recent academic work. For this, the Software Engineering field provides methods, techniques, standards and activities that seek to ensure software quality in general. Such activities are named Verification and Validation (V&V), and Software Testing is inserted into this context. However, some activities need to be adapted on account of the characteristics involving the mobile platform.

In this adaptation, the knowledge of possible failures can provide important information to supplies for utilization of testing activities, specifically to mobile applications.

This thesis presents the definition of the fault model MOBPI-FM (Mobile Platform Independent - Fault Model) specific for mobile applications platform. This model was generated systematically by using the HAZOP technique (Hazard and Operability Study) and consists of three levels of use (Fault, error and defect). The fault level proposed in MOBPI-FM model was evaluated in relation to actual fault coverage through a feasibility study with 58 mobile applications in the Android platform. The results indicated that 86% of the defects were evaluated covered by the model.

Keywords: fault model; mobile applications; mobile testing; fault-based testing

Sumário

AGRADECIMENTOS VI
LISTA DE TABELAS XII
LISTA DE FIGURAS XIII
CAPÍTULO 1 – INTRODUÇÃO 1
1.1. Contextualização e Motivação 1
1.2. Descrição do Problema 3
1.3. Hipótese 4
1.4. Objetivos 4
1.5. Metodologia utilizada 5
1.6. Estrutura do Documento 6
CAPÍTULO 2 – REFERENCIAL TEÓRICO E TRABALHOS CORRELATOS 8
2.1. Aplicações Móveis 8
2.1.1. Características de aplicações móveis 9
2.1.2. Aplicações Móveis em Android 10
2.2. Teste de Software 11
2.2.1. Conceito de Falha e Erro 11
2.2.2. Teste Baseado em Defeitos 12
2.2.3. Modelo de falhas 13
2.2.4. Testes para Aplicações Móveis 16
2.3. Técnica de desvios HAZOP 17
2.4. Considerações sobre o capítulo 19
CAPÍTULO 3 – ESTUDOS PRELIMINARES DE APOIO À CONCEPÇÃO DE UM MODELO DE FALHAS PARA APLICAÇÕES MÓVEIS 21
3.1. Mapeamento sistemático sobre testes baseados em código em aplicações móveis 21
3.2. Estudo exploratório da aplicação de operadores Java no contexto Android 24
3.2.1. Planejamento e execução do estudo 24
3.2.2. Resultados obtidos 29
3.2.3. Ameaças a Validade do Estudo 33
3.3. Mapeamento sistemático sobre definição de operadores de mutação 34
3.3.1. Questões de pesquisa 35
3.3.2. Identificação e Seleção de Estudos Primários 35
3.3.3. Formulário de extração de dados 35
3.3.4. Execução do Estudo de Mapeamento Sistemático 36
3.3.5. Análise dos dados 36
3.3.6. QP1: As fontes utilizadas para a definição dos operadores 38

3.3.7. QP2: abordagens utilizadas para projetar operadores de mutação 39
3.3.8. QP3: Estratégias utilizadas para projetar operadores de mutação 40
3.4. Discussões sobre os estudos preliminares 43
CAPÍTULO 4 – MOBPI-FM: UM MODELO DE FALHAS ESPECÍFICO PARA APLICAÇÕES MÓVEIS 45
4.1. Mapeamento de Características de Mobilidade 46
4.1.1. Conexão 47
4.1.2. Energia 49
4.1.3. Configurações 50
4.1.4. Entrada de dados 51
4.1.5. Interface com o Dispositivo 52
4.1.6. Interrupções 53
4.1.7. Acesso a Dados 54
4.2. Aplicação da técnica HAZOP 55
4.3. Verificação de significância e Refinamento 57
4.4. Apresentação do modelo 59
4.4.1. Nível de Falhas 61
4.4.2. Nível de Erros e Defeitos 64
4.4.2.1. Comparação com outros trabalhos 65
4.5. Considerações sobre o capítulo 68
CAPITULO 5 - AVALIAÇÃO DA COBERTURA DO MODELO MOBPI-FM EM FALHAS DE APLICAÇÕES MÓVEIS REAIS 70
5.1. Definição do estudo 70
5.1.1. Definição do escopo e objetivos 70
5.2. Planejamento do estudo 71
5.2.1. Formulação de hipóteses 71
5.2.2. Seleção de aplicações móveis 71
5.2.3. Seleção dos responsáveis pelo mapeamento 72
5.2.4. Instrumentos 73
5.2.5. Ameaças a validade 74
5.3. Condução do estudo 75
5.4. Análise dos resultados obtidos 77
5.4.1. Cobertura geral 77
5.4.2. Cobertura por categoria de aplicações móveis 83
5.4.3. Cobertura por característica 84
5.5. Considerações sobre o capítulo 85
CAPÍTULO 6 – CONSIDERAÇÕES FINAIS 87
6.1. Contribuições 87

6.2. Limitações da pesquisa 88
6.3. Trabalhos futuros 89
REFERÊNCIAS 91
Apêndice I - Protocolo do Mapeamento Sistemático sobre testes estruturais e baseado em falhas 94
PROTOCOLO 94
RESULTADO 97

LISTA DE TABELAS

Tabela 1. Características específicas de uma aplicação móvel. 9
Tabela 2. Palavras guia da técnica HAZOP utilizadas na indústria química. 18
Tabela 3. Operadores de mutação Java disponíveis na ferramenta PIT. 27
Tabela 4. Resultados obtidos na execução do estudo. 29
Tabela 5. Densidade de mutantes em cada componente Android. 33
Tabela 6. Campos de extração para os trabalhos. 35
Tabela 7. Sumarização dos resultados da tabela. 36
Tabela 8. Trabalhos encontrados no mapeamento sistemático. 37
Tabela 9. Estratégias selecionadas no mapeamento. 41
Tabela 10. Palavras-guia HAZOP utilizadas na geração dos desvios. 56
Tabela 11. Amostra de utilização da técnica HAZOP. 57
Tabela 12. Caracterização dos pesquisadores que compuseram a equipe de verificação. .. 58
Tabela 13. Amostra de avaliação dos desvios com relação à significância. 58
Tabela 14 Composição dos identificadores utilizados no modelo. 60
Tabela 15. Falhas referentes a característica de Entrada de Dados. 61
Tabela 16. Falhas da característica Conexão. 61
Tabela 17. Falhas referentes a característica de configuração do dispositivo. 62
Tabela 18. Falhas referentes ao gerenciamento de energia. 62
Tabela 19. Falhas referentes a característica de Interface com o dispositivo. 63
Tabela 20. Falhas referentes a característica de Interrupções. 63
Tabela 21. Falhas referentes a característica de Acesso a Dados. 63
Tabela 22. Taxonomia das meta-informações contidas nos erros 64
Tabela 23. Amostra no nível de erros referentes à falha F-CN01 64
Tabela 24. Mapeamento dos operadores de mutação para Android com itens do nível de defeitos do modelo MOBPI-FM 66
Tabela 25. Mapeamento da classificação de falhas com o nível de erros do modelo MOBPI-FM 67
Tabela 26. Aplicações móveis selecionados para o estudo. 72
Tabela 27. Caracterização dos desenvolvedores responsáveis pela validação. 73
Tabela 28. Exemplo da diferença entre uma falha RN e uma falha específica 76
Tabela 29. Resultados da verificação de cobertura. 78
Tabela 30. Quantidade de falhas de aplicações reais para cada falha do modelo MOBPI-FM. 80
Tabela 31. Sumarização da cobertura de falhas por categoria de aplicação. 83
Tabela 32. Sumarização da cobertura de falhas por categoria de aplicação. 85
Tabela 33. Instanciação do paradigma GQM 94
Tabela 34. Definição do padrão PICO para a *string* do mapeamento 96
Tabela 35 Resultado do mapeamento sistemático 97

LISTA DE FIGURAS

Figura 1. Metodologia adotada na dissertação, adaptada de (SPÍNOLA *et al*., 2008) 5
Figura 2. Conceitos ordenados por ordem de ocorrência em um programa. 12
Figura 3. Exemplo de classe modificada pela ferramenta PIT. 25
Figura 4. Exemplos dos mutantes gerados para a linha 7 da Figura 4. 26
Figura 5. Capturas de tela das aplicações móveis utilizadas. 28
Figura 6. Resultado da análise dos mutantes para a aplicação *Robolectric Sample.* 30
Figura 7. Resultado da análise dos mutantes na aplicação *BluetoothChat*. 31
Figura 8. Mutantes mortos por *timeout* na aplicação *RobolectricSample*. 31
Figura 9. Mutantes mortos por *timeout* na aplicação *BluetoothChat*. 32
Figura 10. Distribuição dos mutantes em componentes Android. 33
Figura 11. Artigos distribuídos por ano de publicação. 37
Figura 12. Artigos distribuídos por linguagens de programação. 38
Figura 13. Artigos de acordo com as fontes de dados de mutação. 39
Figura 14. Artigos distribuídos por abordagens de mutação. 40
Figura 15. Artigos distribuídos cronologicamente (legenda na Tabela 9). 41
Figura 16. Metodologia de concepção do modelo MOBPI-FM. 45
Figura 17. Modelagem estática das características de aplicações móveis. 47
Figura 18. Característica de Conexão: conexão com a rede. 48
Figura 19. Característica Conexão: conexão por meio de Bluetooth. 49
Figura 20. Característica Conexão: conexão por meio de USB. 49
Figura 21. Característica Energia: consultando o nível da bateria 50
Figura 22. Característica Configurações do Dispositivo: consultando as configurações disponíveis. 50
Figura 23. Característica Entrada de Dados: mapeamento da característica. 51
Figura 24. Característica Interface do Dispositivo: interação com a rotação de tela 52
Figura 25. Característica Interface do Dispositivo: interação com botões físicos. 53
Figura 26. Característica Interrupções: tratamento das interrupções. 54
Figura 27. Característica Acesso a Dados: dados salvos no dispositivo. 55
Figura 28. Característica Acesso a Dados: dados salvos de outras aplicações. 55
Figura 29. Procedimento realizado na aplicação do HAZOP. 56
Figura 30. Arquitetura do modelo MOBPI-FM. 60
Figura 31. Defeito oriundo da instância do erro E-CN01 para a plataforma Android. 65
Figura 32. Representação da condução do estudo. 76
Figura 33. Distribuição dos resultados de cobertura geral. 79
Figura 34. Distribuição das falhas por operação 81
Figura 35. Distribuição das falhas por característica. 84

CAPÍTULO 1 – INTRODUÇÃO

Neste capítulo serão demonstrados o contexto desta dissertação e a motivação. Posterior a isso, serão apresentados seus objetivos, hipótese, metodologia de pesquisa e a organização deste texto.

1.1. Contextualização e Motivação

Pessoas utilizam dispositivos móveis para muitos dos mesmos propósitos que se utilizam os computadores: navegação na Internet, redes sociais, jogos, dentre outros. Com isso, um telefone celular, que outrora se limitava a realizar chamadas telefônicas, evoluiu de tal forma que passou a ser chamado de *smartphone* (do inglês, telefone inteligente) e pode ser conceituado como dispositivo móvel. Tais dispositivos, no cenário atual ultrapassam a função de chamadas telefônicas e alcançam outras aplicações rotineiras, tanto para entretenimento (jogos e redes sociais) quanto para tarefas altamente críticas (movimentações bancárias/financeiras, segurança residencial e tratamentos médicos) (MUCCINI *et al*., 2012).

MEULEN e RIVERA (2014) destacam o grande crescimento da popularidade das aplicações para dispositivos móveis, também chamadas de aplicações móveis. Segundo os autores, esse crescimento só tende a aumentar à medida em que são agregadas novas possibilidades de uso. Ou autores ainda estimam que, por exemplo em 2017, serão realizados mais de 268 bilhões de downloads de aplicações móveis, gerando uma receita diária de mais de $77.000,00 (MEULEN e RIVERA, 2014). Dentre as plataformas móveis atuais, a plataforma Android (GOOGLE ANDROID, 2014) consolida-se como a mais popular do mercado (GLOBALSTATS, 2015). Vários fatores colaboram para esse crescimento, como a grande quantidade de dispositivos móveis disponíveis com esse sistema operacional e também pela grande quantidade de aplicações móveis disponíveis em sua principal loja virtual, a *Google Play*[1]. Outro fator que influencia a sua popularidade é o fato da plataforma ser de código aberto e ter uma vasta documentação disponível, assim como um amplo apoio ferramental e fóruns de discussão (LECHETA, 2013).

A importância das aplicações móveis nos últimos anos para a população em geral é evidente. Falhas durante seu uso podem acarretar diversos problemas, como por exemplo prejuízos financeiros para as empresas de desenvolvimento e de perdas para seus usuários, perdas que podem variar de acordo com a criticidade das aplicações, desde perda de dados de um jogo ou até mesmo dados financeiros. Isso faz com que a garantia de qualidade seja

[1] Google Play: https://play.google.com/store

um fator a ser considerado quando se planeja construir uma aplicação móvel. Para assegurar que tais aplicações tenham a mínima garantia de qualidade, a Engenharia de Software – disciplina que aplica os princípios de engenharia com o objetivo de produzir software de alta qualidade a baixo custo (PRESSMAN e MAXIM, 2014) – evoluiu significativamente na última década.

Por meio de um conjunto de etapas que envolvem o desenvolvimento e aplicação de métodos, técnicas, critérios e ferramentas, a Engenharia de Software busca oferecer meios para que a qualidade dos produtos possa ser alcançada. E um destes meios é o Teste de Software, que tem uma longa história de contribuições para a Engenharia de Software. (MYERS *et al*., 2012) definem teste de software como "o processo de execução de um programa com a intenção de encontrar falhas". O padrão ISO/ICE/IEEE 24765 (IEEE STANDARDS ASSOCIATION, 2010) define teste de software em um escopo mais abrangente como uma atividade em que um sistema ou componente é executado sob condições específicas, os resultados são observados ou registrados, e é feita uma avaliação de alguns aspectos do sistema ou componente.

Para a aplicação de Testes de Software, utilizam-se três diferentes técnicas: testes funcionais, testes estruturais e teste baseado em defeito (DELAMARO *et al*., 2007). A contribuição desta dissertação está relacionada a técnicas de teste baseado em defeitos. Tal técnica utiliza informações sobre os tipos de defeitos mais frequentes no processo de desenvolvimento de software para derivar os requisitos de teste. A ênfase da técnica está nos defeitos que o programador e/ou projetista podem cometer durante o desenvolvimento do software e nas abordagens que podem ser usadas para detectar a sua ocorrência. Semeadura de Defeitos e Análise de Mutantes são critérios de teste relacionados à técnica de teste baseado em defeitos (DELAMARO *et al*., 2007).

Várias pesquisas no meio acadêmico têm realizado vários esforços de padronização de termos referentes a defeitos (mais detalhes no Capítulo 2). Nessa dissertação, os termos engano e defeito serão referenciados como defeito (causa) e os termos erro e falha como falha (consequência) a um comportamento incorreto do programa.

Para o mapeamento dos eventuais defeitos cometidos por programadores ou que resultam em falhas na execução de programas, modelos são propostos em pesquisas acadêmicas para expressar problemas típicos presentes durante o desenvolvimento de sistemas em diferentes plataformas. Estes são chamados de modelos de falhas. Um modelo de falhas identifica relacionamentos e componentes de um sistema sob teste que são mais propensos a ter falhas, visto que não se pode testar tudo (possíveis entradas, caminhos ou estados de um programa), porque o conjunto de possibilidades é demasiadamente grande. O

custo para testar as possibilidades seria enorme para todos, até mesmo para programas mais simples (BINDER, 1999).

Dado este cenário, um dos objetivos de um modelo de falhas é definir uma maneira de projetar um conjunto de testes que exercite o programa suficientemente para revelar a maioria de suas falhas, porém sendo ainda pequeno o suficiente para tornar viável a execução dos testes. Sabendo-se que é praticamente impossível que qualquer sistema seja imune a falhas (PRESSMAN e MAXIM, 2014) um modelo de falhas procura contribuir para a qualidade de software em um paradigma de programação específica ou linguagem.

Assim, a principal motivação desta dissertação é contribuir com pesquisas em testes para aplicações móveis, especificamente com a definição de um modelo de falhas específico para aplicações móveis, chamado MOBPI-FM (*MOBile Platform Independent – Fault Model*), servindo como insumo para a aplicação de técnicas de teste baseado em falhas nesta categoria de programa.

1.2. Descrição do Problema

Mesmo com a contribuição de atividades de teste, a garantia que o produto esteja livre de falhas depende de outros fatores externos. E tal constatação é endossada em plataformas mais recentes, como a plataforma móvel, devido à carência de técnicas, métodos e ferramentas de apoio aos testes. Segundo WASSERMAN (2010), sob o olhar da Engenharia de Software, uma aplicação móvel utiliza as mesmas metodologias, processos e técnicas utilizadas para aplicações embarcadas, mas com uma grande diferença que é a alta integração com informações contextuais fornecidas pelo dispositivo. Logo, as atividades de teste para o contexto móvel têm de ser aprimoradas para que essa integração possa ser feita com qualidade.

Segundo DANTAS *et al*. (2009), a complexidade para se testar uma aplicação móvel exige uma adaptação no processo de teste tradicional, decorrente do fato de que não é possível executar testes da mesma maneira que se realiza em outros tipos de aplicações por conta das características específicas dessa plataforma, principalmente em relação à simulação exata do ambiente onde tal programa irá funcionar.

Isso acontece devido às limitações de recursos existentes nestes dispositivos e a necessidade de interação das aplicações móveis com os demais serviços providos pelo dispositivo, como por exemplo atender chamadas, oscilação de carga da bateria, comunicação por redes Wi-Fi, *Bluetooth*, acesso à câmera fotográfica, dentre outros, o que não pode ser simulado totalmente pelos emuladores providos para cada plataforma (JOORABCHI *et al*., 2013).

O estudo e a aplicação de abordagens de teste em plataformas móveis são fundamentais para diminuir a ocorrência de falhas e, com isso, garantir uma melhor qualidade dos produtos em questão. E este processo ainda se torna mais crítico, pois essas abordagens não cobrem as características peculiares que os dispositivos móveis apresentam, como conectividade, recursos limitados (memória e energia), interface diferenciada, sensibilidade ao contexto, necessidade de adaptação ao ambiente que opera, novos sistemas operacionais e novas linguagens de programação. Segundo (MUCCINI *et al*., 2012), os fatores citados anteriormente influenciam diretamente na atividade de teste para aplicações móveis. Estudos têm sido conduzidos para a plataforma Android, com o objetivo de minimizar a ocorrência de falhas que sejam causados pelos fatores citados.

Para aplicação de critérios de teste relacionados à técnica de teste baseado em defeitos, como por exemplo o critério de teste de mutação, nas demais plataformas, o código fonte é utilizado como insumo (entrada) para a aplicação de tais critérios. Estes critérios partem do princípio que falhas complexas em desenvolvimento de um programa estão associados a defeitos simples. Assim, ao utilizar tais critérios em aplicações desenvolvidas para a plataforma móvel espera-se prevenir erros associados às características citadas por (MUCCINI *et al*., 2012).

O interessante é citar que estes critérios necessitam de um conhecimento prévio dos padrões de defeitos existentes em cada plataforma. Para isso, faz-se necessário o conhecimento prévio de tais defeitos, para que métodos, processos ou ferramentas possam utilizar essa informação para uma melhor adequação e efetividade, e, assim, para melhor garantir a qualidade de aplicações móveis. Portanto, como principal problema a ser resolvido pela dissertação é a falta de um conhecimento prévio de defeitos específicas de aplicações móveis.

1.3. Hipótese

A hipótese definida para este trabalho considera o seguinte cenário:

É possível projetar um modelo de falhas que possa cobrir falhas específicas relacionadas às principais características da plataforma móvel de forma a auxiliar a aplicação de critérios de teste baseado em defeitos para aplicações móveis.

1.4. Objetivos

Este trabalho tem como objetivo definir um modelo de falhas, chamado MOBPI-FM, que descreve um conjunto de possíveis falhas relacionadas a características específicas

presentes em aplicações móveis. Com isso, espera-se contribuir para a utilização de critérios de teste baseado em defeitos no contexto de aplicações móveis. Para alcançar o objetivo geral, são propostos os seguintes objetivos específicos:

- Identificar as características específicas de aplicações móveis na literatura técnica, independentemente da plataforma, indicando como utilizar essas características na garantia de qualidade de aplicações móveis;
- Analisar por meio de estudos empíricos o quanto uma falha pode ser específica para uma aplicação móvel a fim de prover evidências da necessidade de exploração de falhas específicas para plataformas móveis.
- Propor um modelo de falhas independente de plataforma e focado nas características específicas de aplicações móveis de forma sistemática utilizando a técnica de desvios de especificações HAZOP (BRITISH STANDARD, 2005).
- Avaliar a cobertura das falhas propostas no modelo no contexto de aplicações móveis da plataforma Android.

1.5. Metodologia utilizada

Para o alcance dos objetivos previamente propostos, seguiu-se a metodologia representada na Figura 1.

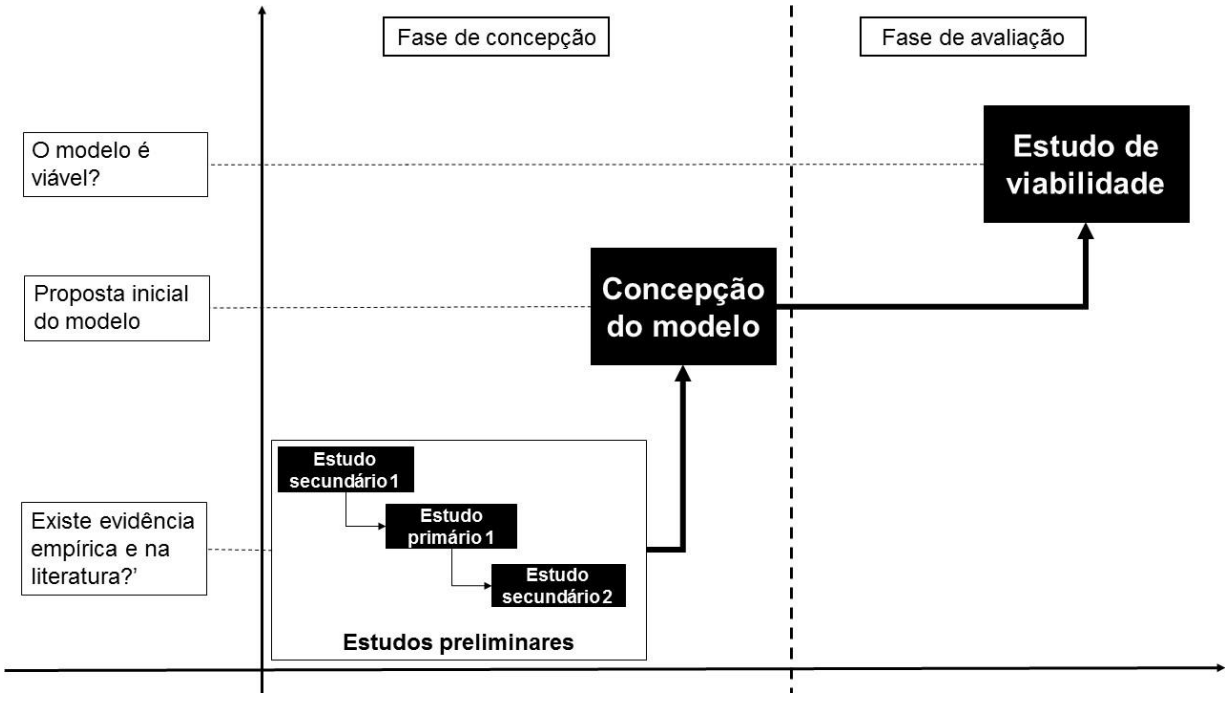


Figura 1. Metodologia adotada na dissertação, adaptada de (SPÍNOLA *et al*., 2008)

Como visualizado na Figura 1, foram realizados três estudos preliminares com o intuito de coletar evidências empíricas sobre a utilização de critérios de teste baseado em código de

aplicações móveis, a saber: um estudo secundário (estudo secundário 1) com o objetivo de caracterizar técnicas de teste estruturais e baseados em código para aplicações móveis; um estudo exploratório com o objetivo de verificar a adequação do critério de Teste de Mutação em aplicações móveis Android utilizando o apoio ferramental para linguagem Java (estudo primário 1); e finalmente outro estudo secundário, desta vez com o objetivo de caracterizar estratégias utilizadas para a definição de operadores de mutação em diversas plataformas/linguagens de programação (estudo secundário 2).

Com base nos estudos empíricos realizados, o próximo passo foi a definição do modelo de falhas MOBPI-FM, baseado nas características específicas de aplicações móveis, e gerado sistematicamente utilizando uma técnica de desvios de especificações chamada HAZOP.

Finalmente, foi realizado um estudo de viabilidade com 58 aplicações móveis dos repositórios de código aberto *Github*[2] e *Google Code*[3], com o objetivo de avaliar a cobertura do modelo proposto em relação à representação de falhas em aplicações móveis. O estudo revelou que o modelo MOBPI-FM foi capaz de cobrir falhas reportadas para aplicações móveis reais.

1.6. Estrutura do Documento

Esta dissertação possui a estrutura descrita a seguir.

No Capítulo 2 serão apresentados conceitos importantes para o andamento da dissertação, a saber: o conceito de falhas, por conseguinte o conceito de modelo de falhas. Em seguida, o conceito de aplicações móveis, com um foco em aplicações Android, além dos conceitos de teste, com ênfase no critério de teste baseado em defeitos.

O Capítulo 3 apresenta três estudos preliminares, realizados com o objetivo de coletar evidências que possam embasar a criação de um modelo de falhas específico para aplicações móveis: (1): um mapeamento sistemático sobre técnicas de teste para aplicações móveis que sejam baseadas em código como insumo para a realização de testes, a fim de identificar fontes ou estratégias para revelar eventuais falhas relacionadas ao código em aplicações móveis; (2) um estudo exploratório que relata a aplicação do critério de Teste de mutação (critério de teste baseado em defeito) em aplicações móveis Android utilizando operadores e ferramenta propostos para a linguagem Java a fim de avaliar a adequabilidade de tais operadores quando aplicados em aplicações móveis; e, finalmente, (3) um segundo mapeamento sistemático, dessa vez com o objetivo de caracterizar estratégias de geração de operadores de mutação para diferentes linguagens de programação a fim de identificar

[2] *Github - Where software is built*: https://github.com/;
[3] *Google Code*: https://code.google.com/;

estratégias adotadas em pesquisas científicas para a proposição de operadores de mutação, que estão diretamente relacionadas a identificação de um conjunto de eventuais falhas em uma plataforma ou linguagem de programação e, assim, poderia servir (e serviu) como base para a realização desta pesquisa.

No Capítulo 4 apresenta-se o modelo de falhas MOBPI-FM, iniciando com a definição da metodologia utilizada para a sua a concepção. Em seguida, os níveis em que o modelo pode ser aplicado são apresentados, bem como os itens que compõem cada nível.

Capítulo 5 apresenta um estudo de viabilidade realizado com o objetivo de verificar a cobertura do modelo MOBPI-FM em relação a falhas de aplicações móveis reais reportadas por usuários e desenvolvedores.

O Capítulo 6 apresenta as limitações deste trabalho, assim como conclusões sobre a dissertação e os trabalhos futuros e oportunidades de pesquisa que se apresentam.

CAPÍTULO 2 – REFERENCIAL TEÓRICO E TRABALHOS CORRELATOS

Este capítulo tem como objetivo introduzir os principais conceitos referentes a cada uma das áreas do conhecimento utilizadas na construção desta proposta, identificando também os trabalhos relacionados com essa pesquisa.

2.1. Aplicações Móveis

Uma aplicação móvel é conceituada como uma aplicação desenvolvido para executar em um dispositivo eletrônico móvel (MUCCINI *et al*., 2012). (LECHETA, 2013) complementa tal conceito afirmando que uma aplicação móvel se diferencia de demais aplicações por quatro restrições: recursos limitados, segurança e vulnerabilidade, variabilidade de desempenho e confiabilidade e fonte de energia finita.

Geralmente, tais aplicações encontram-se disponíveis em plataformas de distribuição popularmente conhecidas como lojas de aplicações. Tais lojas são geralmente responsáveis por uma plataforma específica, como por exemplo *Google Play*[4], que distribui aplicações móveis para a plataforma Android, *App Store*[5], que gerencia as aplicações definidas para a plataforma iOS e a *Windows Phone Store*[6], responsável pela plataforma Windows Phone.

Aplicações móveis podem ser classificadas de acordo com a arquitetura adotada na sua construção, podendo assumir as seguintes classificações:

- **Nativas**: tal aplicação é desenvolvida para um sistema operacional móvel específico (JOORABCHI *et al.*, 2013). Uma aplicação nativa é livre para acessar todos os recursos que são disponibilizados pelo fornecedor do sistema operacional. Em muitos casos, esse tipo de aplicação tem características únicas que são funções típicas de um sistema operacional móvel específico. Para criar uma aplicação nativa, os desenvolvedores devem escrever o código-fonte e prover recursos adicionais, tais como imagens, segmentos de áudio, entre outros, de acordo com a especificação do sistema operacional, além de ferramentas e kits de desenvolvimento (SDK's) fornecidos pela plataforma (IBM, 2012);
- **Web**: são aplicações móveis desenvolvidas utilizando tecnologias Web e que são executadas utilizando um navegador móvel(JOORABCHI *et al*., 2013). O grande marco na história das aplicações móveis baseadas em tecnologias web veio com o lançamento do HTML5. O HTML5 incorporou uma série de elementos multimídia

[4] Principal loja de aplicações móveis Android: https://play.google.com/store;
[5] Loja de aplicações móveis Apple: https://store.apple.com/br/shop
[6] Loja de aplicações Windows Phone: https://www.microsoft.com/pt-br/store/apps/windows-phone;

ao HTML comum, o que fez com que ele rapidamente se tornasse uma excelente opção de desenvolvimento. A grande vantagem dessa linguagem é o fato de que ele pode ser lido em qualquer navegador, móvel ou não (IBM, 2012);

- **Híbridas**: são aplicações que permitem ao desenvolvedor criar um núcleo do programa de forma nativa, o que permite que haja um controle sobre as funcionalidades (e pode permitir acesso aos recursos do dispositivo) da aplicação, e dentro da aplicação, uma parte multi-plataforma (JOORABCHI; MESBAH; KRUCHTEN, 2013). Tal abordagem vem ganhando muito destaque nos últimos tempos, pois necessita de um conhecimento básico sobre as plataformas nativas, uma vez que a parte principal da aplicação será desenvolvida utilizando tecnologias web, com o uso do HTML5 (IBM, 2012).

Essa pesquisa é aplicada no contexto de aplicações móveis, independentemente de sua arquitetura, visto que os três modelos possuem características específicas.

2.1.1. Características de aplicações móveis

Pela definição da palavra, uma característica é algo que distingue um ser. No contexto dessa pesquisa, define-se característica de aplicação móvel como fatores que diferenciam uma aplicação móvel de uma aplicação para outras plataformas (MUCCINI *et al*., 2012). Geralmente tal diferença encontra-se na interação de uma aplicação móvel com o dispositivo que ela opera e a plataforma para qual ela foi projetada (MYERS *et al*., 2012).

Foram encontradas várias bibliografias que identificam características de aplicações móveis, tanto em trabalhos acadêmicos (HOLL e ELBERZHAGER, 2014), (MUCCINI *et al*., 2012), (JOORABCHI *et al*., 2013); também em bibliografias sobre Teste de Software (MYERS *et al*., 2012), e finalmente em artigos de empresas de tecnologia (IBM, 2012). Sumarizando os resultados, as principais características de uma aplicação móvel estão na Tabela 1.

Tabela 1. Características específicas de uma aplicação móvel.

Característica	Referente a
Conexão	Métodos de conexão que são fornecidos pelos dispositivos móveis.
Energia	Consumo de energia de um dispositivo móvel.
Configurações do dispositivo	Configurações do dispositivo que influenciam na aplicação.
Interface do dispositivo	Interfaces nativas do dispositivo que se relacionam com a aplicação.
Interface da plataforma	Conceitos de interface específico de determinada plataforma
Interrupções	Poder das aplicações em se adequarem às interrupções do dispositivo
Acesso a dados	Forma como a aplicação manipula dados salvos no dispositivo
Entrada de dados	Múltiplos métodos de entrada fornecidos pela aplicação

Nessa pesquisa, utilizam-se essas características para basear o modelo de falhas MOBPI-FM com o intuito de gerar falhas específicas para aplicações móveis.

2.1.2. Aplicações Móveis em Android

O Android é um sistema operacional projetado para dispositivos móveis baseado no núcleo Linux e atualmente mantido pela empresa de tecnologia *Google* (LECHETA, 2013). Com uma interface de usuário baseada na manipulação direta, o Android é projetado principalmente para dispositivos móveis com tela sensível ao toque, como *smartphones e tablets*, com interfaces específicas para outros dispositivos, como relógios, televisores e até mesmo automóveis (LECHETA, 2013).

Aplicações Android são escritas na linguagem de programação Java. As ferramentas do SDK do Android realizam a compilação do código, juntamente com todos os dados e arquivos de distribuição de recursos, e com isso gera um arquivo com a extensão APK (*Android Package*). Tal arquivo contém todo o conteúdo necessário para a execução de uma aplicação Android em um dispositivo móvel (MEIER, 2008).

Com relação à aplicação propriamente dita, as aplicações Android são baseadas em alguns componentes específicos. Tais componentes são blocos de código essenciais de algumas funcionalidades que compõem uma aplicação Android. Cada um desses componentes é um ponto diferente, por meio do qual o sistema operacional Android realiza suas tarefas (LECHETA, 2013). Nem todos os componentes são os pontos de entrada reais para o usuário e alguns dependem um do outro, mas cada um existe como uma entidade própria e desempenha um papel específico e que ajuda a definir o comportamento geral de uma aplicação (MEIER, 2008).

Existem cinco tipos diferentes de componentes de aplicações móveis Android (GOOGLE ANDROID, 2014). Cada tipo tem uma finalidade distinta e tem um ciclo de vida distinto que define como o componente é instanciado e liberado ao longo da execução do programa. Abaixo, tais componentes são listados:

- ***Activity***: representa uma única tela com uma interface de usuário. Apesar da plataforma Android ter como arquivo de layout arquivos XML, eles podem ser manipulados por meio do componente *Activity* (MEIER, 2008).
- ***Services***: é um componente que é executado em segundo plano com o objetivo de realizar operações de longa execução ou tarefas para processos remotos (MEIER, 2008)
- ***Content Provider***: gerencia um conjunto compartilhado de dados de aplicações. Na plataforma Android, é possível armazenar os dados no sistema de arquivos do

dispositivo, em um banco de dados *SQLite*[7], na web ou qualquer outro local de armazenamento persistente que seja acessível (MEIER, 2008).

- ***Broadcast Receiver***: é um componente que torna determinada aplicação capaz de responder aos eventos que ocorrem no sistema operacional (MEIER, 2008).
- ***Intent***: este componente possui as funções de realizar a comunicação entre o sistema e a aplicação, chamada dos recursos do dispositivo e demais componentes da aplicação (MEIER, 2008).

Nessa pesquisa utilizam-se aplicações móveis Android, mais especificamente as falhas oriundas dessas aplicações móveis para verificar a cobertura do modelo de falhas proposto.

2.2. Teste de Software

A atividade de teste consiste em uma análise dinâmica do produto, sendo relevante para a identificação e eliminação de falhas que persistem em um software. Além disso, essa atividade representa a última revisão da especificação, projeto e codificação do software antes de sua entrega definitiva ao usuário final (DELAMARO *et al*., 2007). Segundo MYERS *et al*. (2012), o objetivo principal do teste de software é revelar falhas que indiquem a presença de falhas no produto.

Apesar de em geral não ser possível provar por meio de testes que um programa está correto, estes contribuem para aumentar a confiança de que o software desempenha as funções especificadas (PRESSMAN e MAXIM, 2014). Além disso, apesar das limitações próprias da atividade de teste, sua aplicação de maneira planejada pode garantir ao software algumas características mínimas, importantes para a qualidade do produto.

Nesse sentido, o teste bem-sucedido é aquele que consegue determinar casos de teste para os quais o programa sendo testado possua falhas. A próxima subseção descreve os conceitos de Erro e Falha, para facilitar o entendimento desta dissertação.

2.2.1. Conceito de Falha e Erro

Segundo o dicionário da língua portuguesa, define-se **falha** como uma condição anormal de que não era prevista. No contexto computacional, compreende-se como uma condição ou comportamento anormal ocorrido em um componente, equipamento, subsistema ou sistema, que pode impedir o seu funcionamento como planejado (WINTER *et al*., 2011).

[7] Gerenciador de banco de dados **autocontido**, **compacto**, com **suporte nativo no Android** e sem necessidade de configuração ou instalação. Fonte: https://sqlite.org/.

Para auxiliar o entendimento desta dissertação de uma maneira geral, é importante entender o significado distinto entre os muitos termos utilizados para definir uma falha: “engano”, “defeito”, o próprio termo “falha” e “erro”.

A IEEE tem realizado vários esforços de padronização, entre eles a padronização da terminologia utilizada no contexto de Engenharia de Software. O padrão IEEE 610.12-1990 (IEEE, 1990) diferencia os termos citados anteriormente:

- **Defeito**: passo, processo ou definição de dados incorreto, como uma instrução ou comando incorreto;
- **Engano**: ação humana que produz um resultado incorreto, como uma ação incorreta tomada pelo programador;
- **Erro**: diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro;
- **Falha**: produção de uma saída incorreta com relação à especificação.

Neste texto, serão utilizados os ternos defeito, erro e falha, na definição das camadas e relacionamentos do modelo (mais detalhes no Capítulo 4). Incluindo tais termos em uma ordem de ocorrência em um programa, a Figura 2 apresenta os termos dispostos em uma sequência crescente de execução.

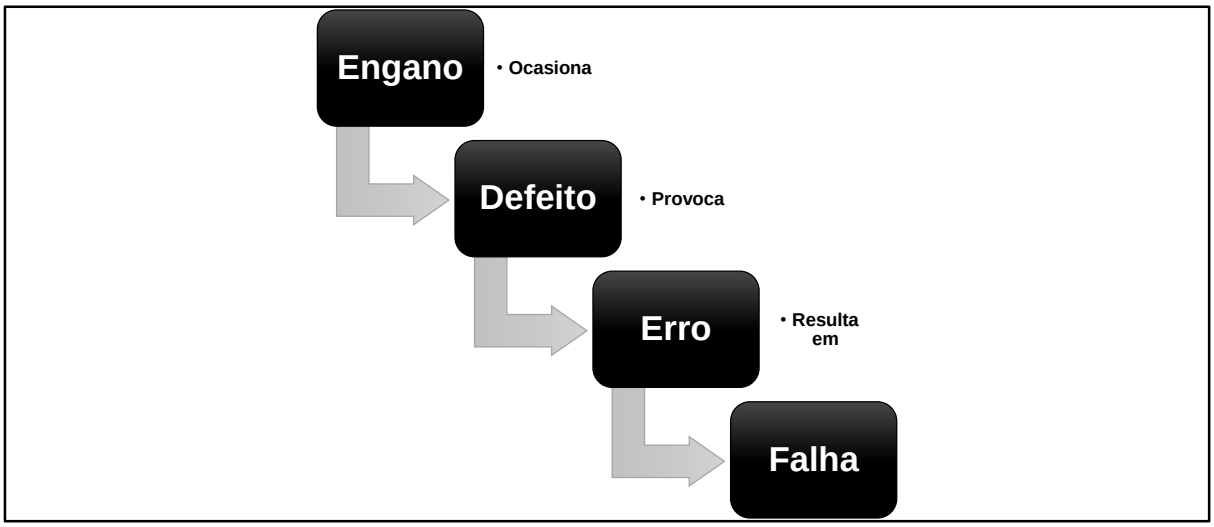


Figura 2. Conceitos ordenados por ordem de ocorrência em um programa.

2.2.2. Teste Baseado em Defeitos

Dentro dessa perspectiva, para que a atividade de teste possa ser conduzida, faz-se necessária a aplicação de técnicas e critérios que indiquem como testar o software, quando parar os testes e que, se possível, forneçam uma medida objetiva do nível de confiança e de

qualidade alcançados com os testes realizados. Em geral, os critérios de teste de software são estabelecidos, basicamente, a partir das técnicas funcional, estrutural e baseada em defeitos (DELAMARO *et al*., 2007).

Na técnica funcional, os requisitos de teste são estabelecidos a partir da especificação do software. Na técnica estrutural, os requisitos são derivados a partir dos aspectos de implementação do software. Na técnica baseada em defeitos, os requisitos de teste são obtidos a partir do conhecimento sobre defeitos típicos cometidos durante o processo de desenvolvimento de software. A ênfase da técnica está nos defeitos que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência. Semeadura de Defeitos e Análise de Mutantes são critérios típicos que se concentram em defeitos. O critério de teste de mutação será detalhado com mais informação nesta seção, pois seus conceitos foram usados ao longo desta pesquisa.

Conceitua-se teste de mutação como um critério da técnica baseada em defeitos que utiliza informações do processo de desenvolvimento de software para derivar requisitos de teste. Este critério será detalhado com mais informação nesta seção pois seus conceitos foram usados ao longo desta pesquisa. A técnica se baseia na alteração do programa sobre teste diversas vezes, criando-se um conjunto de programas alternativos ou mutantes. Logo, um mutante pode ser conceituado como um programa *P'*, que foi gerado a partir de uma alteração sintática em um programa *P*. Assim, como nas outras técnicas de teste, cada programa mutante define um subdomínio do domínio de entrada, formado por aqueles dados capazes de distinguir o comportamento do programa *P* original de programa mutante *P'* (DELAMARO *et al*., 2007). Estes defeitos são inseridos por meio dos operadores de mutação. Entende-se por operador de mutação as regras que definem as alterações que devem ser aplicadas no programa original *P*. Os operadores de mutação são construídos para satisfazer a um entre dois propósitos: (1) induzir mudanças sintáticas simples com base nos defeitos típicos cometidos pelos programadores (como trocar o nome de uma variável); ou (2) forçar determinados objetivos de teste (por exemplo, executar cada caminho possível dentro de um programa) (DELAMARO *et al*., 2007).

2.2.3. Modelo de falhas

BINDER (1999) conceitua o termo modelo de falha como “a identificação de componentes de um sistema que são propensos a ter falhas". Testar todas as possíveis entradas, caminhos ou estados de um sistema é praticamente impossível. O custo computacional seria alto, mesmo para aplicações mais simples. Com isto, encontra-se um desafio de projetar um conjunto de testes que exercite suficientemente um programa para encontrar a maioria das falhas, porém isso ainda não é suficiente para a garantia total de

qualidade. Para auxiliar tal exercício, são necessários subsídios que indiquem locais no programa onde há uma propensão ou uma probabilidade de haver falhas. Um modelo de falhas pode ser considerado tal subsidio, pois procura prover uma indicação de possíveis falhas para um paradigma de programação específico ou uma linguagem específica.

(MAUL *et al*., 2016) complementam tal conceito afirmando que modelos de falhas podem ser utilizados para fornecer informações sobre a capacidade do sistema para satisfazer os requisitos do mesmo. Na fase inicial do projeto do sistema, um modelo de falhas auxilia na verificação dos requisitos e pode resultar em alterações de especificação que seria mais caro de implementar, se deixados para mais tarde no processo de desenvolvimento.

Em relação aos tipos de modelos de falhas, são identificados dois tipos básicos de modelos de falha e as estratégias de teste que podem utilizá-los como insumo (BINDER, 1999):

- **Modelos de falhas para testes de conformidade**: que visa estabelecer a conformidade com os requisitos. Este tipo de teste se baseia em um modelo de culpa não específica: qualquer falha é suficiente para evitar a conformidade. Um modelo de falhas direcionado para tal tipo de teste não necessita de um aprofundamento da definição de suas falhas em nível de implementação.
- **Modelos de falhas para testes direcionado a falhas**: focado em estratégias de teste que procuram revelar falhas de implementação. Este tipo de teste demanda um modelo de falhas específico que possa auxiliar a busca por falhas, pois o número combinações de entrada, saída, estados e caminhos é exponencial ao tamanho da aplicação móvel.

O foco desta dissertação está no segundo tipo de modelo sugerido por (BINDER, 1999) onde o modelo é específico para aplicações móveis, e procura auxiliar na busca por falhas específicas de uma aplicação móvel na utilização dos recursos do dispositivo no qual ela opera e na plataforma que a aplicação foi construída.

Baseado em características específicas, OFFUTT *et al*., (2001) apresentam um modelo de falhas específicas para programas construídos no paradigma orientado a objeto (OO). O modelo define e discute especificamente falhas referentes à herança e polimorfismo. Os autores indicam que o modelo e suas categorias podem ser utilizados para apoiar e inspirar estudos empíricos e técnicas ou abordagens de teste para programas OO, e finalmente para ajudar a aprimorar o projeto e desenvolvimento de programas orientado a objetos. Embora as falhas não fossem amplamente estudadas, há muitos aspectos de falhas que não eram compreendidos na época, principalmente em programas OO. Não apenas para o simples fato de conhecer as falhas, que já é um importante objetivo, mas uma compreensão completa das

características de falhas é fundamental para diversas áreas de pesquisa (OFFUTT *et al*., 2001).

BAEKKEN (2006) apresenta um modelo de falha para dois principais componentes da linguagem Aspect-J: pontos de junção (*pointcuts*) e adendos (*advices*). O modelo de falha fornece uma análise de cada falha em relação aos componentes, e quais destas falhas podem ser inseridas no código, e como essa inserção pode ser propagada para o estado final do programa. O modelo de falha também inclui uma lista de tipos de falhas, juntamente com os defeitos que elas podem gerar, e que consequentemente possam ser introduzidos nos programas Aspect-J. Cada falha é descrita em termos de instanciação sintática no código-fonte, bem como na forma como ele pode alterar estado do programa. O autor acredita que o modelo de falhas pode ajudar testadores e programadores identificar os locais em um programa onde as falhas são mais propensas a aparecer, e identificar que tipos de falhas de podem ocorrer ao lidar com os componentes, e com isso também pode ser utilizado para derivar critérios de adequação de teste para tais programas e elaborar estratégias de teste específicas para programas Aspect-J (BAEKKEN, 2006).

No contexto de aplicações móveis, HOLL e ELBERZHAGER (2014) apresentam uma forma de classificar falhas para aplicações móveis. Por definição, uma forma de classificar falhas pode ser interpretada como um modelo de falhas. Os autores também entendem que o conhecimento de padrões típicos de falhas pode auxiliar a garantia de qualidade de aplicações. (HOLL e ELBERZHAGER, 2014). Com isso, em um trabalho preliminar (HOLL e ELBERZHAGER, 2014) é proposto uma forma de classificar tais falhas, também baseado em características de mobilidade. Em outro trabalho (HOLL *et al*., 2015) avaliam a classificação obtida anteriormente, por meio da leitura baseada em perspectiva, apontando cenários – baseados em papéis do processo de desenvolvimento – onde possíveis falhas de aplicações móveis podem ocorrer.

O trabalho de (HOLL e ELBERZHAGER, 2014) é bastante semelhante ao apresentado nessa dissertação, já que os autores procuram apresentar uma forma de classificar falhas em aplicações móveis, e tal classificação comporta falhas para 6 das 7 características específicas utilizadas no modelo MOBPI-FM, proposto nesta pesquisa. Contudo, as falhas que constam no modelo de (HOLL e ELBERZHAGER, 2014) foram obtidas por meio de pesquisas diretas em repositórios de aplicações móveis reais (HOLL e ELBERZHAGER, 2014). Já o modelo MOBPI-FM, proposto nesta pesquisa, foi construído sistematicamente por meio da aplicação da técnica HAZOP de posse apenas das características específicas. Além disso, no trabalho de (HOLL *et al*., 2015), os autores avaliam a classificação obtida por meio de inspeção formal no intuito de fornecer cenários onde possíveis falhas podem acontecer (verificação). Já o

modelo MOBPI-FM foi avaliado em relação à cobertura de falhas de aplicações móveis reais, e possui como objetivo auxiliar em técnicas e estratégias de teste (validação).

2.2.4. Testes para Aplicações Móveis

Pesquisas recentes em testes para aplicações móveis têm focado na proposição de soluções para problemas técnicos específicos desta plataforma nas seguintes áreas:

Teste Funcional e Comportamental: valida funções de serviços, APIs web móveis, comportamentos de sistemas externos, interfaces do usuário e seus gestos, funções baseadas em localização, perfis de usuários, dados do sistema e dados do usuário. Muitas técnicas de teste funcional (caixa-preta) são úteis em aplicações móveis. Os testes baseados em cenários são bons exemplos. Testes baseados em interface gráfica também tem sido discutido de forma intensa em pesquisas científicas (MUCCINI *et al*., 2012)

Teste Estrutural: visa percorrer os diferentes caminhos possíveis de execução de um código de uma aplicação móvel a fim de atingir uma alta cobertura de código e garantir a qualidade da aplicação sob teste. Métodos existentes de teste estrutural (caixa-branca) para outras plataformas ainda são aplicáveis a aplicações móveis. Os itens que podem ser monitorados durante a execução do programa, aplicando essa técnica de teste, são geralmente voltados para a verificar e validar características de desempenho. Essas características podem ser referentes ao uso de recursos físicos, como processador, bateria e memória, que impactam diretamente no desempenho da aplicação móvel em frente a sua execução e os recursos disponíveis no dispositivo. (MUCCINI *et al*., 2012)

Validação de Requisitos de Qualidade de Serviço (*Quality of Service – QoS*): avalia carga de dados, desempenho, confiabilidade/disponibilidade/privacidade, escalabilidade e vazão de dados do sistema. Pesquisas mencionam a necessidade em monitorar, medir e visualizar diversas métricas de aplicações móveis, como uso de memória, bateria ou CPU (para fins de otimização) e desempenho de redes (em vários tipos de redes de comunicação) (MUCCINI *et al*., 2012)

Teste de Usabilidade: avalia conteúdo e alertas na interface do usuário, fluxos e cenários de operação do usuário, mídias e apoio a interação por gestos. Testes de usabilidade, em laboratório e em campo, ajudam a melhorar a qualidade da experiência do usuário em dispositivos móveis. Este fator tem um grande impacto no sucesso ou fracasso de uma aplicação móvel. Se usuários gostam de uma aplicação, eles baixam, começam a usá-la e avaliam-na positivamente nas lojas. Se não gostam, apagam, partem para outra opção imediatamente e reclamam em redes sociais. Assim, uma versão de baixa qualidade e pouco

atrativa aos usuários pode ter consequências devastadoras para uma aplicação móvel (MUCCINI *et al*., 2012).

Teste de Compatibilidade e Conectividade: avalia a compatibilidade de aplicações a diferentes navegadores web, plataformas e conectividade com redes de comunicação. Hoje em dia, dispositivos móveis suportam várias opções de conectividade, então aplicações móveis devem ser testadas com conectividades e contextos diversificados. Da mesma forma, uma aplicação pode ser desenvolvida para funcionar na maior gama de dispositivos possíveis, o que resulta na necessidade de testá-la em diferentes plataformas. As dificuldades para estes testes seriam a incapacidade dos emuladores em simular o comportamento real dos dispositivos, ferramentas com apoio limitado a recursos importantes para estes testes, tais como mobilidade, serviços de localização, sensores e diferentes tipos de entrada, e ferramentas que permitam migrar testes entre plataformas (MUCCINI *et al*., 2012).

2.3. Técnica de desvios HAZOP

HAZOP (*Hazard and Operability Studies*) é uma técnica estruturada e sistemática que possui o objetivo de identificar e avaliar eventuais problemas que possam representar riscos para pessoas ou equipamentos, ou impedir o funcionamento eficiente de um processo. Sua aplicação é realizada utilizando uma equipe multidisciplinar (chamada de equipe HAZOP) durante uma série de reuniões (MCDERMID e PUMFREY, 1994). A técnica HAZOP é qualitativa, e tem como objetivo, utilizando a interpretação dos participantes, identificar potenciais riscos e problemas de operabilidade; estrutura e integridade são dadas usando palavras-guia.

HAZOP foi projetada originalmente para a indústria química. Hoje, ela é amplamente utilizada para analisar sistemas críticos de segurança em numerosos domínios. Eles oferecem subsídios para uma interpretação imaginativa do comportamento de uma especificação para determinar potenciais riscos que podem surgir, contudo a análise é feita de forma sistemática (MCDERMID e PUMFREY, 1994).

O primeiro passo para aplicação do modelo HAZOP é identificar entidades do sistema (elementos) e os seus atributos examinando uma descrição do sistema a ser avaliado. A descrição do sistema pode estar em um formato lógico ou de projeto de especificação. Na indústria química, por exemplo, a descrição do sistema geralmente se aplica a diagramas de instrumentação. Para um software, pode ser utilizado um diagrama de fluxo de dados ou um diagrama de transição de estado (REDMILL *et al*., 1997).

O passo seguinte é a essência da técnica, que consiste na aplicação de um certo número de palavras-guia predeterminadas, oriunda dos atributos de elementos do sistema

para investigar possíveis desvios, e determinar as possíveis causas e consequências dos desvios. A palavra guia é uma palavra ou frase que expressa e define um tipo específico de desvio. Por exemplo, a Tabela 2 apresenta uma lista de palavras-guia padrão utilizadas com sucesso na indústria química (REDMILL *et al*., 1997).

Tabela 2. Palavras guia da técnica HAZOP utilizadas na indústria química.

Palavra guia	Descrição
NONE	Nenhum fluxo quando deve haver, ou um fluxo reverso.
MORE	Mais de qualquer propriedade física relevante do que deveria haver (e.g. uma pressão mais elevada).
LESS	Menos de qualquer propriedade física relevante do que deveria haver (e.g. temperatura inferior)
PART OF	Composição do sistema diferente do que deveria ser, por exemplo falta de componentes
AS WELL AS	Mais componentes presentes no sistema do que deveria haver (e.g. contaminante num fluido)
OTHER THAN	Outro fluxo que pode acontecer, além de operação normal modo de operação alternativa, ou errado tipo de óleo utilizado para lubrificar uma peça do motor.

A função principal da palavra-guia é atuar como um acelerador. Ele se concentra em um determinado tipo de comportamento anômalo, para assim prover algum tipo de discussão sobre tal comportamento. Cada palavra-guia relevante é aplicada a cada atributo, então uma busca minuciosa por desvios considerada de uma maneira estruturada. Utilizando novamente o exemplo da indústria química, em um tubo de fluido podem haver vários atributos (temperatura, pressão, viscosidade) (MCDERMID e PUMFREY, 1994). Ao considerar a pressão do fluido no tubo para uma aplicação das palavras guia NONE e MORE, podemos cogitar as seguintes perguntas: *"O que acontece se não houver pressão no fluido? "*, ou então *"O que acontece se a pressão do fluido é mais (ou menos) do que o suportado? "*.

A combinação entre palavra-guia e atributo específico deve gerar uma interpretação. Por exemplo, a palavra guia mais aplicada ao valor de dados numéricos "o atributo é melhor interpretado como' maior. Cada palavra guia pode ter mais do que uma interpretação no contexto da sua aplicação na especificação do sistema. Haverá alguma das palavras-guia padrão que não terão uma interpretação significativa. Por outro lado, alguns sistemas podem exigir a adição de outras palavras-guia de acordo com seu padrão de especificação.

No contexto computacional, a técnica HAZOP foi aplicada por KIM *et al*. (1999) na criação do primeiro conjunto de operadores de mutação para a linguagem Java. Os autores utilizaram a técnica de desvios aplicada na especificação sintática da linguagem Java, para

identificar possíveis falhas específicas da linguagem. Com base nessas falhas, 20 operadores de mutação Java foram concebidos, caindo em seis grupos: Tipos e variáveis, Nomes, declaração de Classes/ Interface, Blocos, Expressões e outros (KIM *et al*., 1999).

Outro trabalho no contexto de teste de mutação que utilizou a técnica HAZOP na concepção foi o trabalho realizado por ARAUJO *et al*., (2011), onde os autores focam em modelos de sistemas dinâmicos, especificamente em modelos *Simulink*, que são considerados padrões em muitos domínios de aplicação industrial, como o de aviação civil e de controle automotivo.

A técnica HAZOP foi aplicada para investigar e analisar todas as principais características de tais modelos, a fim de que os operadores mutantes resultantes fossem gerados sistematicamente. Desenvolveu-se um ambiente de teste para apoiar o teste de mutação para modelos de sistemas dinâmicos, que foi usado para empregar os operadores de mutação definidos em uma aplicação de exemplo (ARAUJO *et al*., 2011).

2.4. Considerações sobre o capítulo

Este capítulo procurou abordar os principais conceitos que envolvem o andamento dessa dissertação. Primeiramente, apresenta-se o conceito de aplicações móveis, bem como suas características específicas, e conceitos sobre a plataforma Android.

Em seguida, abordaram-se conceitos referentes a Teste de Software. Quando abordado o conceito de falhas, consequentemente de modelo de falhas, o capítulo também apresentou trabalhos acadêmicos que também propõem modelos de falhas para diferentes plataformas e linguagens.

O trabalho que trata de uma classificação para falhas em aplicações móveis de negócio (HOLL e ELBERZHAGER, 2014) e posteriormente a aplicação em técnicas de inspeção sob três perspectivas (HOLL *et al*., 2015) são os trabalhos que mais se assemelham ao proposto nessa dissertação. Acredita-se que o diferencial do modelo MOBPI-FM em comparação à classificação proposta pelos autores supracitados, apesar de ambos citarem falhas específicas para aplicações móveis, é que as falhas do modelo MOBPI-FM foram concebidas utilizando uma técnica de desvios sistemática, e posteriormente avaliadas em falhas de aplicações móveis reais.

Pelas definições apresentadas, verifica-se que a dissertação pode apoiar a utilização de testes baseados em falhas, provendo um modelo de falhas específico para aplicações móveis, baseado nas características que diferenciam uma aplicação móvel das demais aplicações.

Contudo, faz-se necessário entender como um modelo pode ser aplicado no contexto de aplicações móveis. Na próxima seção, a dissertação apresenta uma série de estudos preliminares que puderam fornecer evidências sobre a necessidade de criação de um modelo de falhas específico para plataformas móveis.

CAPÍTULO 3 – ESTUDOS PRELIMINARES DE APOIO À CONCEPÇÃO DE UM MODELO DE FALHAS PARA APLICAÇÕES MÓVEIS

Este capítulo apresenta resultados de três estudos preliminares realizados com o objetivo de coletar evidências empíricas que indiquem a necessidade e o nível de adequação de um modelo de falhas específico para aplicações móveis.

Três estudos foram realizados inicialmente nesta pesquisa como mecanismo para coletar evidências relacionadas a testes baseados em código de aplicações móveis. O primeiro deles é um estudo secundário conduzido com o objetivo de identificar trabalhos sobre testes baseado em código em aplicações móveis e que revelou a necessidade de um modelo de falhas pela ausência ou carência de trabalhos relacionados ao tema. Em seguida, foi realizado um estudo primário com o objetivo de utilizar a técnica de Teste de Mutação com operadores desenvolvidos para a linguagem Java no contexto de aplicações móveis Android e que revelou a necessidade de operadores de mutação específicos para lidar com características da plataforma móvel. Finalmente, outro estudo secundário foi conduzido com o objetivo de caracterizar estratégias de definição de operadores de mutação a fim de identificar meios para a proposição de um modelo de falhas para aplicações móveis.

3.1. Mapeamento sistemático sobre testes baseados em código em aplicações móveis

De início, foram identificados alguns trabalhos que analisam a literatura técnica em busca do estado da arte no que diz respeito ao Teste de Mutação em geral, porém nenhum trabalho estaria relacionado à plataforma móvel. Então, foi identificada a necessidade de mapear a literatura técnica existente sobre testes para aplicações móveis que utilizem o código fonte como base, como uma forma de ser mais abrangente em relação ao escopo deste mapeamento da literatura técnica.

Portanto, essa subseção descreve trabalhos acadêmicos identificados em tal mapeamento apontando como os mesmos podem trazer contribuições para esta proposta. Um estudo de mapeamento sistemático (SMS – *Systematic Mapping Study*) é planejado e executado como forma de prover uma visão geral da área de pesquisa e determinar uma evidência de pesquisa em tópicos existentes. SMS pode identificar área apropriadas para a condução de uma revisão sistemática da literatura e, também, áreas nas quais um estudo preliminar é mais apropriado (KITCHENHAM e CHARTERS, 2007).

O protocolo do mapeamento foi direcionado para o Apêndice I - Protocolo do Mapeamento Sistemático sobre testes estruturais e baseado em falhas. Os principais trabalhos acadêmicos provenientes do SMS estão descritos de forma resumida nas próximas subseções, agrupados por estratégia de teste.

3.1.1. Testes em componentes específicos de uma plataforma móvel

AVANCINI e CECCATO (2013) apresentam uma abordagem de teste de segurança para aplicações móveis Android que tem como objetivo principal validar a comunicação interna entre aplicações móveis. No trabalho, os autores exploram tanto a geração de casos de teste e a verificação de sua adequação dentro da plataforma Android. Um ponto positivo deste trabalho é o fato de trabalhar com um componente específico da plataforma Android (*Intents*), e também a forma como o teste é executado (inserindo *Intents* corrompidas) e assim validando os testes. O teste em si requer o processo de análise estática de código para identificar a localização e o conteúdo das *Intents*, e análise dinâmica para a execução dos casos de teste gerados. Uma premissa interessante do trabalho, portanto, é a injeção de falhas (*Intents* modificadas) durante o processo de adequação dos testes.

Ainda em testes para o componente *Intent*, SALVA e ZAFIMIHARISOA (2013) propõem uma abordagem de teste de segurança baseada em modelo para tentar detectar vulnerabilidades de dados em aplicações Android. Esta abordagem é capaz de gerar casos de teste para verificar se componentes são vulneráveis a ataques, enviados por meio de *Intents*, que podem trafegar dados de comunicação entre aplicações e realizar chamadas outros serviços e componentes do Android. Essas vulnerabilidades no trabalho são expressas formalmente por meio de autômatos, como uma forma de padronizá-las. Com isso, a abordagem é capaz de criar modelos formais de especificação da aplicação já com as informações a serem tratadas pelas *Intents*. E finalmente, os casos de teste são gerados automaticamente a partir de padrões de vulnerabilidade mapeados e especificações citadas anteriormente. Além de endossar testes para os principais componentes Android, este trabalho contribui com a proposta no sentido de formalizar uma falha, pois uma formalização da mesma contribui para a aplicação de quaisquer critérios de teste.

LU *et al*. (2012) apresentam método chamado *Activity page-based Model*, que possui o objetivo de automatizar testes funcionais utilizando o componente *Activity*. Como dito anteriormente, *Activities* definem o ciclo de vida de uma aplicação Android dentro do contexto da plataforma, e esse ciclo de vida pode ser modelado como um grafo, onde cada nó representa um determinado evento da aplicação. Com isso, o método proposto gera casos de teste de forma automática, utilizando como entrada para os testes informações que façam uma determinada aplicação realizar um determinado evento. A execução dos testes utiliza as

ferramentas *Robotium*[8] e MonkeyRunner[9]. Um ponto positivo do trabalho, além do teste para um componente Android, consiste no uso de dados de teste válidos capazes de explorar todos os possíveis eventos que uma aplicação pode realizar.

No contexto de Teste de Mutação para aplicações móveis, (DENG *et al*., 2015) propõem direcionamentos para a utilização do critério em aplicações móveis Android. Baseado nos conceitos abordados anteriormente, de que testes para plataformas tradicionais necessitam de adaptação para serem aplicados no contexto de aplicações móveis. Os autores definem operadores de mutação específicos para as características de aplicativos Android (*Activities, Services,* entre outros), como o uso extensivo de arquivos XML para mutação em interface gráfica (DENG *et al*., 2015).

Os autores também implementam uma ferramenta para gerar, instalar e executar mutantes em aplicações Android, e relatam resultados preliminares que mostram a viabilidade da aplicação do Teste de Mutação em aplicações Android, e identificam os desafios de pesquisa para o critério tornar-se mais eficaz (DENG et al., 2015). Os autores apontam que um dos desafios de pesquisa é a definição de um modelo de falhas específico para aplicações móveis.

3.1.2. Testes que exploram características do dispositivo

ZHANG e ELBAUM (2014) desenvolveram uma abordagem automatizada para apoiar a detecção de falhas no código de tratamento de exceções, mais especificamente em trechos de código que manipulam recursos externos do dispositivo. Essa abordagem é simples, escalável e eficaz na prática, quando combinado com um conjunto de testes que invocam os recursos do dispositivo que são manipulados, bem como altamente configurável para explorar vários outros aspectos relacionados às características da aplicação.

Como um primeiro passo, a abordagem instrumenta o programa de destino, de modo que os resultados das chamadas para recursos externos de interesse podem ser alterados à vontade para retornar exceções. Em seguida, casos de teste existentes são sistematicamente amplificados, e executados no programa instrumentado para explorar o seu comportamento no trecho da exceção. Quando um teste amplificado revela uma falha, o padrão aplicado erroneamente serve como uma explicação sobre a falha induzida pelo recurso externo. Por fim, esta pesquisa contribui com a proposta sobre a ótica da exploração de falhas causadas e influenciadas por características específicas de um dispositivo.

[8] Ferramenta *Robotium*: http://robotium.com/
[9] Link MonkeyRunner: http://developer.android.com/intl/pt-br/tools/help/MonkeyRunner.html;

Por fim, JIANG *et al*. (2011) propõem um arcabouço de localização estática de falhas específicas de aplicações móveis, especialmente falhas relacionadas com um outro componente Android denominado *Content Provider*. A principal contribuição deste trabalho é utilização de análise estática para localização das falhas associadas ao componente *Content Provider*, que segundo o estudo, consegue prevenir dada aplicação destes determinados tipos de falhas. Ele se encontra nesta subseção porque, segundo experimentos apresentados no trabalho, a grande maioria das falhas estão localizadas em trechos de código que realizam chamadas a funções específicas do dispositivo.

3.2. Estudo exploratório da aplicação de operadores Java no contexto Android

A partir dos resultados do mapeamento sistemático descrito na seção anterior, verificou-se que estratégias de teste são definidas para exercitar trechos de código que gerenciam características específicas de aplicações móveis Android. Com isso, a técnica de teste baseados em defeitos, quando utilizadas no contexto de aplicações móveis Android, deveriam também exercitar tais características.

No intuito de verificar se esse comportamento procede, foi realizado um estudo experimental com o objetivo de verificar a viabilidade do uso do critério de Testes de Mutação (um critério da técnica de teste baseada em defeitos) em plataformas móveis, especificamente na plataforma Android já que, antes de se conhecer o trabalho científico realizado por DENG *et al*. (2015), foram encontrados registros anteriores de tal aplicação. Nesta seção são apresentados os objetivos que foram traçados e que o estudo procurou alcançar:

- Indicar quão apropriado é o critério de Teste de Mutação, quando aplicado em aplicações Android com a utilização de operadores de mutação da linguagem Java. Isto pode ser apresentado por meio do cálculo do escore de mutação, apresentado no Capítulo 2.
- Verificar a relação dos mutantes gerados pelos operadores utilizados em trechos de código referentes aos componentes da plataforma Android.

3.2.1. Planejamento e execução do estudo

Para que fosse possível a aplicação do critério, foram selecionadas uma ferramenta para teste de mutação na linguagem Java e um arcabouço de testes unitários para Android. No decorrer desta seção, será justificada a escolha de cada destes componentes.

Ferramenta de Teste de Mutação. A lista inicial de ferramentas de mutação para Java foi obtida a partir de um experimento encontrado na literatura (DELAHAYE e DU BOUSQUET, 2013), onde os autores fazem uma comparação entre oito ferramentas diferentes em relação

à algumas de teste de unidade, facilidade de uso e entre outros. Nos resultados, a ferramenta PIT[10] apresentou resultados satisfatórios, como a eficiência no Teste de Mutação, facilidade no uso e uma robustez no aspecto de integração com ambientes de desenvolvimento Java.

Com base na robustez da ferramenta, inclusive na integração com o ambiente de desenvolvimento para a plataforma Android, foi decidido utilizar a ferramenta PIT no experimento proposto. A ferramenta pode ser executada a partir da linha de comando, mas também possui integração com ferramentas de integração continua, tais como Maven[11], Ant[12] e Gradle[13]. Ela é altamente configurável, tem padrões de projeto bem definidos e um endereço eletrônico com documentação clara. A ferramenta PIT realiza a mutação no nível de *bytecode* (linguagem intermediária de aplicações Java). DELAHAYE e DU BOUSQUET (2013) citam que, para fins didáticos, a ferramenta se mostrou ideal, por conta da forma como os mutantes são apresentados e também os passos do critério serem de fácil compreensão do usuário. A Figura 3 ilustra o uso da ferramenta PIT para gerar mutantes e mostrar a cobertura de código, e a Figura 4 mostra que foram gerados oito mutantes gerados para a linha 7 do código-fonte da Figura 3.

```
1    package com.pivotallabs.util;
2
3    import java.io.*;
4
5    public class Strings {
6        public static boolean isEmptyOrWhitespace(String s) {
7  8         return (s == null) || (s.trim().length() == 0);
8        }
9
10       public static String fromStream(InputStream inputStream) throws IOException {
11 1         char[] buffer = new char[0x10000];
12          StringBuilder out = new StringBuilder();
13          Reader in = new InputStreamReader(inputStream, "UTF-8");
14          int read;
15 5         while ((read = in.read(buffer, 0, buffer.length)) > 0) {
16 1            out.append(buffer, 0, read);
17          }
18          return out.toString();
19       }
20
21       public static InputStream asStream(String string) {
22          return new ByteArrayInputStream(string.getBytes());
23       }
24   }
```

Figura 3. Exemplo de classe modificada pela ferramenta PIT.

[10] *PIT Mutation Testing*: http://pitest.org/;
[11] *Apache Maven Project*: https://maven.apache.org/;
[12] *The Apache Ant Project*: http://ant.apache.org/;
[13] *Gradle | Modern Open-Source Enterprise Build Automation*: http://gradle.org/.

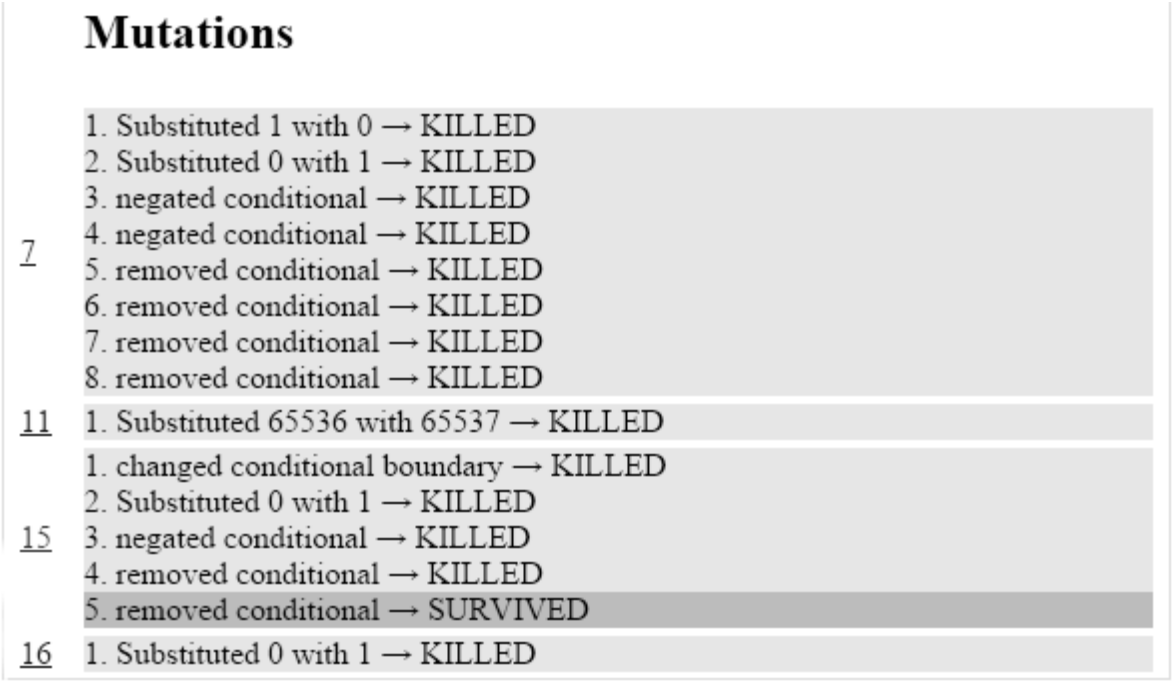


Figura 4. Exemplos dos mutantes gerados para a linha 7 da Figura 4.

Porém, ainda segundo DELAHAYE e DU BOUSQUET (2013), para fins de execução de pesquisa empíricas de laboratório, outras ferramentas de mutação eram melhor adequadas por conta do número de operadores que possuem, tais como *MuJava* (MA *et al*., 2006), Major (JUST, 2014) e *Jumble*[14], e tal afirmação é endossada em outras pesquisas científicas (SINGH e SURI, 2013). Contudo, nenhuma destas ferramentas se mostrou integrável com a plataforma Android como a ferramenta PIT, mesmo com o pequeno número de operadores de mutação utilizados na ferramenta. A Tabela 3 apresenta os operadores que são disponibilizados pela ferramenta PIT.

Com relação aos mutantes gerados pela ferramenta, após sua execução, eles podem ser classificados da seguinte maneira:

- **Morto** (*Killed*): quando pelo menos um caso de teste identificou em um mutante uma saída diferente do programa original;
- **Vivo** (*Survived*): quando nenhum caso de teste no conjunto é capaz de distinguir a execução do mutante do programa inicial, ou seja, para todos os casos de teste as saídas são o mesmo que o programa original.
- **Não coberto** (*Not Coverage*): Um mutante foi gerado em trechos de código que ainda não foram executados por qualquer caso de teste;

[14] *Jumble Mutation Testing:* http://jumble.sourceforge.net/index.html;

Tabela 3. Operadores de mutação Java disponíveis na ferramenta PIT.

ID	Operador
PIT1	*Conditionals Boundary Mutator*
PIT2	*Negate Conditionals Mutator*
PIT3	*Remove Conditionals Mutator*
PIT4	*Math Mutator*
PIT5	*Increments Mutator*
PIT6	*Invert Negatives Mutator*
PIT7	*Inline Constant Mutator*
PIT8	*Return Values Mutator*
PIT9	*Void Method Calls Mutator*
PIT10	*Non Void Method Calls Mutator*
PIT11	*Constructor Calls Mutator*

Algumas alterações sintáticas providas pelos operadores podem ocasionar falhas sintáticas. Neste caso, a ferramenta classifica o mutante gerado com o seguinte estado, de acordo com o tipo de falha:

- **Limite de tempo** *(Time out)*: Uma mutação pode expirar se causar um laço infinito, como remover o incremento de um contador em um laço "for".
- **Não viável** (*Non-Viable*): A mutação não viável é aquela que não pode ser carregada pela *JVM*[15], como o *bytecode* foi de alguma forma inválida. PIT tenta minimizar o número de mutações não viáveis que ele cria.
- **Falha de memória** (*Memory-error*): uma falha de memória pode ocorrer devido a uma mutação que demande ao programa uma quantidade maior de memória utilizada;
- **Falha de execução** (*Run Error*): uma falha de execução significa que algo anormal ocorreu ao tentar testar a mutação. Algum tipo de mutação não viável atualmente pode resultar em um falha de execução.

Arcabouço de Teste Unitário. Com relação ao arcabouço de teste, optou-se por usar a ferramenta Robolectric[16]. Tal arcabouço é capaz de executar testes na JVM. Apesar de ser baseado no JUnit, o mesmo funciona de forma diferente quando comparado com JUnit para Android que necessita de um emulador ou dispositivo para executar os testes. Robolectric consegue isso interceptando as chamadas as funções do SO e direcionando tais chamadas

[15] Máquina virtual Java (***J**ava **V**irtual **M**achine*) é um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* em código executável de máquina;
[16] *Robolectric Unit Testing for Android*: http://robolectric.org/;

para objetos implementados no arcabouço denominados *Shadow Objects*. Assim, é possível executar testes dentro de um JVM normalmente, o que proporciona um ganho de tempo no processo de execução dos testes.

Aplicações móveis. As duas aplicações utilizadas no estudo são de código aberto e estão disponíveis em repositórios na Web. São elas: *Robolectric Sample*[17] e *Bluetooth Chat*[18]. Ambas são aplicações nativas da plataforma Android e fornecem todos os artefatos relacionados ao projeto, incluindo o código fonte e um conjunto de casos de teste projetados por seus desenvolvedores/testadores, visto que ambos são requisitos para executar testes de mutação por meio da ferramenta PIT. A Figura 5 mostra capturas de tela das aplicações móveis utilizadas no estudo.

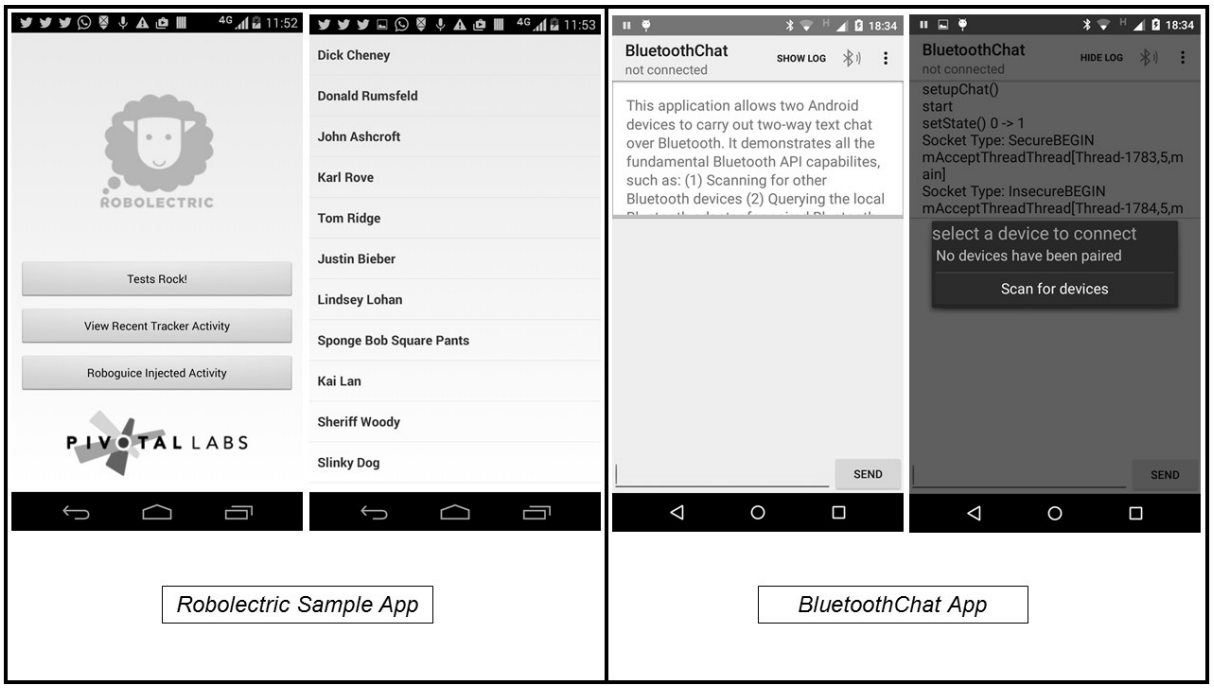


Figura 5. Capturas de tela das aplicações móveis utilizadas.

Com relação à execução do estudo, o ambiente em que o mesmo foi executado tem a seguinte configuração:

- **Sistema Operacional**: Windows 8.1 *Single Language*;
- **Processador**: Intel I7;
- **Memória**: 4 GB;
- **Versão JDK**: JDK 8;
- **Versão da PIT**: 1.1.3;

[17] Projeto de código aberto que visa mostrar a configuração da ferramenta de teste *Robolectric*. Disponível em <https://github.com/robolectric/robolectric-samples>.
[18] Projeto de demonstração da funcionalidade de Bluetooth em aplicações Android: <http://developer.android.com/samples/BluetoothChat/index.html>

- **Android *SDK***: 2.3.6 (API 10)
- **Versão *Robolectric***: 2.3.
- **Versão do Maven**: 3.2.1.

Para a execução desse trabalho, foi usada a configuração indicada pela documentação da ferramenta PIT por meio da ferramenta de gerenciamento de dependências Maven. O procedimento de teste está disponível na documentação da PIT. Para fins de estudo, primeiramente o processo foi executado utilizando todos os operadores em conjunto, e também separadamente para cada operador de mutação disponível na ferramenta. Os resultados são apresentados na seção seguinte.

3.2.2. Resultados obtidos

Nesta seção, será apresentada a análise dos resultados obtidos no estudo, com foco em atingir os objetivos propostos na seção anterior. Usando os 11 operadores disponíveis na ferramenta PIT para cada aplicação móvel, os resultados são resumidos na Tabela 4.

Tabela 4. Resultados obtidos na execução do estudo.

Métricas	***Robolectric Sample***	***BluetoothChat***
Mutantes Gerados	614	635
Mutantes Mortos	595	610
Mutantes mortos por *TIMEOUT*	522 (88%)	256 (42%)
Mutantes não mortos por *TIMEOUT*	73 (12%)	354 (58%)
Mutantes equivalentes	19	25
Escore de mutação (%)	100%	100%
Cobertura de Código (%)	98%	99%

Como primeira análise, foram avaliados individualmente o comportamento de cada operador de mutação Java no contexto de aplicações móveis Android. Os resultados para cada uma das aplicações móveis são discutidos a seguir.

Para a aplicação *Robolectric Sample* (Figura 6), pode ser observado que o operador de mutação PIT10 (*Non Void Method Calls Mutator*) gerou o maior número de mutantes (168). Por outro lado, o operador PIT06 (*Invert Negatives Mutator*) não gerou nenhum mutante, o que se explica pela falta da estrutura sintática requerida pelo operador na aplicação. As características e funcionalidades da aplicação podem explicar tal cenário, pois a principal função da aplicação se resume na transição de telas, fazendo assim muito uso da classe *Activity*, que tem seu ciclo de vida baseado em chamadas de eventos por meio de funções.

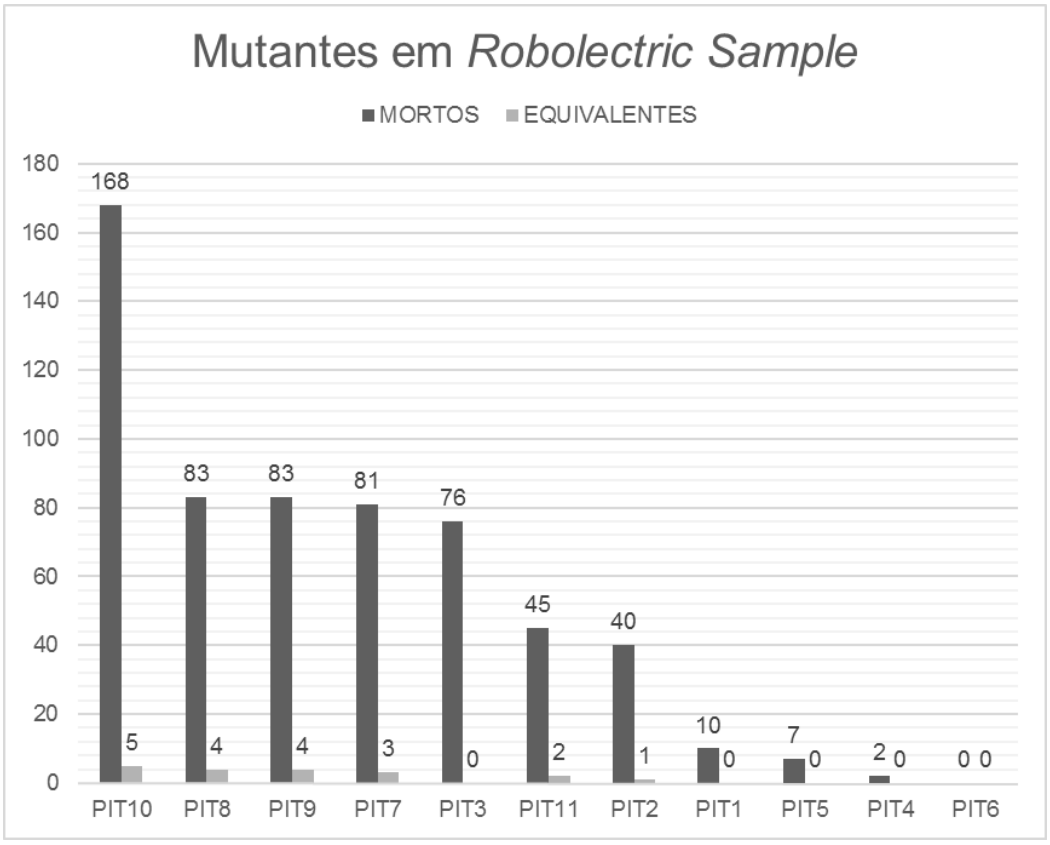


Figura 6. Resultado da análise dos mutantes para a aplicação *Robolectric Sample.*

Na aplicação *BluetoothChat* (Figura 7), foi possível observar que o operador PIT11 (*Constructor Calls Mutator*) gerou o maior número de mutantes (149). Por outro lado, mais uma vez os operadores PIT6 (*Invert Negatives Mutator*) e o operador PIT5 (*Increments Mutator*) não geraram nenhum mutante. Podemos ver que o PIT11 operador ainda gera mais mutantes do que os outros, embora a aplicação *BluetoothChat* tenha mais recursos do que a primeira aplicação, o que pode ser explicado pelo aumento da utilização de métodos na implementação da aplicação. Além disso, o operador aritmético (PIT6) não obteve sucesso na criação de mutante, o que seria explicado pelo pouco uso de operações aritméticas, e este padrão pode ser observado na aplicação *RobolectricSample* também.

Assim, foi possível executar o processo de teste de mutação com sucesso em ambas as aplicações, o que soluciona o primeiro objetivo deste estudo. Tal afirmação é apoiada pelos valores obtidos nos escores das aplicações utilizadas no estudo. Uma questão interessante observado nesta análise foi o alto número de mutantes mortos por tempo de espera em ambas as aplicações móveis. A seguir, a Figura 8 e a Figura 9 mostram a distribuição dos mutantes mortos e não mortos por *timeout* para cada operador nas aplicações do estudo.

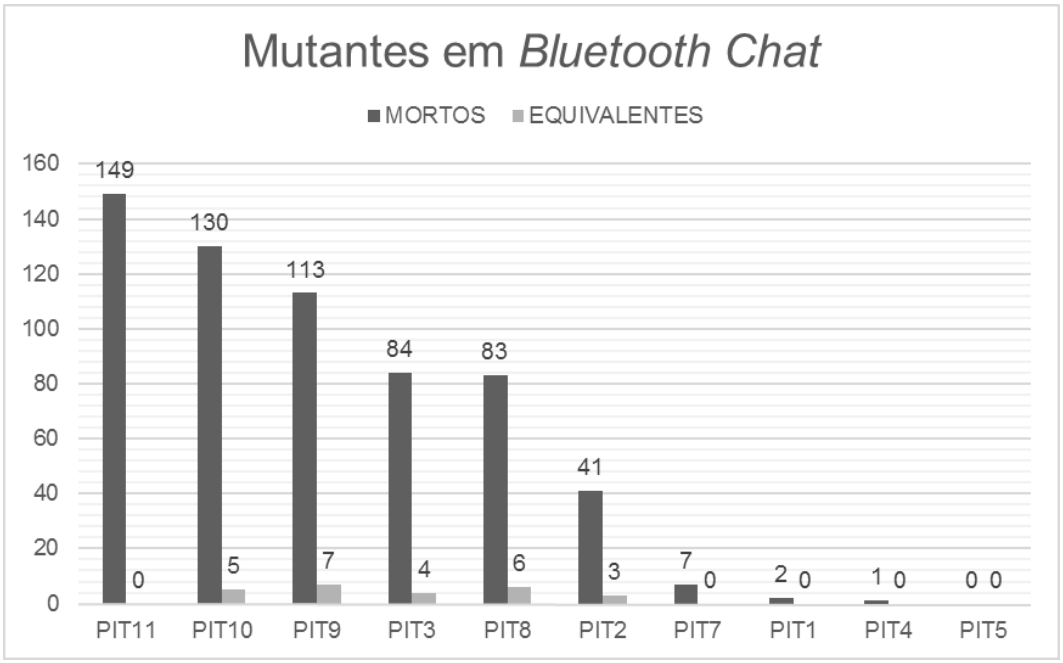


Figura 7. Resultado da análise dos mutantes na aplicação *BluetoothChat*.

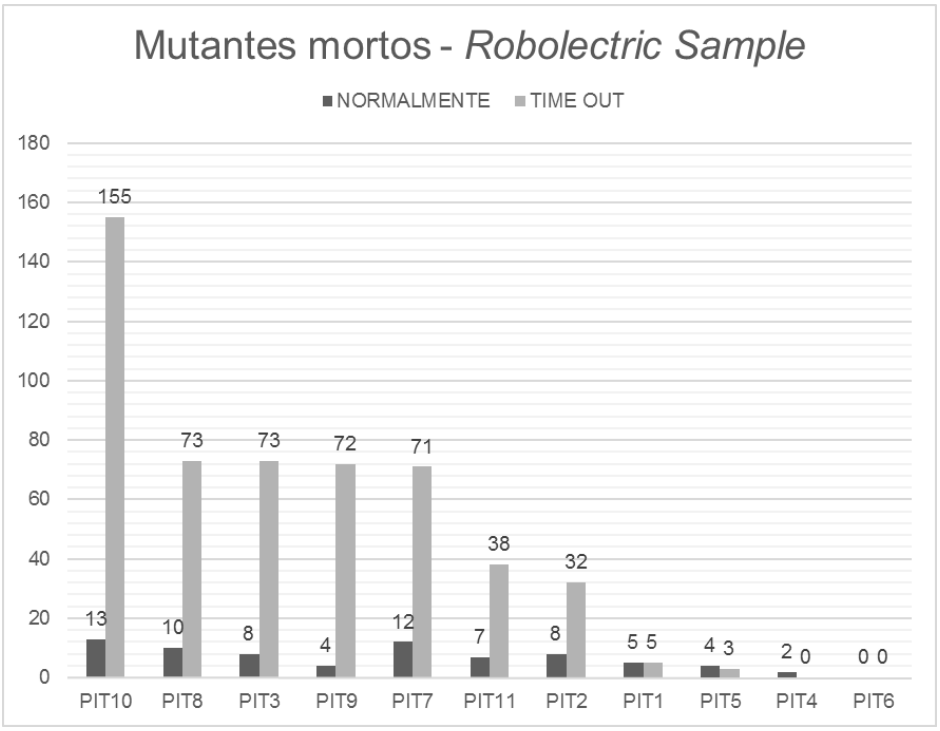


Figura 8. Mutantes mortos por *timeout* na aplicação *RobolectricSample*.

Os operadores que modificam as chamadas de método (PIT10 e PIT9) e retornos de métodos (PIT8) são os que geram mais mutantes mortos por *timeout*. Os operadores que geram, proporcionalmente, menos mutantes mortos por *timeout* são aqueles que substituem operadores relacionais (PIT1) e operadores aritméticos (PIT 5 e PIT4). Nestes operadores, o número de mutantes mortos por *timeout* foi menor ou igual do que os mutantes não mortos por *timeout*.

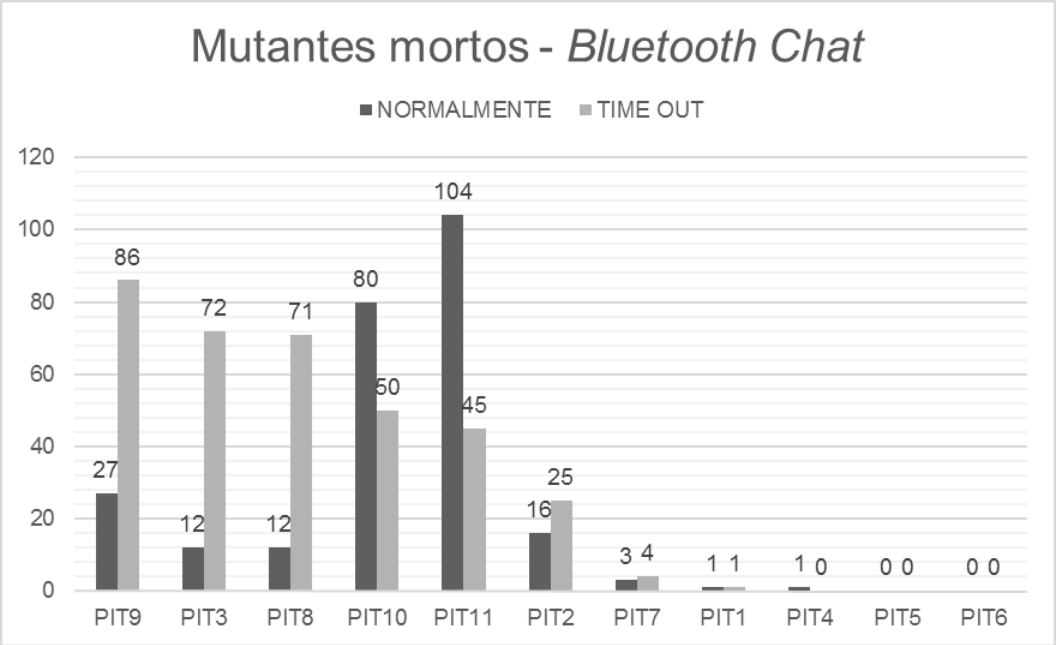


Figura 9. Mutantes mortos por *timeout* na aplicação *BluetoothChat*.

O elevado número de mutantes mortos por *timeout* ocorre porque os operadores de mutação alteram partes do código da aplicação móvel que influenciam no fluxo de execução em aplicações Android, criando laços infinitos. Exemplos dessas mudanças são: eliminação de funções de retorno (transformando-os em procedimentos), a exclusão de chamadas em classes essenciais de construtores por exemplo, *new Activity()*, ou a substituição de objetos por outro do mesmo tipo, porém com um conteúdo nulo.

Assim, entendemos os operadores de mutação tradicionais Java podem ser aplicados para gerar mutantes em aplicações móveis Android. No entanto, eles têm de ser adaptados para lidar com características específicas presentes em códigos Android.

Portanto, este estudo considera estes mutantes como mortos, porque os testes são capazes de identificar saídas inesperadas. Podemos observar que para o contexto Android, este operador de mutação pode ser melhorado, a fim de gerar mutantes mais relevantes, ou seja, os casos de teste conseguiram identificar o ponto exato de mutação. Contudo, são necessários mais estudos que possam verificar se tal problema é universal para qualquer componente *Activity*, por exemplo. Mas como uma conclusão inicial, podemos assumir ou que os métodos da classe *Activity* devem ser excluídos do processo de Teste de Mutação, ou os operadores citados acima que mais geraram esse tipo de mutação que necessitam ser adaptados para o contexto de aplicações Android.

Um outro ponto a ser avaliado é a influência dos operadores tradicionais componentes específicos da plataforma Android. O resultado dessa avaliação pode ser visualizado na Figura 10.

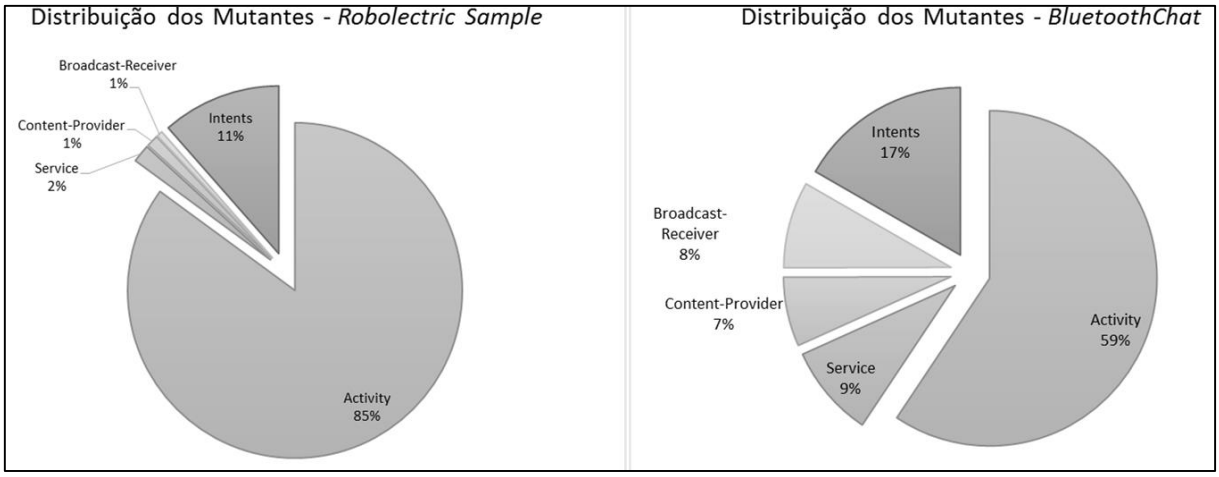


Figura 10. Distribuição dos mutantes em componentes Android.

Verifica-se que o componente *Activity* é o que mais recebe mutação, muito por conta das funcionalidades das aplicações móveis escolhidas para o estudo, principalmente na aplicação *Robolectric Sample*. Na aplicação *BluetoothChat*, o número de mutante se equilibra mais devido a maior utilização dos outros componentes.

Complementar a esta análise, a Tabela 5 apresenta a densidade de mutantes por componente Android. Esta medida foi concebida pela razão entre o número de mutantes por linhas de código (correspondente a cada componente).

Tabela 5. Densidade de mutantes em cada componente Android.

Componentes	Número de Mutantes	LOC	Densidade de Mutantes
Activity	468	397	1.18
Service	51	80	0.64
Intent	38	110	0.35
Content Provider	46	70	0.66
Broadcast Receiver	110	46	2.39

3.2.3. Ameaças a Validade do Estudo

Validade interna. A validade interna está relacionada com o controle do processo de experimentação para recolher dados analisados neste estudo. Utilizou-se a ferramenta PIT para a geração de mutantes. Obviamente, a baixa quantidade de operadores de mutação implementado pela ferramenta com respeito aos descritos em (AHMED *et al.*, 2010) (apresentados na Seção 3.2.1) não permite resultados mais conclusivos.

Validade externa. Refere-se ao risco de generalizar os resultados obtidos a partir do experimento apresentado. Foram identificadas as seguintes ameaças:

- Neste estudo, foram utilizadas apenas duas aplicações móveis (apresentadas na seção 3.2.1) em um total de 925 linhas de código, que é um número muito baixo de aplicações para assumir resultados mais conclusivos.
- Outra limitação refere-se à relevância das aplicações móveis utilizadas no estudo. Como são aplicações móveis com um propósito didático, não é possível generalizar os resultados para situações reais ou industriais, por exemplo.
- Outra limitação refere-se aos programas escolhidos devido ao fato de que, para alguns operadores, nenhum mutante foi gerado (PIT6 em *Robolectric Sample*, PIT5 e PIT6 em *BluetoothChat*). Isto ocorre quando o programa não contém a estrutura sintática necessária pelo operador para gerar mutantes.
- Por fim, o conjunto de teste utilizado no experimento foi provido juntamente com as aplicações, o que poderia representar uma limitação na investigação de critério de Teste de Mutação. No entanto, eles foram projetados por desenvolvedores das aplicações móveis em questão.

Validade de Construção. O risco de construção deste trabalho reside no processo utilizado no experimento. Para isso, o experimento foi conduzido pelos pesquisadores, que têm experiência no assunto investigado.

3.3. Mapeamento sistemático sobre definição de operadores de mutação

Com base no estudo exploratório apresentado anteriormente, verificou-se que o critério de teste de mutação necessita de adaptações para a plataforma Android, a saber: operadores específicos e um apoio ferramental adequado. Tal conclusão também é apresentada por DENG *et al*. (2015). Pelo fato do critério de teste de mutação utilizar falhas pré-definidas para gerar programas mutantes, verificou-se a necessidade de obter um conhecimento de como tais operadores forma definidos para outras linguagens.

Com isso, essa subseção apresenta um mapeamento sistemático (KITCHENHAM e CHARTERS, 2007) que teve como objetivo compreender como operadores de mutação são concebidos, e também quais os principais insumos utilizados na construção de tais operadores. Para isso, esta seção apresenta inicialmente o protocolo do mapeamento, em que são apresentados os objetivos e insumos iniciais do mapeamento. Em seguida, apresentamos a condução do mapeamento, juntamente com os dados objetivos. Finalmente, os resultados são analisados e alguns dados são apresentados.

3.3.1. Questões de pesquisa

Para alcançar os objetivos deste mapeamento sistemático foram propostas as seguintes questões de pesquisa:

- (QP1): Quais são os principais elementos ou fontes de dados utilizados pelos autores na definição dos operadores de mutação?
- (QP2): Quais são as principais abordagens que auxiliam a definição dos operadores de mutação?
- (QP3): Quais são as estratégias para a geração dos operadores de mutação?

3.3.2. Identificação e Seleção de Estudos Primários

Foram usadas como fonte de pesquisa três bibliotecas digitais online: *Scopus*[19], *IEEEXplore*[20] e *ScienceDirect*[21]. Para realizar tal busca, utilizou-se a seguinte *string*:

("mutation testing" OR "mutation analysis" OR "mutant testing" OR "fault injection" OR "fault seeding" OR "fault-based testing") ***AND*** *("operators" OR "operator" OR "mutagens" OR "mutagenic" OR "rules")*

Os seguintes critérios de inclusão forma adotados na seleção dos estudos:

- O artigo escrito em Inglês;
- Publicações disponíveis na Internet ou obtidos contatando com seus autores;
- Publicações devem estar relacionados ao projeto de operadores de mutação.
- Artigos que não cumprissem todos os critérios de inclusão foram excluídos.

3.3.3. Formulário de extração de dados

De cada trabalho selecionado, extraímos os seguintes dados dispostos na Tabela 6.

Tabela 6. Campos de extração para os trabalhos.

Campo de extração	Descrição
Título	O título do trabalho
Autor (es)	O (s) autor (es) do trabalho
Fonte	Periódicos/conferência de publicação do trabalho
Ano de publicação	O ano de publicação do trabalho
Tipo de estudo	O tipo de estudo realizado pelo trabalho

19 Scopus: http://www.scopus.com;

20 IEEExplore Digital Library: http://ieeexplore.ieee.org/Xplore/home.jsp;

21 Science Direct: http://www.sciencedirect.com;

Tipo de plataforma	Tipo de plataforma descrita no trabalho
Nome plataforma	O nome da plataforma
Linguagem	A linguagem de programação que sofre a mutação
Fonte de Dados	A fonte de dados que baseia os operadores
Abordagem de geração	Abordagem utilizada na definição dos operadores.

3.3.4. Execução do Estudo de Mapeamento Sistemático

Todas as atividades do SMS foram realizadas entre junho e dezembro de 2015. Foram encontrados 878 documentos nas bibliotecas digitais definidos no protocolo SMS. Depois de aplicar o filtro 1 (seleção com base no título, palavras-chave e resumo), foram selecionados 140 estudos. No passo seguinte, foi aplicado o filtro 2 (análise completa do estudo), 75 estudos publicados entre 1998 e 2015 foram selecionados para a extração de informação. Finalmente, no terceiro filtro, onde foram selecionados apenas trabalhos que relatam a definição de operadores de mutação que tem como base linguagens de programação. Os resultados descritos são sumarizados na Tabela 7.

Tabela 7. Sumarização dos resultados da tabela.

Biblioteca digital	**Artigos Selecionados**	**Filtro 1**	**Filtro 2**	**Filtro 3**
IEEEXplore	170	56	23	16
Scopus	500	78	43	24
ScienceDirect	208	6	2	0
TOTAL	**878**	**140**	**68**	**40**

Com base nos filtros, a Tabela 8 apresenta os trabalhos acadêmicos reportados de forma sumarizada.

3.3.5. Análise dos dados

Nesta seção é apresentada a análise dos resultados obtidos no SMS, com foco na realização dos objetivos e responder às questões de investigação propostos na seção anterior. A Figura 11 mostra a distribuição de artigos por ano de publicação. Pode ser observada uma série regular de publicações que relatam novos operadores de mutação desde 1998. Em 2014 foi observado um aumento no número de publicações.

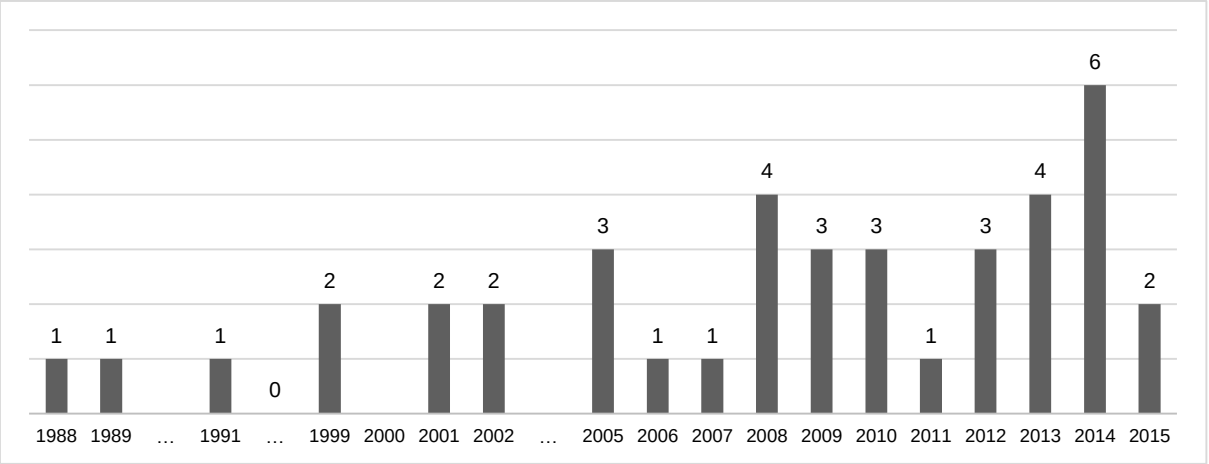


Figura 11. Artigos distribuídos por ano de publicação.

Tabela 8. Trabalhos encontrados no mapeamento sistemático.

ID	Conferência /Jornal	Ano	Linguagem de Programação	Fonte de Dados	Abordagem
P01	ICST	2014	C/C++/C#	Aprimoramento	Não sistemático
P02	WICT	2014	Java	Características específicas	Não sistemático
P03	ICSTW	2015	Android	Características específicas	Não sistemático
P04	ICSM	2005	Java	Aprimoramento	Não sistemático
P05	ICST	2008	Aspect-J	Falhas comuns	Não sistemático
P06	ISSRE	2002	Java	Características específicas	Não sistemático
P07	ICSTW	2010	Java	Falhas comuns	Não sistemático
P08	ICSTW	2015	Ruby	Aprimoramento	Não sistemático
P09	CONFLUENCE	2014	Aspect-J	Falhas comuns	Não sistemático
P10	ANSS	2008	OWL-S	Falhas comuns	Não sistemático
P11	QSIC	2005	SQL	Falhas comuns	Não sistemático
P12	LATW	2012	C/C++/C#	Características específicas	Sistemático
P13	ICST	2013	JavaScript	Aprimoramento	Não sistemático
P14	APSEC	1999	Java	Características específicas	Não sistemático
P15	ISSRE	2002	Java	Características específicas	Não sistemático
P16	ICWS	2009	OWL-S	Falhas comuns	Não sistemático
P17	ICST	2012	Java	Falhas comuns	Não sistemático
P18	Telecom Journal	2014	C/C++/C#	Características específicas	Não sistemático
P19	WICT	2014	Java	Características específicas	Não sistemático
P20	Springer	2014	Jason	Falhas comuns	Não sistemático
P21	APM	2013	Não especificado	Características específicas	Não sistemático
P22	SCP	2013	Java	Falhas comuns	Não sistemático
P23	IFAAMS	2013	AgentSpeak	Falhas comuns	Systematic
P24	ICCS	2012	Não especificado	Características específicas	Não sistemático
P25	-	2011	Java	Falhas comuns	Não sistemático
P26	ICSTW	2010	Java	Falhas comuns	Não sistemático
P27	IJSEKE	2010	C/C++/C#	Características específicas	Não sistemático
P28	COMPSAC	2009	Java	Características específicas	Não sistemático
P29	LATW	2009	WSDL	Características específicas	Não sistemático
P30	HASE	2008	C/C++/C#	Características específicas	Não sistemático

P31	ISSEKE	2008	Não especificado	Características específicas	Não sistemático
P32	MUTATION	2006	Java	Características específicas	Não sistemático
P33	ISSRE	2001	XML	Falhas comuns	Não sistemático
P34	TSE	2001	C/C++/C#	Falhas comuns	Não sistemático
P35	IST	2007	SQL	Falhas comuns	Não sistemático
P36	-	1989	C/C++/C#	Falhas comuns	Não sistemático
P37	SPE	1991	Fortran	Falhas comuns	Não sistemático
P38	-	1988	Ada	Falhas comuns	Não sistemático
P39	Springer	2005	C/C++/C#	Características específicas	Não sistemático
P40	ICESSA	1999	Java	Características específicas	Não sistemático

Analisando a linguagem para qual os operadores de mutação foram projetados, a Figura 12 mostra a distribuição dos trabalhos. Como esperado, a maioria dos artigos está relacionada às linguagens C/C++/C# e Java, que são as linguagens de programação de software mais robustas e populares.

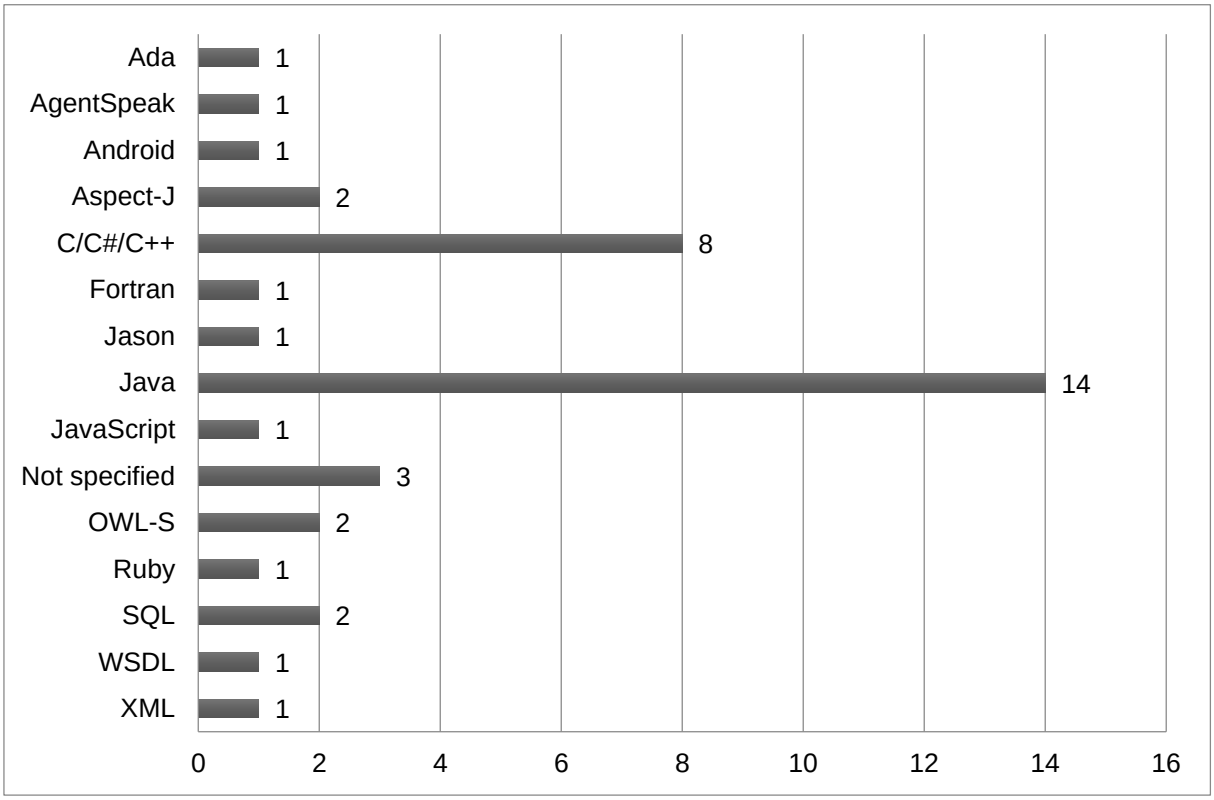


Figura 12. Artigos distribuídos por linguagens de programação.

3.3.6. QP1: As fontes utilizadas para a definição dos operadores

Com o objetivo de responder à primeira questão da pesquisa (QP1), foi possível identificar artigos que relatam as seguintes fontes principais para a definição do operador são descritos a seguir:

- **Falhas comuns:** A mais instintiva de todas, pois alinha-se profundamente com a hipótese de um programador competente descrita anteriormente. Refere-se a falhas comuns oriundas de defeitos de sintaxe da linguagem de programação.
- **Características específicas**: Quando o operador é proposto a partir de características específicas da linguagem ou plataforma, onde tais características influenciam diretamente o desenvolvimento do software da linguagem ou da plataforma.
- **Aprimoramento**: Quando os pesquisadores identificam, por meio de estudos empíricos, lacunas nos operadores previamente definidos. Com base nos dados obtidos, novos operadores são designados.

A Figura 13 mostra a distribuição de artigos por fontes utilizadas para projetar operadores. Podemos observar duas fontes (Falhas comuns e características específicas) com o mesmo número de artigos (18). Apenas quatro trabalhos tentaram evoluir operadores a partir de um conjunto anterior (Aprimoramento).

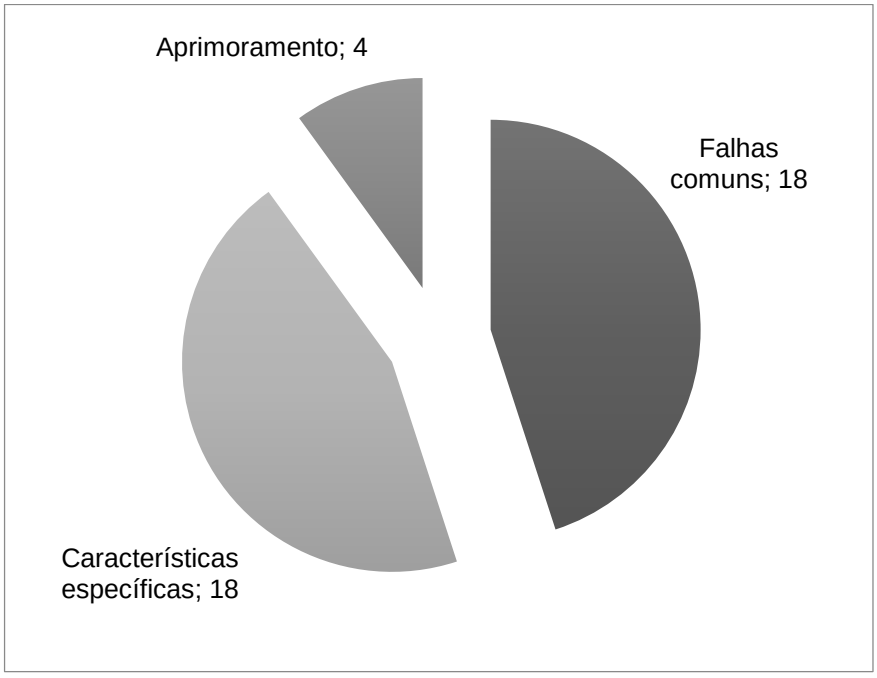


Figura 13. Artigos de acordo com as fontes de dados de mutação.

3.3.7. QP2: abordagens utilizadas para projetar operadores de mutação

Com o objetivo de responder à segunda questão de pesquisa (QP2), foi possível identificar nos artigos o relato de abordagens para a definição de operador de mutação. As abordagens encontradas são as seguintes:

- **Sistemática**: Quando os pesquisadores usam uma metodologia sistemática que auxilia na geração de falhas (exemplo [P40] utilizou a técnica HAZOP);

- **Não sistemática**: A definição de falhas é feita de acordo com as modificações não-sistemáticas das estruturas sintáticas da linguagem ou na modificação da estrutura do modelo. Normalmente, ele é baseado na experiência de pesquisadores ao propor em novos operadores.

A Figura 14 mostra a distribuição de artigos por abordagens utilizadas para projetar operadores. Podemos observar que a maioria dos trabalhos (37) utilizou uma abordagem não sistemática. Apenas três artigos apontam metodologias sistemáticas de propor novos operadores de mutação.

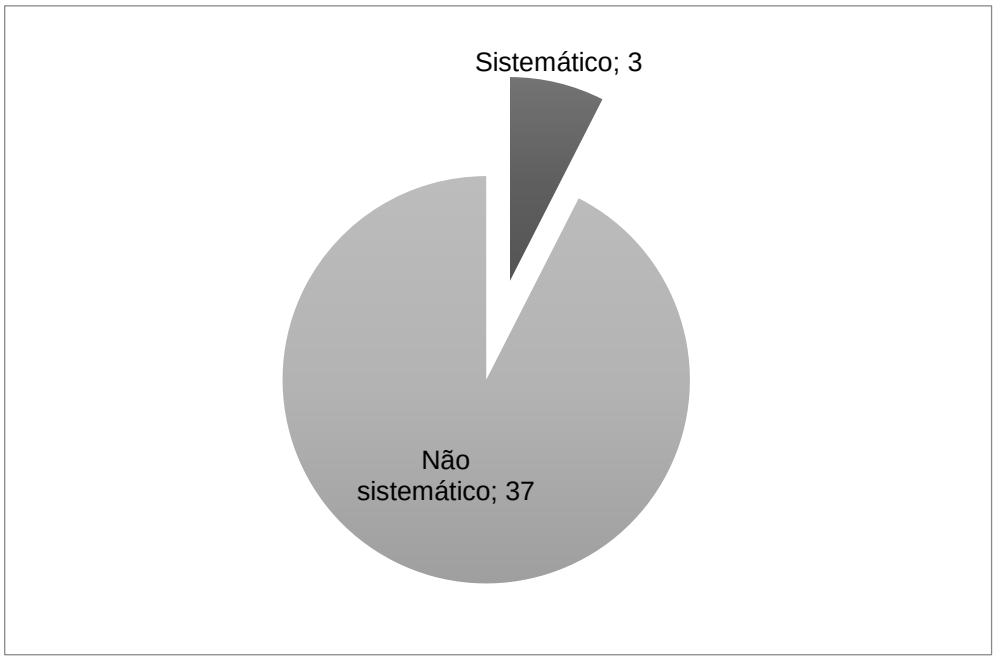


Figura 14. Artigos distribuídos por abordagens de mutação.

3.3.8. QP3: Estratégias utilizadas para projetar operadores de mutação

Por fim, foram analisadas as estratégias utilizadas para gerar operadores de mutação para cada linguagem. As estratégias consistem na combinação de uma fonte e uma abordagem, descrita nas seções anteriores. A distribuição de artigos por estratégia é apresentada na Tabela 9.

Tabela 9. Estratégias selecionadas no mapeamento.

Símbolo	Fontes	Abordagens de apoio	#Artigos
●	Falhas comuns	Sistemático	1
■	Falhas comuns	Não sistemático	17
◆	Características específicas	Sistemático	2
▲	Características específicas	Não sistemático	16
★	Aprimoramentos	Sistemático	0
⬟	Aprimoramentos	Não sistemático	4

Foram usados os símbolos (círculo, quadrado, losango, triângulo, estrela e pentágono) para representar cada estratégia na representação gráfica. Os trabalhos encontrados foram dispostos em ordem cronológica, tal disposição pode ser visualizada na Figura 15.

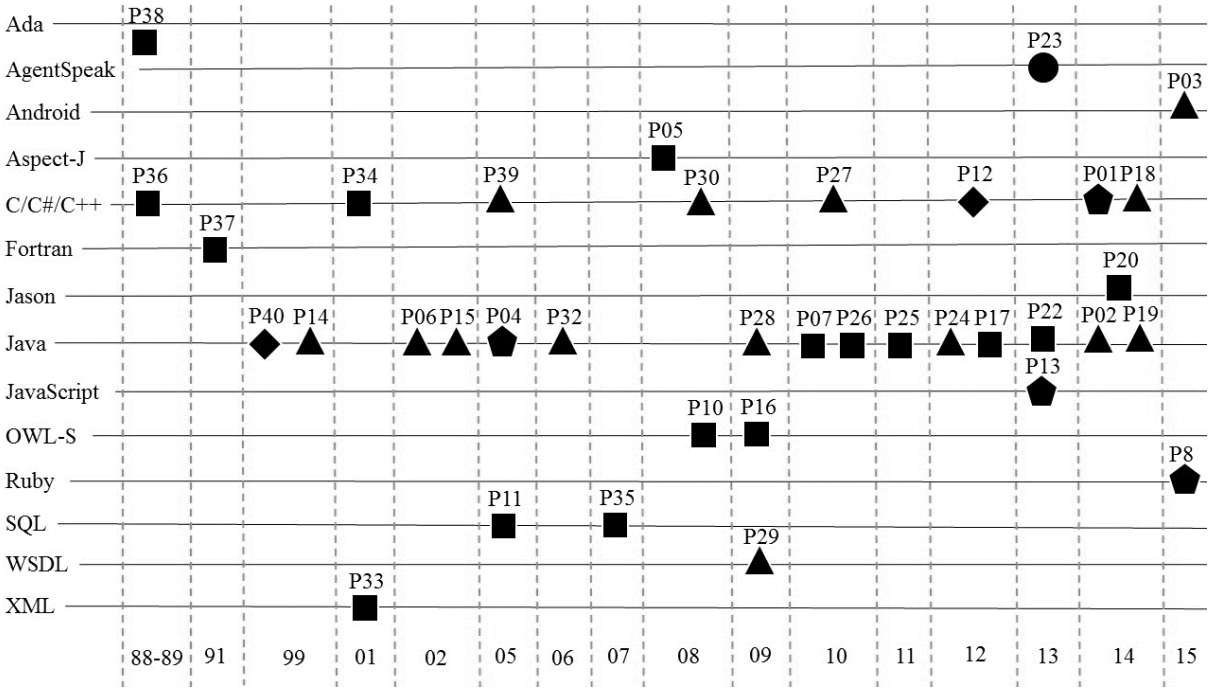


Figura 15. Artigos distribuídos cronologicamente (legenda na Tabela 9).

A estratégia representada por um círculo refere-se a trabalhos relatando definição de operadores de mutação usando falhas comuns relatadas pelos desenvolvedores e utilizando uma abordagem sistemática. Ele foi encontrado em apenas 1 artigo (verificar Tabela 8).

A estratégia representada por um quadrado refere-se a trabalhos relatando definição de operadores de mutação usando também falhas comuns relatadas pelos desenvolvedores,

mas utilizando uma metodologia não-sistemática, geralmente baseadas em conhecimento das pesquisas. É a estratégia mais comum, encontrada em 17 trabalhos acadêmicos.

A estratégia representada por um losango refere-se a trabalhos que empregaram características e especificidades da linguagem também usando uma metodologia sistemática para definir falhas contempladas pelos operadores propostos. Ela foi encontrada em dois artigos (P12 e P40).

A estratégia representada por um triângulo refere-se a trabalhos que empregaram características e especificidades da linguagem, mas que não utilizam uma metodologia sistemática (utilizando o conhecimento prévio do pesquisador) para definir operadores de mutação. Encontrou-se 16 artigos, sendo a segunda estratégia mais comum neste estudo.

A estratégia representada por um começo refere-se a trabalhos em que os operadores foram concebidos a partir de qualquer limitação de um conjunto anterior de operadores, e para melhorá-lo uma metodologia sistemática foi utilizada. Nenhum documento foi encontrado usando essa estratégia. Finalmente, a estratégia representada por um pentágono refere-se a trabalhos em que os operadores foram concebidos a partir de qualquer limitação de um conjunto anterior de operadores, e para melhorá-lo uma metodologia não-sistemática foi utilizada para propor novos operadores. Quatro artigos foram encontrados usando essa estratégia.

Ainda em relação à estratégia do operador, a Figura 15 apresenta cronologicamente, a evolução da definição de operadores de mutação. Nela, cada símbolo definido representa uma estratégia definida na Tabela 9, e o identificador acima de cada símbolo representa o artigo que a descreve.

Verificou-se que os primeiros operadores propostos para diferentes linguagens são projetados com base em falhas comuns com abordagem não sistemática (quadrado). Uma razão possível é a ausência de linha de base para executar e que uma tentativa para seguir a hipótese de programador competente. O critério ao longo do tempo estabelecido como critério de teste potentes, e também novas linguagens surgem, e com eles, as suas características específicas. Assim, o uso da estratégia de design mutação não sistemática tem sido intensificada com características específicas- (triângulo).

A mesma estratégia se repete em outras plataformas que têm a linguagem Java como base, como a Aspect-J [P05] e Android [P03], onde os autores relataram que os operadores para a linguagem Java não cobrem características específicas de Aspect-J e Android plataformas, respectivamente.

O primeiro artigo que propôs operadores de mutação para a linguagem Java utilizando recursos específicos desta linguagem também foi o primeiro trabalho que gerou novos operadores de mutação utilizando uma abordagem sistemática [P40] (losango). Esta estratégia não tem sido usada muitas vezes (além [P40], apenas o papel [P23] em 2013), o que pode indicar uma possível oportunidade de pesquisa para novos operadores de mutação.

Ao longo dos anos, foi possível medir a robustez das linguagens por meio de estudos empíricos. Assim, lacunas deixadas pelos operadores anteriores foram descobertas e, consequentemente, vem ganhando a possibilidade de utilizar tais dados empíricos não sistematicamente [P01] (Pentágono). O artigo em questão realiza estudos empíricos sobre operadores de mutação para a linguagem C, especificamente os operadores que removem comandos do código. Os dados obtidos do estudo apontaram a possibilidade da criação de novos operadores.

3.4. Discussões sobre os estudos preliminares

Neste capítulo, a dissertação apresenta um conjunto de estudos realizados com o intuito de levantar evidências da utilização de conjunto de falhas na garantia de qualidade de aplicações móveis.

O primeiro deles foi um mapeamento sistemático com o intuito e levantar trabalhos acadêmicos que utilizem técnicas de teste estruturais e baseado em defeitos no contexto de aplicações móveis Android. Dos trabalhos encontrados, verificou-se que os autores concentraram esforços nos principais componentes sintáticos de uma aplicação móvel Android. Um modelo de falhas específico para aplicações móveis poderia contribuir com tais estratégias, pois apresentaria tipos de falhas que podem ser retratados nos componentes sintáticos de aplicações móveis, nesse caso de aplicações móveis Android.

Com base no que foi analisado no estudo exploratório, pode ser verificado que alguns operadores utilizados para a plataforma Java podem ser aprimorados para a Linguagem Android, afim de serem melhor adequados à estrutura semântica das aplicações móveis.

Apesar de ter a estrutura sintática da linguagem Java, os componentes específicos da linguagem Android possuem características que, quando sofrem a mutação dos operadores da linguagem Java, continuam causando um comportamento incorreto de acordo com o programa original, mas com um resultado de timeout. Com isso, casos de testes simples identificam este comportamento incorreto, o que não permite que o responsável pela geração dos casos de teste possa propor testes que cubram falhas mais significativas.

Em caso hipotético, se houvessem operadores que modificassem os trechos de código referentes aos componentes Android, de uma forma em que a aplicação não apresente um

defeito semelhante ao de *timeout* na maioria dos casos, casos de teste mais relevantes poderiam ser criados, e de acordo com a hipótese do efeito de acoplamento, falhas complexas referentes à plataforma móvel manipulada por tais trechos de código poderiam ser evitadas. Essa hipótese é confirmada no trabalho de DENG *et al*. (2015).

Um modelo de falhas específico para aplicações móveis poderia contribuir neste aspecto, pois forneceria um corpo de conhecimento sobre aspectos específicos de aplicações móveis, consequentemente de aplicações móveis Android.

Finalmente, foi realizado um estudo de mapeamento sistemático visando identificar estratégias realizadas para projetar operadores de mutação para diferentes linguagens de programação. Como resultado, encontramos 40 artigos que propõem operadores de mutação para 14 linguagens diferentes.

Acredita-se que tais resultados podem contribuir para o trabalho futuro de propor mudanças operadores em múltiplas plataformas. Observou-se que, para as linguagens mais robustas, é interessante investir em falhas que são específicas ou com base nas limitações dos operadores anteriores.

Foi possível verificar que, como novas linguagens surgiram, bem como a robustez das línguas mais antigas tem crescido, foram utilizadas novas estratégias para a definição do operador, como as que utilizam recursos específicos da linguagem ou dados empíricos de outros operadores. Contudo, não existem estudos empíricos que comparem o uso de estratégias com abordagens sistemáticas, acompanhado por estudos empíricos para provar a sua relevância.

Entende-se que o desenvolvimento de operadores de mutação para novas plataformas/linguagens, como plataformas móveis, necessita de um conhecimento prévio de falhas específicas de plataforma, principalmente porque eles têm linguagem robusta como uma base (por exemplo, o Android é baseado em Java para a plataforma móvel).

Com estes resultados empíricos, verifica-se que um modelo de falhas que possa fornecer um corpo de conhecimento sobre falhas específicas de plataformas móveis, auxiliaria no aprimoramento de estratégias de teste específicas para aplicações móveis, bem como no provável estabelecimento de novas estratégias. Com isso, o próximo capítulo apresenta um modelo de falha específico para falhas de aplicações móveis oriundas do uso de recursos do dispositivo e da plataforma.

CAPÍTULO 4 – MOBPI-FM: UM MODELO DE FALHAS ESPECÍFICO PARA APLICAÇÕES MÓVEIS

Este capítulo tem como objetivo apresentar o modelo de falhas específico para aplicações móveis propostos nesta pesquisa, chamado de MOBPI-FM (MOBile Platform Independent – Fault Model). Serão apresentados o processo de construção do modelo para cada característica específica presente na plataforma móvel, assim como a aplicação da técnica HAZOP para proposição dos desvios sintáticos possíveis em aplicações móveis.

Os testes para aplicações móveis constitui um novo desafio de pesquisa, muito por conta da contemporaneidade da plataforma e das características específicas que a mesma possui. Contudo, além da utilização de técnicas de teste mais tradicionais, novas formas de teste têm sido utilizadas neste contexto com o objetivo de buscar garantir a qualidade das aplicações móveis.

Este capítulo apresenta o modelo de falhas MOBPI-FM, projetado a partir de características específicas de mobilidade e com o objetivo de obter uma melhor compreensão de falhas que podem ocorrer em aplicações móveis. Para a concepção do modelo MOBPI-FM, utilizaram-se os passos apresentados na Figura 16, que representam a atividade de concepção do modelo da metodologia apresentada no Capítulo 1.

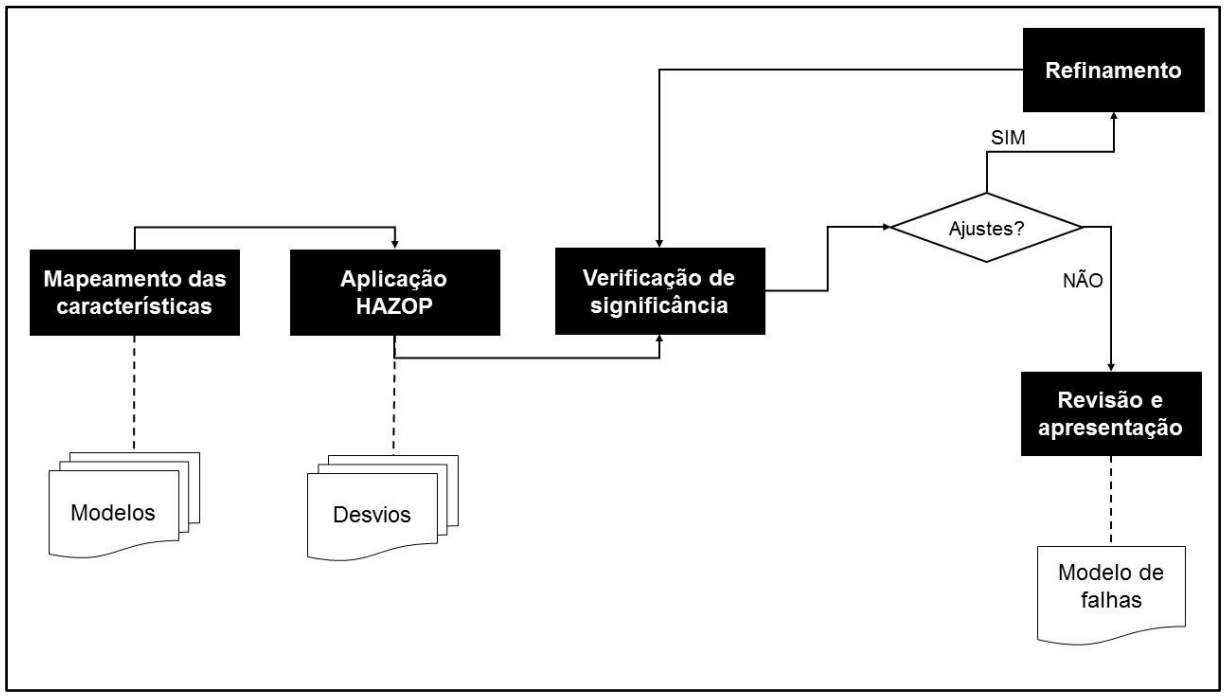


Figura 16. Metodologia de concepção do modelo MOBPI-FM.

Primeiramente, foi realizado o mapeamento das características de mobilidade, baseados nas características previamente reportadas e no seu relacionamento com o dispositivo e a plataforma. Em seguida, foi aplicada a técnica HAZOP, para a geração de desvios que pudessem ser modelados como falhas. Em seguida foi realizada a verificação de significância dos resultados da aplicação do HAZOP, e foi necessário realizar alguns refinamentos, que necessitaram de outra verificação, até o processo de revisão e apresentação. As subseções a seguir descrevem os passos apresentados na metodologia do modelo MOBPI-FM.

4.1. Mapeamento de Características de Mobilidade

O Capítulo 2 apresentou uma série de características de mobilidade que estão presentes em aplicações móveis. Tais características foram utilizadas para a concepção do modelo proposto nesta pesquisa na tentativa de derivar falhas que sejam específicas para esta plataforma. Observou-se que para tentar verificar o comportamento das características era necessário modelá-las, pois as definições, além de possuírem variância muito alta, são descritas de forma abstrata.

Partindo desse princípio, necessitou-se entender o relacionamento entre uma aplicação móvel, a plataforma para qual foi construída e o dispositivo em que opera. Para isso, utilizou-se diagramas da UML[22] para realizar um mapeamento do desse relacionamento.

O processo de construção do modelo iniciou-se pelo mapeamento das possíveis formas de interação de uma aplicação com o dispositivo e a plataforma na ótica das características levantadas anteriormente. Para isso, foram utilizados os diagramas de classe e de estados da UML. O diagrama de classes foi utilizado com o propósito de mapear quais os possíveis dados que tais aplicações operam, visando uma modelagem estática. Já o diagrama de estados mostra como a aplicação interage com as características, mostrando uma visão dinâmica desta interação.

Utilizando diagramas da UML, foi possível verificar a utilização das características no relacionamento da aplicação com o dispositivo que está localizado e a plataforma para qual foi construída de forma estática. Para essa análise estática, foi utilizado o diagrama de classes, pois é possível identificar como cada característica é composta, bem como suas hierarquias quando existirem e seu relacionamento com a aplicação (ver Figura 17).

[22] Linguagem de Modelagem Unificada (do inglês, UML - *Unified Modeling Language*) é uma linguagem de modelagem que permite representar um sistema computacional de forma padronizada.

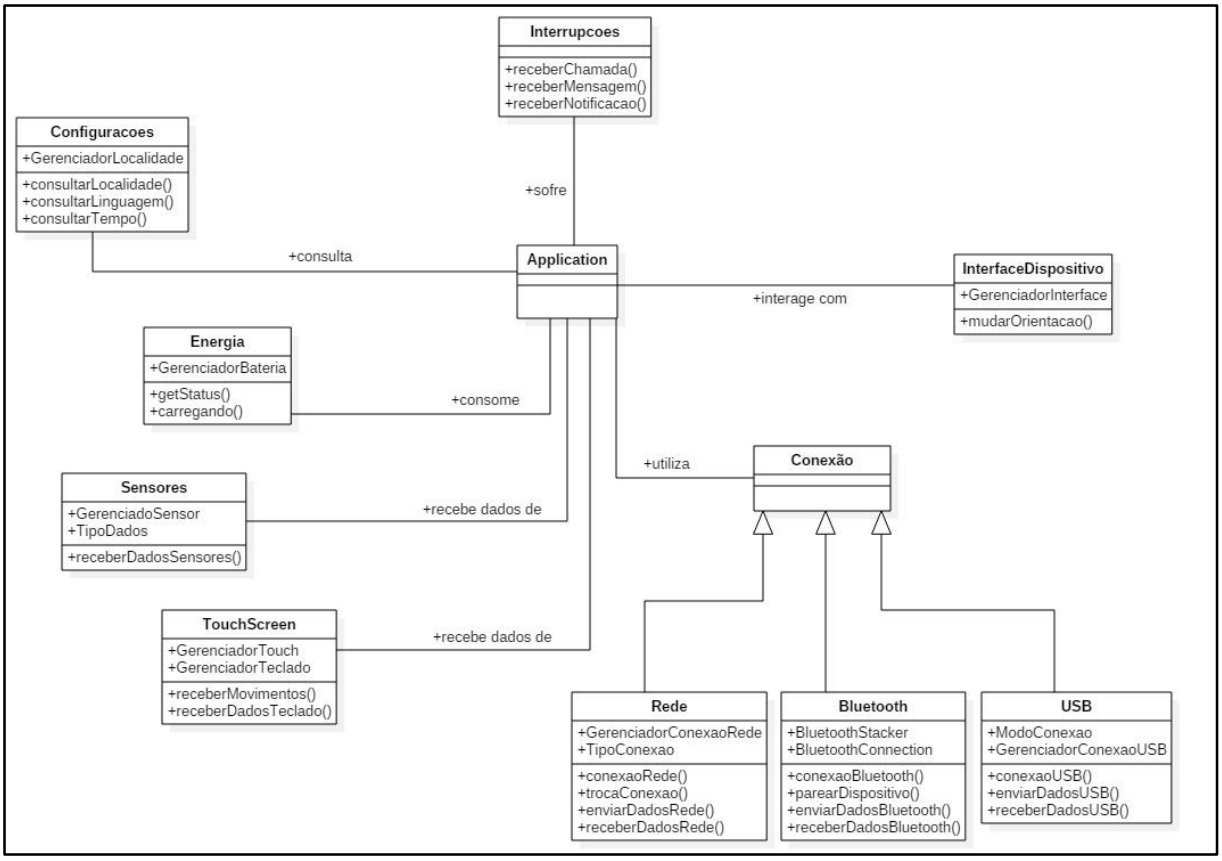


Figura 17. Modelagem estática das características de aplicações móveis.

Ainda em relação ao mapeamento, foi possível determinar um comportamento comum de aplicações na utilização das mesmas. Para realizar essa modelagem, utilizou-se o diagrama de estados da UML, cujos estados representam os vários possíveis estados de uma aplicação no uso de determinada característica. Após cada modelagem, foi realizada uma consulta da documentação das plataformas de aplicações móveis Android[23] e Windows Phone[24], com o intuito de verificar se o comportamento modelado correspondia as especificações de cada plataforma. A seguir, será realizado a descrição dos comportamentos mapeados por característica.

4.1.1. Conexão

Para a característica de conexão, interpretou-se que um aplicativo pode realizar conexão das seguintes maneiras:

- Com a rede, por meio de redes sem fio ou rede de dados;
- Por meio do *Bluetooth*;
- Por USB.

[23] Documentação da plataforma Android: http://developer.android.com/reference/packages.html
[24] Documentação da plataforma Windows Phone: https://msdn.microsoft.com/pt-br/library/windows/apps/ff626516(v=vs.105).aspx

Para a conexão de rede, verificou-se que um aplicativo está em um estado denominado **desconectado**, e a partir disso, realiza uma conexão com a rede. A partir do momento que realiza a conexão, assume o estado de **conectada**. Ao assumir este estado, interpreta-se que quando a aplicação móvel não recebe ou transfere dados por meio da rede, a mesma se encontra em um estado inativo (***idle***). O dispositivo pode trocar o modo de conexão com a rede, alternando entre sem fio e rede de dados. Ao sair da inatividade, uma aplicação pode **receber dados** pela conexão com a rede, ou **transferir dados** por uma conexão com a rede. A Figura 18 apresenta o diagrama de estados que descreve a conexão com a rede.

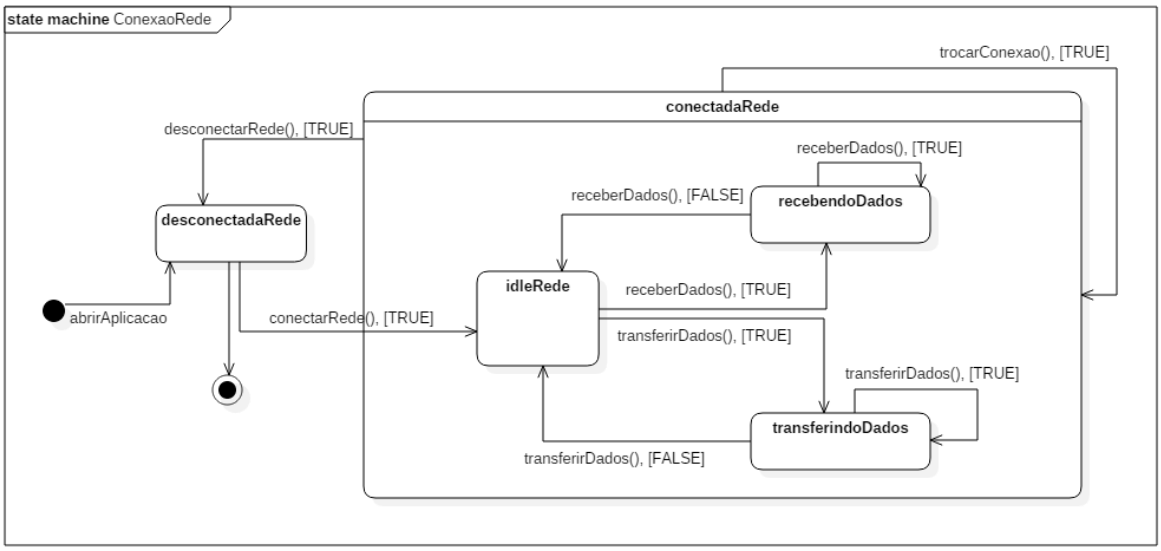


Figura 18. Característica de Conexão: conexão com a rede.

Para a conexão *Bluetooth* foi mapeado que a aplicação assume inicialmente o estado **desconectado**, e necessita realizar a conexão Bluetooth disponível no dispositivo, e ao realiza-la entra no estado de **conectada**. Porém, por conta da especificação da conexão Bluetooth, a aplicação para se comunicar com outro dispositivo, necessita estar **pareado**. Ao atingir este estado, a aplicação por um momento assume o estado de inatividade (***idle*)**, enquanto aguarda o comando, seja para **receber dados** ou para **transferindo dados**. A Figura 19 apresenta o comportamento descrito anteriormente.

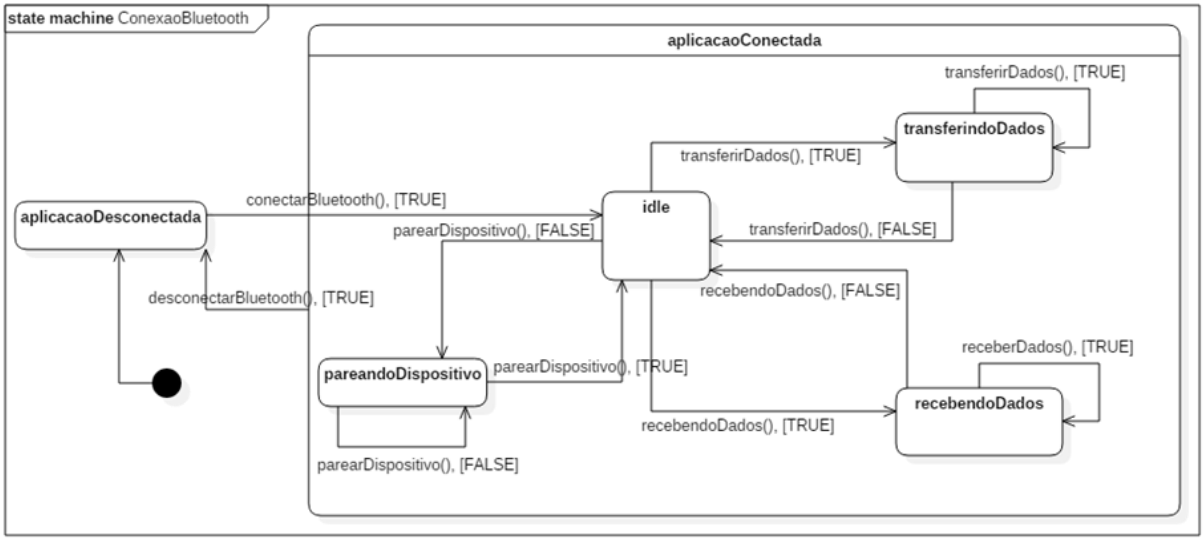


Figura 19. Característica Conexão: conexão por meio de Bluetooth.

Na conexão USB, a aplicação inicia no estado **desconectado**, realiza a conexão e assume o estado padrão de conectado. Incialmente assume o estado de inatividade (***idle***), aguardando o comando do usuário, que pode fazer com que a aplicação assuma o estado de **enviando dados** ou **recebendo dados**. A Figura 20 apresenta o comportamento descrito anteriormente.

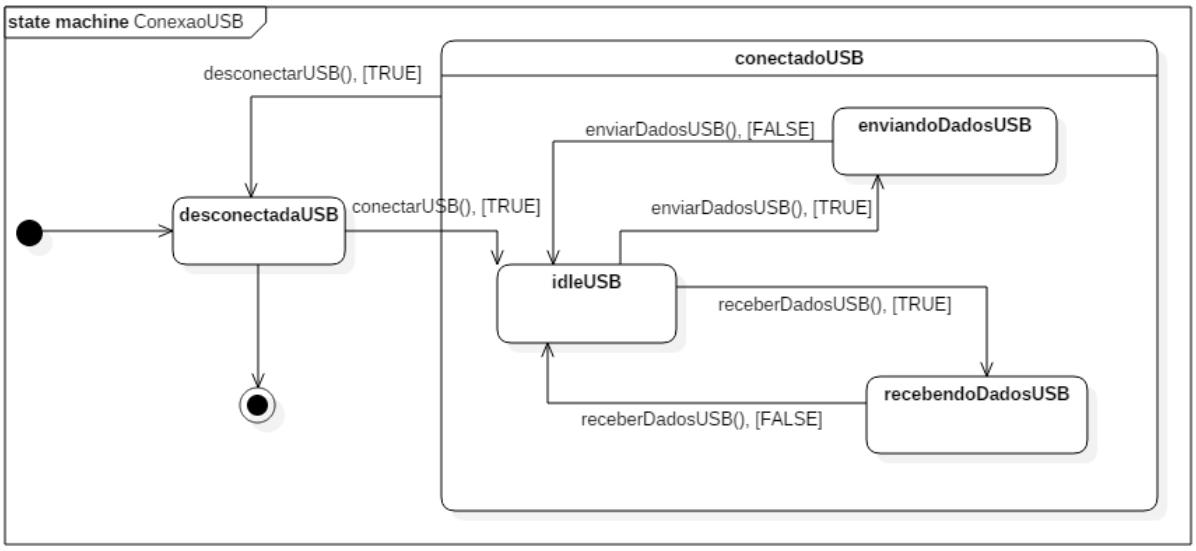


Figura 20. Característica Conexão: conexão por meio de USB.

4.1.2. Energia

Para a característica de energia, verificou-se que uma aplicação pode consultar o dispositivo para verificar o nível de energia da bateria.

Em uma situação de uso, o aplicativo está em **execução**, e caso seja necessário, o aplicativo verifica o nível da bateria. Com um nível mínimo estipulado pela plataforma, o

aplicativo entra em **execução restrita**. Com isso, o aplicativo deve consultar o nível de bateria para realizar determinada funcionalidade ou não. Existem aplicativos que realizam ou não determinadas funções dependentes do nível da bateria. Eventualmente, o dispositivo pode estar recebendo carga, o que faz o aplicativo estar em execução, porém no estado de **carregando** dispositivo. A Figura 21 representa tal relacionamento.

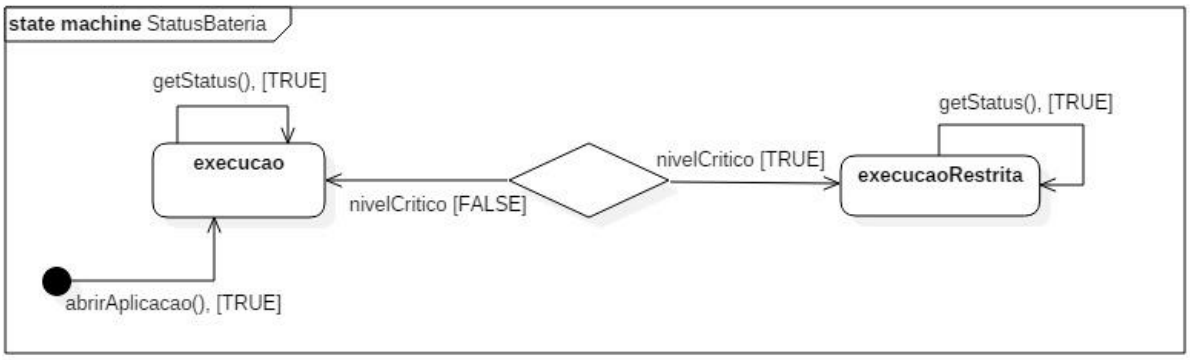


Figura 21. Característica Energia: consultando o nível da bateria.

4.1.3. Configurações

Quando uma aplicação acessa as configurações do dispositivo, entendemos que a mesma pode ter acesso a informações que o dispositivo contém, principalmente em relação a localidade do dispositivo, a linguagem que o usuário escolheu para o dispositivo e configurações de data e hora, que mudam em relação aos parâmetros descritos anteriormente. Para essa característica, assume-se que a aplicação móvel esteja em execução e realize consultas ao dispositivo sobre os parâmetros descritos. A Figura 22 apresenta o comportamento de aplicação móvel ao realizar essas consultas.

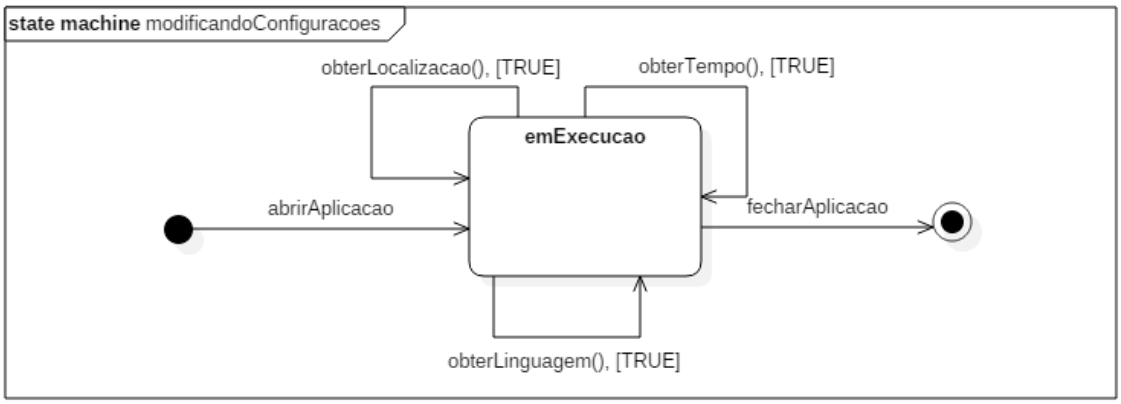


Figura 22. Característica Configurações do Dispositivo: consultando as configurações disponíveis.

4.1.4. Entrada de dados

Aplicativos possuem mais opções de entrada de dados do que por exemplo uma aplicação tradicional. Isso porque o dispositivo móvel permite ao aplicativo a utilização de métodos de entrada específicos do mesmo como movimentos da tela sensível ao toque, dados de sensores, entre outros. Para a característica de Entrada de Dados, foi interpretado que um aplicativo pode utilizar os seguintes métodos:

- Dados de sensores que o dispositivo possui;
- Dados de localização do dispositivo;
- Dados do teclado móvel;
- Movimentos da tela sensível ao toque.

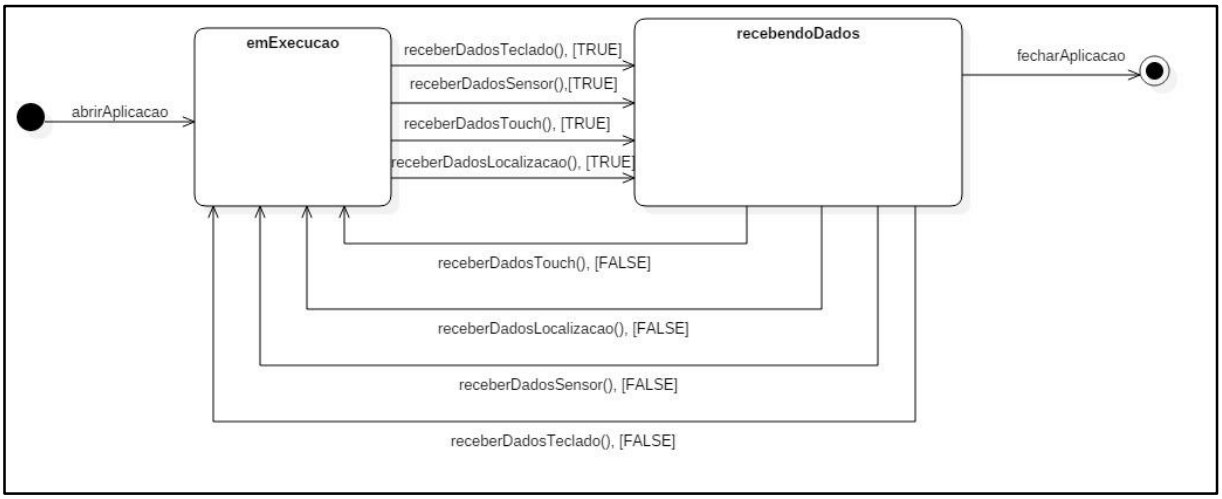


Figura 23. Característica Entrada de Dados: mapeamento da característica.

A Figura 23 apresenta o mapeamento realizado para a característica Entrada de Dados. Para o método de entrada de dados por sensores, verificou-se que a aplicação móvel está **em execução**, e em determinado momento, a aplicação solicita dados à determinado sensor, assumindo o estado de **recebendo dados**.

Para o método de entrada de dados de localização, verificou-se que a aplicação móvel está **em execução**, e em determinado momento, a aplicação solicita os dados de localização, sendo por meio do sensor GPS, assumindo o estado de **recebendo dados**.

Para o método de entrada de dados por meio do teclado, verificou-se que a aplicação móvel está **em execução**, e em determinado momento, componentes gráficos de entrada por texto solicitam dados por meio do teclado, assumindo o estado de **recebendo dados**.

Para o método de entrada de dados por meio da tela sensível ao toque, verificou-se que a aplicação móvel está **em execução**, e em determinado momento, a aplicação solicita

que seu usuário realize algum movimento com o dedo ou com algum aparato externo (e.g. canetas sensitivas), assumindo o estado de **recebendo dados**.

4.1.5. Interface com o Dispositivo

Dispositivos móveis atualmente variam muito de tamanho e forma, e com isso possuem interface com as aplicações projetadas para eles. Tais interfaces podem interferir na forma como a aplicação apresenta seus dados e componentes para seu usuário (e.g. adequar a rotação de tela), ou mesmo pausar ou finalizar seu funcionamento (e.g. botões físicos de menu). Para a característica de interface com o dispositivo, foi interpretado que um dispositivo pode fornecer as seguintes formas de interação:

- Rotação do dispositivo, alternando a disposição dos componentes de tela;
- Botão físico de menu, onde o usuário pode pausar o funcionamento de uma aplicação;
- Botão físico de saída, onde o usuário pode finalizar o funcionamento da aplicação;

Obviamente, a variedade de dispositivos pode fornecer outros botões físicos, mas entendemos que o seu comportamento em relação a aplicação se resume naqueles descritos anteriormente. Com relação à rotação de tela, verificou-se que uma aplicação pode estar em execução normalmente, em qualquer disposição que o dispositivo assuma, com isso, a aplicação assume os estados de disposição **em retrato** ou disposição **em paisagem**. A Figura 24 mostra o comportamento descrito anteriormente.

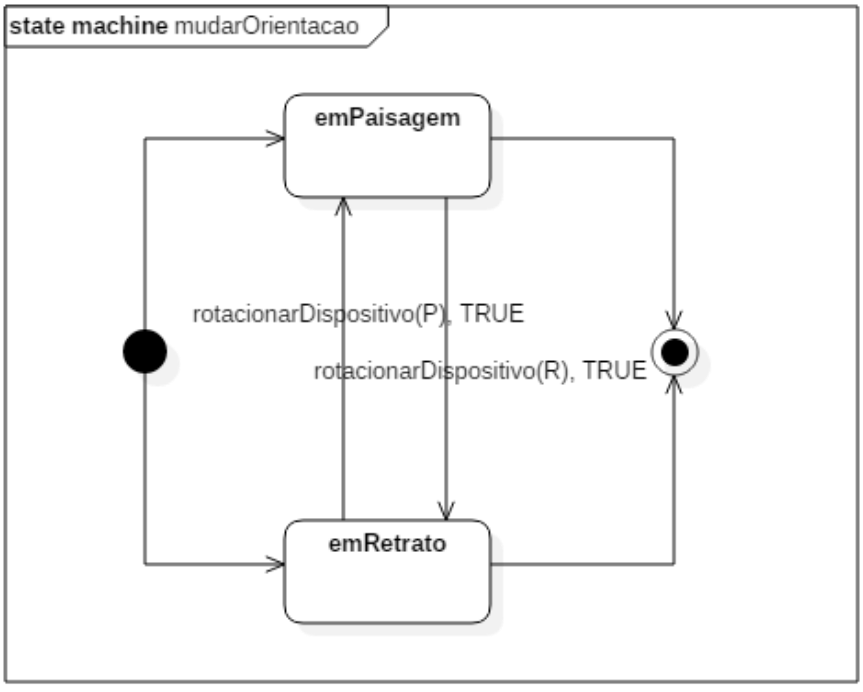


Figura 24. Característica Interface do Dispositivo: interação com a rotação de tela

Com relação à interação com os botões físicos, verificou-se que uma aplicação pode estar em **execução** por algum motivo o usuário decida pausar seu funcionamento ou finalizá-la. Com isso, a aplicação é encerrada. Contudo, aplicações móveis podem ser compostas por múltiplas telas, e a navegação entre tais telas é auxiliada por botões físicos. Nesse caso a aplicação continua assumindo o estado de execução, porém em outra tela, guiada pelo usuário. A Figura 25 mostra o comportamento descrito anteriormente.

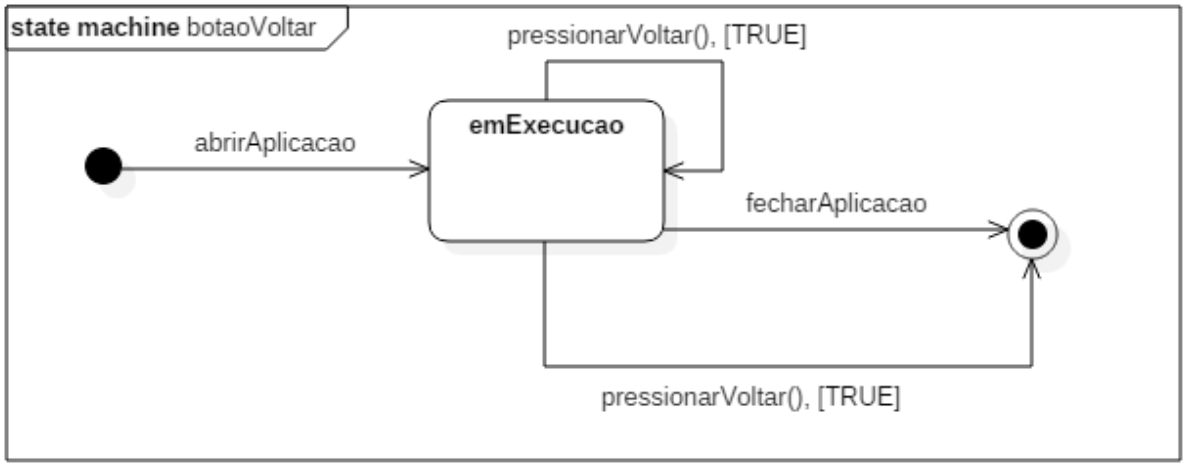


Figura 25. Característica Interface do Dispositivo: interação com botões físicos.

4.1.6. Interrupções

Em meio a tantas funcionalidades suportadas pelos dispositivos móveis, os mesmos ainda são utilizados para realizar chamadas telefônicas, e uma chamada tem prioridade sobre qualquer outra funcionalidade. Com isso, ao utilizar uma aplicação móvel, o seu funcionamento é interrompido quando o dispositivo recebe uma chamada. Ao fim da chamada, espera-se que o usuário possa retornar à atividade anterior sem perda de dados ou alteração no estado da aplicação. Para essa característica, foram mapeados os seguintes tipos de interrupções:

- Por chamada telefônica, quando um dispositivo a recebe;
- Por notificação, quando o dispositivo recebe ou gera uma notificação.

No que diz respeito a interrupções, aplicações móveis assumem o estado de **executando**, e em determinado momento, podem receber a chamada ou a notificação. Ao receber uma chamada, aplicações móveis assumem o estado **recebendo chamada**, o que implica em uma pausa no seu funcionamento. O mesmo acontece com uma notificação, onde assume o estado **recebendo notificação**. Espera-se que no encerramento da chamada ou no ato de recebimento da notificação, a aplicação retorne ao seu funcionamento normalmente. A Figura 26 apresenta o mapeamento do da relação descrita anteriormente.

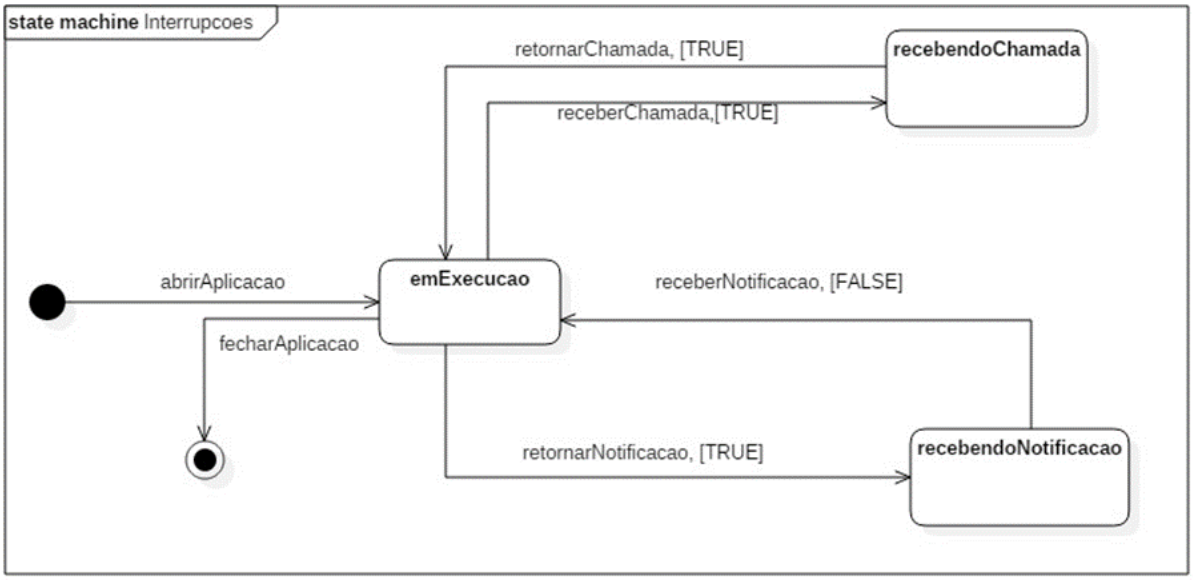


Figura 26. Característica Interrupções: tratamento das interrupções.

4.1.7. Acesso a Dados

Aplicações móveis podem manipular dados salvos no dispositivo, onde manipular pode ser interpretado como a criação, alteração e exclusão de dados. Isso inclui dados que os mesmos criam (e.g. um arquivo XML com dados exportados pela aplicação) e consequentemente manipulam, como até dados que a plataforma disponibiliza (e.g. contatos do usuário). As plataformas móveis existentes possuem mecanismos de acesso a dados diferentes entre si, e esse mecanismo muda também em relação ao dispositivo. Também se considera os dados que são manipulados por outras aplicações. Com isso, foram mapeados os seguintes comportamentos:

- Dados salvos pela aplicação;
- Dados salvos por outra aplicação.

Para os dados salvos pela aplicação no dispositivo, a aplicação assume o estado **em execução**, e a partir do momento que manipula os dados, assume o estado de **acessando dados do dispositivo**. Esse comportamento é descrito na Figura 27, e um comportamento semelhante acontece no acesso de dados de outas aplicações, onde ao enviar dados de outras aplicações, a aplicação assume o estado enviando **dados para aplicações**, e ao receber dados, assume o estado **recebendo dados de aplicações**. Esse processo é ilustrado na Figura 28.

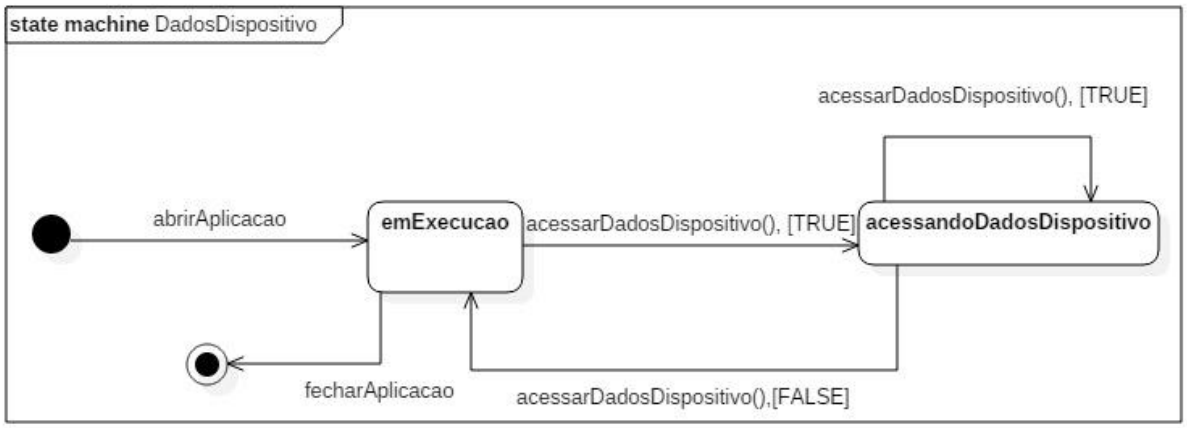


Figura 27. Característica Acesso a Dados: dados salvos no dispositivo.

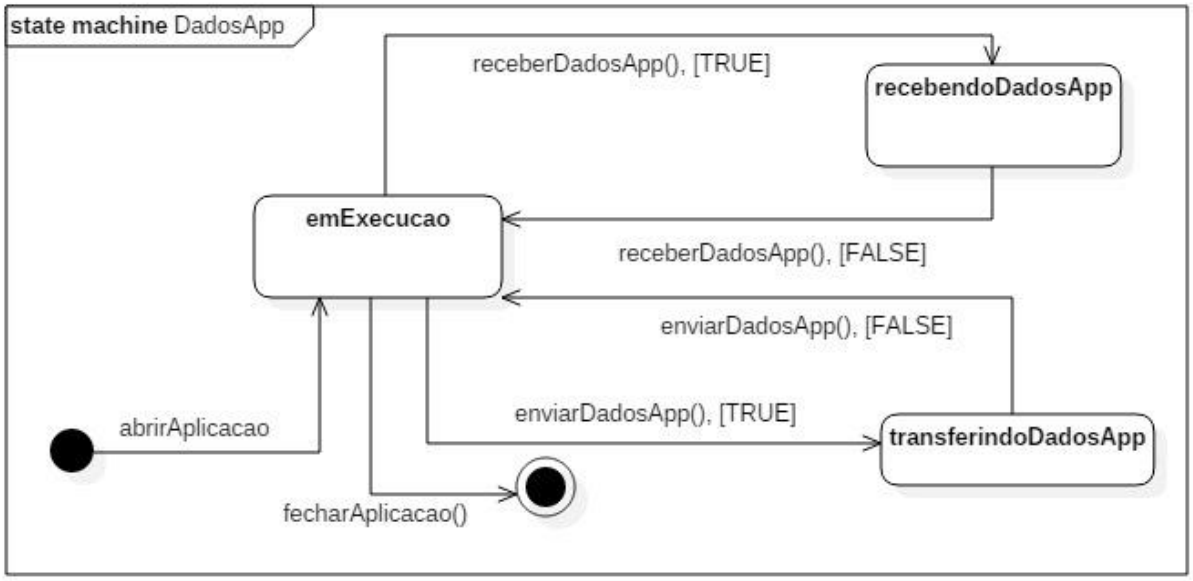


Figura 28. Característica Acesso a Dados: dados salvos de outras aplicações.

4.2. Aplicação da técnica HAZOP

Em poder das características mapeadas, era necessário pensar em como uma aplicação móvel falharia. Então surgiu a necessidade de uma forma de “prever” o que poderia acontecer quando uma aplicação interage com o dispositivo ou com a plataforma. Baseado nas estratégias apresentadas no capítulo anterior, optou-se por buscar uma técnica sistemática que auxiliasse na geração de tais falhas.

Foi com esse objetivo que foi utilizada a técnica HAZOP (STANDARD, 2005). Tal abordagem tem como objetivo gerar possíveis desvios em uma determinada especificação, com o intuito de prever quais danos podem ser causados caso o projeto não seja seguido.

Com base nos diagramas apresentados anteriormente, aplicou-se a técnica HAZOP para gerar os possíveis desvios, utilizando as palavras guias apresentadas. Para isso, era

necessário um conjunto adequado de palavras-guia da técnica, que fossem aplicáveis ao contexto de computação em geral.

Segundo relatos de aplicações anteriores, como os casos de (REDMILL et al., 1997), (MCDERMID e PUMFREY, 1994) previamente descritos no referencial teórico desta dissertação, e também (KIM et al., 1999) descrito nos estudos preliminares pelo uso do HAZOP na geração de falhas de sintaxe Java, as palavras-guias aplicáveis utilizadas estão sumarizadas na Tabela 10.

Tabela 10. Palavras-guia HAZOP utilizadas na geração dos desvios.

Palavras-guia	Descrição Aplicada à computação
NO OR NOT	Dados **não disponíveis** ou comandos **não realizados.**
MORE	Os dados são passados a uma **taxa maior** do que o pretendido, ou **dados a mais** são transferidos.
PART OF	Os dados ou comandos estão **incompletos.**
OTHER THAN	Dados ou comandos completos, mas **incorretos.**
EARLY	Determinado evento **adiantado** em relação ao tempo de relógio.
LATE	Determinado evento **atrasado** em relação ao tempo de relógio.
BEFORE	Determinado evento **adiantado** em relação ao especificado.
AFTER	Determinado evento **atrasado** em relação ao especificado.

Após a sumarização das palavras-guia que seriam utilizadas, foi seguido o procedimento ilustrado na Figura 29.

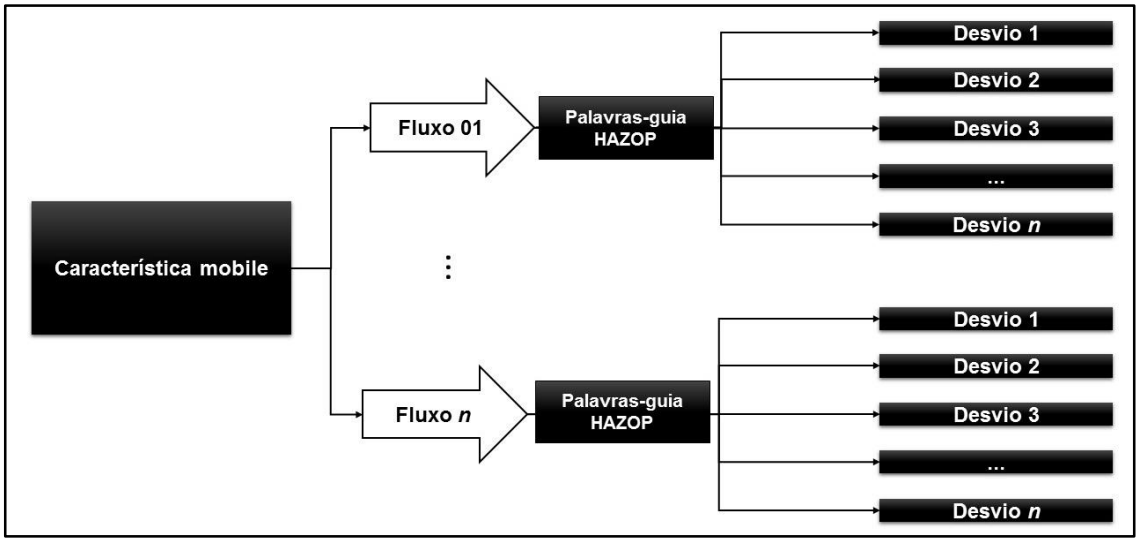


Figura 29. Procedimento realizado na aplicação do HAZOP.

Define-se como **fluxo** as transições dos diagramas de estados mapeados para cada característica. Para isso, os mesmos foram sumarizados e agrupados por característica. A

partir disso, as palavras-guia foram aplicadas para cada um dos fluxos, e com isso, para cada um dos fluxos sumarizados, 8 fluxos alterados pelas palavras-guia, e esses fluxos alterados foram chamados **desvios**.

Para uma melhor compreensão do processo descrito anteriormente, a Tabela 11 apresenta uma amostra coletada da aplicação das palavras-guias em um fluxo da característica Entrada de Dados, mostrado na Figura 23, onde o fluxo alterado é [***receberDadosSensor()***].

Tabela 11. Amostra de utilização da técnica HAZOP.

ID	Fluxo	Palavra-guia	Desvio gerado
F02	A aplicação recebe dados por meio do sensor do dispositivo [***receberDadosSensor()***]	*NO OR NOT*	A aplicação não recebe dados do sensor especificado
		MORE	A aplicação recebe dados a mais por meio do sensor especificado ou mais vezes que o esperado
		PART OF	A aplicação recebe dados de um sensor especificado de forma incompleta ou com dados incompletos
		OTHER THAN	A aplicação recebe dados incorretos ou de forma incorreta do sensor especificado
		EARLY	A aplicação recebe dados do sensor antes do tempo especificado
		LATE	A aplicação recebe dados do sensor depois do tempo especificado
		BEFORE	A aplicação recebe dados do sensor antes do especificado
		AFTER	A aplicação recebe dados do sensor depois do especificado

4.3. Verificação de significância e Refinamento

Como especificado na metodologia, foi necessário realizar uma verificação da significância dos desvios gerados pela técnica HAZOP. Essa avaliação faz parte da aplicação da técnica HAZOP, e filtra os fluxos que, na interpretação da equipe, não representavam um desvio que poderia gerar uma falha significativa.

Tal verificação foi realizada por três pesquisadores da área de engenharia de software, caracterizados na Tabela 12. Tal escolha foi baseada na experiência com pesquisa em engenharia de software e a disponibilidade para realizar a verificação.

Tabela 12. Caracterização dos pesquisadores que compuseram a equipe de verificação.

Pesquisador	Caracterização
PQ1	Pesquisador na área de engenharia de software, especificamente na área de Ecossistemas de Software, com ênfase no desenvolvimento de aplicações móveis.
PQ2	Pesquisadora na área de engenharia de software, especificamente na área de Teste de software, com ênfase em testes baseado em modelo.
PQ3	Pesquisadora na área de engenharia de software, especificamente na área de Teste de software, com ênfase em automação de testes.

Foram fornecidos aos pesquisadores os seguintes instrumentos:

- Uma planilha eletrônica com os todos os desvios gerados;
- Os diagramas de estado mapeados para cada característica.

Em posse dos instrumentos, os pesquisadores executaram o seguinte procedimento:

- Análise de cada um dos desvios gerados, verificando com os possíveis desdobramentos de cada desvio mediante o mapeamento apresentado anteriormente;
- Caso fosse verificado que o desvio fizesse sentido e impactasse no relacionamento da aplicação com a característica, o desvio era marcado como **significativo** [código SIG];
- Caso contrário, o desvio era marcado como **hipotético** [código HIP].

Como pedido opcional, foi solicitado aos pesquisadores que, se marcassem um desvio como significativo, que fornecessem um exemplo do desvio. Tal solicitação visa dar uma precisão de sua opinião. Para uma melhor visualização do processo descrito anteriormente, a Tabela 13 apresenta uma amostra da avaliação dos pesquisadores para os desvios alterados na subseção anterior. Quando um desvio recebia três avaliações da sigla SIG, foi considerado um indício de que o desvio era significativo. Para desvios que recebiam três ou duas avaliações da sigla HIP (hipotético), foi considerado um indício de desvio hipotético. Para desvios que receberam duas avaliações significativas, foi verificado com a equipe um consenso sobre a sua significância.

Tabela 13. Amostra de avaliação dos desvios com relação à significância.

Fluxo	Palavras-guia	Fluxos alterados	PQ1	PQ2	PQ3
F02	*NO OR NOT*	A aplicação não recebe dados do sensor especificado	SIG	SIG	SIG

	MORE	A aplicação recebe dados a mais por meio do sensor especificado ou mais vezes que o esperado	SIG	HIP	HIP
	PART OF	A aplicação recebe dados de um sensor especificado de forma incompleta ou com dados incompletos	HIP	SIG	HIP
	OTHER THAN	A aplicação recebe dados incorretos ou de forma incorreta do sensor especificado	HIP	SIG	HIP
	EARLY	A aplicação recebe dados do sensor antes do tempo especificado	SIG	HIP	HIP
	LATE	A aplicação recebe dados do sensor depois do tempo especificado	SIG	SIG	HIP
	BEFORE	A aplicação recebe dados do sensor antes do especificado	SIG	HIP	HIP
	AFTER	A aplicação recebe dados do sensor depois do especificado	SIG	SIG	HIP

Para os desvios que necessitavam de uma revisão, percebeu-se um comportamento padrão: 78 deles foram gerados pelas palavras-guia *MORE, PART OF* e *OTHER THAN*. Como justificativa da equipe, os desvios gerados por essas palavras-guia tinham como consequência o desvio gerado pela palavra NO.

Por exemplo, para os desvios apresentados na Tabela 13, os desvios foram gerados pelas palavras-guia *MORE, PART OF* e *OTHER THAN* tinham como consequência *"Não receber dados do sensor especificado"*. Tal comportamento se replicava nos demais desvios, com exceção dos desvios que lidavam com o recebimento ou transferência de dados. Por isso, o primeiro refinamento foi a utilização dessas três palavras-guia apenas em fluxos que manipulam dados.

Um segundo refinamento realizado a partir da observação das avaliações referentes às palavras *EARLY*, *LATE*, *BEFORE* e *AFTER*. As avaliações dos pesquisadores indicam que as alterações dos desvios gerados pelas 4 palavras tinham o mesmo resultado: causar uma alteração em determinado evento. Com isso, outro refinamento realizado foi o agrupamento dos desvios de tais palavras.

Verificou-se também a necessidade de mapear no modelo, falhas que fossem referentes a regras de negócio de uma aplicação móvel. Elas não constituem uma característica específica, mas fazem parte de qualquer aplicação computacional.

4.4. Apresentação do modelo

O modelo de falhas MOBPI-FM contempla os 4 níveis de falhas, mostrados no Capítulo 2. A arquitetura do modelo é representada no diagrama de classes apresentado na Figura 30.

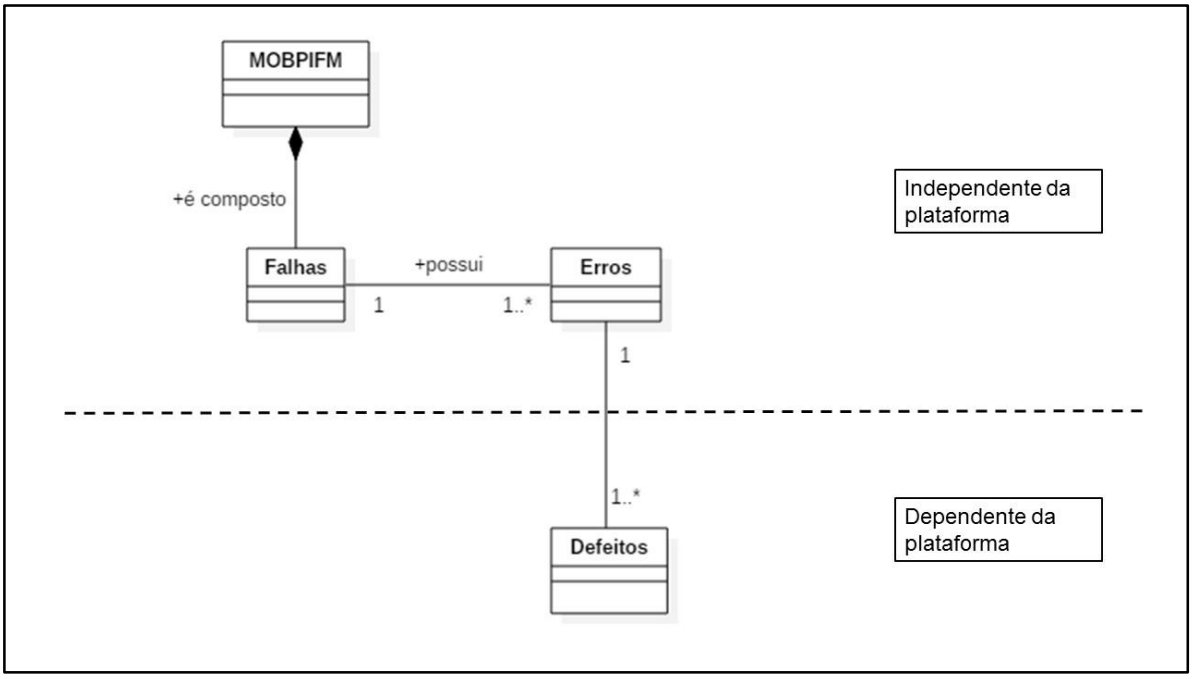


Figura 30. Arquitetura do modelo MOBPI-FM.

O modelo MOBPI-FM é composto por um conjunto de falhas geradas sistematicamente. Cada falha do modelo possui um ou mais erros associados a ela, e para cada erro, um ou mais defeitos associados a ele. Em relação à dependência da plataforma, o modelo possui dois níveis:

- **Independente de plataforma**: componentes do modelo que não necessitam estar associados à uma plataforma móvel específica. Tais componentes são as falhas e os erros do modelo;
- **Dependente de plataforma**: componentes do modelo que necessitam ser instanciados em uma plataforma específica. No caso do modelo MOBPI-FM, os defeitos necessitam ser complementados com componentes específicos da plataforma.

Com o objetivo de uma melhor organização, os itens (a saber: as falhas, os erros e os defeitos) são dispostos utilizando os identificadores compostos como descritos utilizando a taxonomia <<*nível*>> **-** <<*característica*>><<*numeração sequencial*>>, onde:

Tabela 14 Composição dos identificadores utilizados no modelo.

Componente	**Descrição**
<<nível>>	O nível de definição (e.g. (F) para falhas, (E) para erros e (D) para defeitos).

<<característica>>	Composto por duas letras, que procura resumir a característica à qual o item está associado.
<<numeração sequencial>>	Uma numeração sequencial utilizada para diferenciar o item.

4.4.1. Nível de Falhas

Esta subseção apresenta o nível de falhas do modelo. Como já visto anteriormente, o modelo MOBPI-FM é baseado em características específicas de aplicações.

Para o primeiro nível, esta dissertação apresenta, em tabelas separadas, pelas características, as falhas associadas à característica de entrada de dados (Tabela 15), conexão (Tabela 16), configurações do dispositivo (Tabela 17), gerenciamento de energia (Tabela 18), interface com o dispositivo (Tabela 19), interrupções (Tabela 20) e, finalmente, acesso a dados (Tabela 21). Tais falhas podem ser classificadas de acordo com sua origem:

- **Funcionalidade**: são falhas referentes à não execução de uma funcionalidade que envolva determinada característica;
- **Dados**: quando há uma falha na manipulação dos dados da aplicação móvel em seu relacionamento com a característica;
- **Evento**: quando uma falha adianta ou atrasa determinado evento da aplicação móvel no relacionamento com a característica;

Tabela 15. Falhas referentes a característica de Entrada de Dados.

ID Falha	Falha	Operação
F-DE01	A aplicação não recebe dados por meio dos sensores do dispositivo	Funcionalidade
F-DE02	A aplicação recebe dados inconsistentes por meio dos sensores	Dados
F-DE03	A aplicação atrasa ou adianta as funcionalidades de coleta de dados por meio dos sensores	Evento
F-DE04	A aplicação não recebe dados por meio do teclado	Funcionalidade
F-DE05	A aplicação recebe dados inconsistentes por meio do teclado	Dados
F-DE06	A aplicação não recebe dados de localização	Funcionalidade
F-DE07	A aplicação recebe dados de localização inconsistentes	Dados
F-DE08	A aplicação atrasa ou adianta eventos referentes à localização	Evento
F-DE09	A aplicação não recebe os movimentos do dispositivo	Funcionalidade
F-DE10	A aplicação recebe movimentos inconsistentes	Dados
F-DE11	A aplicação atrasa ou adianta eventos que envolvem movimentos do dispositivo	Evento

Tabela 16. Falhas da característica Conexão.

ID Falha	Falha	Operação
F-CN01	A aplicação não realiza a conexão com a rede	Funcionalidade
F-CN02	A aplicação tem o evento de conexão adiantado ou atrasado	Evento

F-CN03	A aplicação não suporta a troca de tipos de conexão com a rede	Funcionalidade
F-CN04	A aplicação adianta ou atrasa funcionalidades, devido a troca de conexão	Evento
F-CN05	A aplicação não recebe dados pela conexão com a rede	Funcionalidade
F-CN06	A aplicação recebe dados inconsistentes por meio da conexão com a rede	Dados
F-CN07	A aplicação tem o recebimento de dados pela rede alterado	Evento
F-CN08	A aplicação não envia dados pela conexão com a rede	Funcionalidade
F-CN09	A aplicação envia dados inconsistentes por meio da conexão com a rede	Dados
F-CN10	A aplicação tem o envio de dados pela rede alterado	Evento
F-CN11	A aplicação não realiza conexão USB	Funcionalidade
F-CN12	A aplicação tem a conexão USB adiantada ou atrasada	Evento
F-CN13	A aplicação não envia dados por meio da conexão USB	Funcionalidade
F-CN14	A aplicação envia dados inconsistentes por meio da conexão USB	Dados
F-CN15	A aplicação tem o recebimento de dados por USB alterado	Evento
F-CN16	A aplicação não recebe dados por meio da conexão USB	Funcionalidade
F-CN17	A aplicação recebe dados inconsistentes por meio de uma conexão USB	Dados
F-CN18	A aplicação tem o recebimento de dados alterado	Evento
F-CN19	A aplicação não realiza conexão Bluetooth	Funcionalidade
F-CN20	A aplicação tem o evento de conexão Bluetooth adiantado ou atrasado	Evento
F-CN21	A aplicação não realiza o pareamento com outra aplicação	Funcionalidade
F-CN22	A aplicação não envia dados por uma conexão Bluetooth	Funcionalidade
F-CN23	A aplicação envia dados inconsistentes por meio da conexão Bluetooth	Dados
F-CN24	A aplicação tem o envio de dados por Bluetooth alterado	Evento
F-CN25	A aplicação não recebe dados por uma conexão Bluetooth	Funcionalidade
F-CN26	A aplicação recebe dados inconsistentes por meio da conexão Bluetooth	Dados
F-CN27	A aplicação tem o recebimento de dados por Bluetooth alterado	Evento

Tabela 17. Falhas referentes a característica de configuração do dispositivo.

ID Falha	Falha	Operação
F-CF01	A aplicação não manipula corretamente a localidade do dispositivo	Funcionalidade
F-CF02	A aplicação possui eventos que são alterados por conta da manipulação errada da localidade	Evento
F-CF03	A aplicação não manipula corretamente a linguagem do dispositivo	Funcionalidade
F-CF04	A aplicação possui eventos que são alterados por conta da manipulação errada da linguagem	Evento
F-CF05	A aplicação não manipula corretamente dados temporais do dispositivo	Funcionalidade
F-CF06	A aplicação possui eventos que são alterados por conta da manipulação errada de dados temporais	Evento

Tabela 18. Falhas referentes ao gerenciamento de energia.

ID Falha	Falha	Operação
F-EN01	A aplicação não realiza a consulta do nível da bateria	Funcionalidade

F-EN02	A aplicação realiza a consulta do nível de bateria em um tempo diferente do especificado	Evento

Tabela 19. Falhas referentes a característica de Interface com o dispositivo.

ID Falha	Falha	Operação
F-DI01	A aplicação não adequa os componentes à rotação do dispositivo	Funcionalidade
F-DI02	A aplicação apresenta dados ou componentes inconsistentes quando o dispositivo altera sua orientação	Dados
F-DI03	A aplicação atrasa ou adianta eventos por conta da rotação do dispositivo	Evento
F-DI04	A aplicação não retorna à tela anterior ao pressionar o botão Voltar	Funcionalidade
F-DI05	A aplicação retorna à tela anterior com dados ou componentes inconsistentes	Dados
F-DI06	A aplicação apresenta atraso ao retornar a tela anterior	Evento
F-DI07	A aplicação não é encerrada ao pressionar o botão Voltar	Funcionalidade
F-DI08	A aplicação tem seus dados alterados ao se pressionar o botão "Voltar"	Dados
F-DI09	A aplicação atrasa seu encerramento	Evento
F-DI10	A aplicação não é pausada ao pressionar o botão central do dispositivo	Funcionalidade
F-DI11	A aplicação retorna da pausa com dados ou componentes inconsistentes	Dados
F-DI12	A aplicação tem o evento de pausa adiantado ou atrasado	Evento

Tabela 20. Falhas referentes a característica de Interrupções.

ID Falha	Falha	Operação
F-IT01	A aplicação não retorna ao estado anterior após receber uma chamada	Funcionalidade
F-IT02	A aplicação retorna ao estado anterior com dados inconsistentes	Dados
F-IT03	A aplicação atrasa o retorno ao estado anterior após receber uma chamada	Evento
F-IT04	A aplicação tem seu estado modificado depois de receber uma notificação	Funcionalidade
F-IT05	A aplicação tem dados alterados após receber uma notificação	Dados
F-IT06	A aplicação atrasa algum evento em virtude do recebimento de notificações	Evento

Tabela 21. Falhas referentes a característica de Acesso a Dados.

ID Falha	Falha	Operação
F-DA01	A aplicação não acessa os dados salvos no dispositivo	Funcionalidade
F-DA02	A aplicação acessa dados inconsistentes no dispositivo	Dados
F-DA03	A aplicação atrasa ou adianta o acesso a dados do dispositivo	Evento
F-DA04	A aplicação não consegue salvar dados no dispositivo	Funcionalidade
F-DA05	A aplicação salva dados inconsistentes no dispositivo	Dados
F-DA06	A aplicação sofre uma alteração no evento de salvar dados	Evento
F-DA07	A aplicação não envia dados para outra aplicação	Funcionalidade
F-DA08	A aplicação envia dados inconsistentes para outras aplicações	Dados
F-DA09	A aplicação adianta ou atrasa o envio de dados para outras aplicações	Evento
F-DA10	A aplicação não recebe dados de outra aplicação	Funcionalidade
F-DA11	A aplicação recebe dados inconsistentes de outras aplicações	Dados
F-DA12	A aplicação adianta ou atrasa o recebimento de dados de outras aplicações	Evento

4.4.2. Nível de Erros e Defeitos

O nível de erros do modelo MOBPI-FM é o nível onde cada falha é expandida, e utilizando a informação sobre os componentes da plataforma que implementam as características específicas. Esse nível ainda é independente de plataforma, contudo o modelo possui meta-informações, que devem ser preenchidas no nível de defeitos. As meta-informações possuem a seguinte taxonomia disposta na Tabela 22.

Tabela 22. Taxonomia das meta-informações contidas nos erros.

Meta dado	Descrição
<<nome classe/componente>>	A classe ou componente da plataforma responsável pelo gerenciamento das características
<<nome método/procedimento>>	Um método ou procedimento utilizado pelo componente para realizar determinada ação
<<nome parâmetro>>	Uma variável que manipula dados utilizado pela característica

Para mostrar a utilização do nível de erros do modelo, a Tabela 23 apresenta a expansão da falha F-CN01, na qual cada erro possui um ou mais meta-dados a serem preenchidos no nível de defeitos.

Tabela 23. Amostra no nível de erros referentes à falha F-CN01.

ID Erro	Erro
E-CN01	A aplicação não realiza a conexão com a rede, pois a classe/componente <<nome classe/componente>> não foi instanciada (o)
E-CN02	A aplicação não realiza a conexão com a rede, pois o método <<nome método/procedimento>> da classe/componente <<nome classe/componente>> não foi chamado
E-CN03	A aplicação não realiza a conexão com a rede, pois o parâmetro <<nome parâmetro>> utilizado pelo método <<nome método>> ou componente <<nome componente>> possui um valor incorreto
E-CN04	A aplicação não realiza a conexão com a rede, pois os meios de conexão do dispositivo estão desabilitados

Já no nível de defeitos, para cada instância de erro o modelo apresenta um defeito. Um defeito é gerado a partir do preenchimento das meta-informações de um erro, e os dados preenchidos são específicos de plataforma.

Para exemplificar a geração de um erro, a Figura 31 mostra o preenchimento dos dados para o erro E-CN01 apresentado na subseção anterior. Tal demonstração utilizou uma

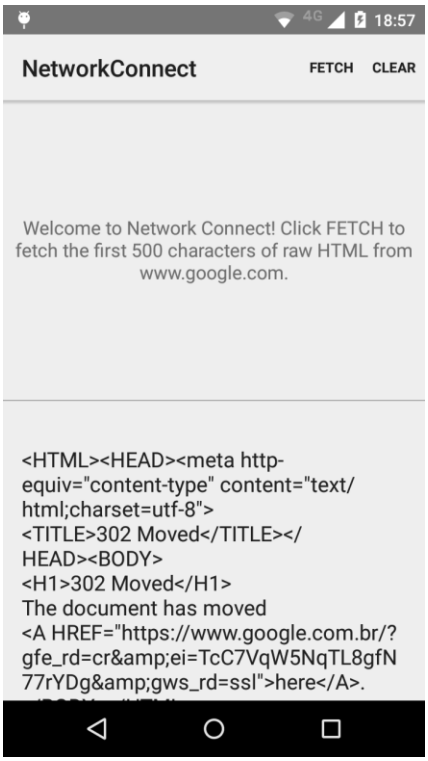
aplicação de exemplo da plataforma Android, disponível no portal de desenvolvedores[25] da plataforma

F-CN01	A aplicação não realiza a conexão com a rede pois a funcionalidade não foi implementada
E-CN01	A aplicação não realiza a conexão com a rede, pois a classe <<nome_classe>> não foi instanciada

- **Defeito instanciado**: A aplicação não realiza a conexão com a rede, pois a classe *HTTPUrlConnection* não foi instanciada
- **Engano**: HTTPUrlConnection var = null;

```
URL url = new URL(urlString);
//HttpURLConnection conn = (HttpURLConnection) url.openConnection();
HttpURLConnection conn = null;
conn.setReadTimeout(10000 /* milliseconds */);
```

Resultado esperado



Resultado obtido com o defeito

Figura 31. Defeito oriundo da instância do erro E-CN01 para a plataforma Android.

4.4.2.1. Comparação com outros trabalhos

Para os operadores de mutação propostos por DENG *et al*. (2015) encontram-se referências em falhas do modelo MOBPI-FM. A Tabela 24 mostra a distribuição dos códigos dos itens do modelo para cada operador proposto.

[25] *Samples – Android Developers*: http://developer.android.com/intl/pt-br/samples/index.html

Tabela 24. Mapeamento dos operadores de mutação para Android com itens do nível de defeitos do modelo MOBPI-FM

Operadores de mutação	Identificador da falha no modelo MOBPI-FM
Intent Payload Replacement (IPR)	F-CN05, F-CN06, F-CN07, F-CN08, F-CN09, F-CN13, F-CN14, F-CN15, F-CN16, F-CN17, F-CN18, F-CN22, F-CN23, F-CN24, F-CN25, F-CN26, F-CN27, F-IT01, F-IT03, F-IT05, F-DA01, F-DA02, F-DA03, F-DA04, F-DA05, F-DA06, F-DA07, F-DA08, F-DA09, F-DA10, F-DA11, F-DA12
Intent Target Replacement (ITR)	F-CN05, F-CN06, F-CN07, F-CN08, F-CN09, F-CN13, F-CN14, F-CN15, F-CN16, F-CN17, F-CN18, F-CN22, F-CN23, F-CN24, F-CN25, F-CN26, F-CN27, F-IT01, F-IT03, F-IT05, F-DA01, F-DA02, F-DA03, F-DA04, F-DA05, F-DA06, F-DA07, F-DA08, F-DA09, F-DA10, F-DA11, F-DA12
OnClick Event Replacement (ECR):	F-DE09, F-DE10, F-DE11
OnTouch Event Replacement (ETR):	F-DE09, F-DE10, F-DE11
Lifecycle Method Deletion (MDL)	F-DI04, F-DI05, F-DI06, F-DI07, F-DI08, F-DI09, F-DI10, F-DI11, F-DI12
Button Widget Deletion (BWD)	Não coberto pelo modelo
EditText Widget Deletion (TWD):	Não coberto pelo modelo
Activity Permission Deletion (APD):	F-DA01, F-DA03, F-DA10, F-DA12

Verifica-se pelo código do item que, no que diz respeito a características específicas de aplicações móveis, os operadores não cobrem as características de Entrada de Dados, Interface com o Dispositivo. Já o modelo não faz referência a possíveis itens de interface gráfica dentro de uma aplicação que estejam faltando (caso dos operadores BWD e TWD).

Com relação à classificação de falhas proposta por HOLL e ELBERZHAGER (2014), a Tabela 25 apresenta a distribuição da classificação também no contexto de falhas do modelo.

Tabela 25. Mapeamento da classificação de falhas com o nível de erros do modelo MOBPI-FM

Classe	Subclasse		Item do modelo MOBPI-FM
Comportamento	Transição	Errado	F-CN01, F-CN03, F-CN05, F-CN06, F-CN08, F-CN09, F-CN11, F-CN13, F-CN14, F-CN16, F-CN17, F-CN19, F-CN21, F-CN22, F-CN23, F-CN25, F-CN26, F-EN01, F-CF01, F-CF03, F-CF05, F-DI01, F-DI02, F-DI04, F-DI05, F-DI07, F-DI08, F-DI10, F-DI11, F-IT01, F-IT02, F-IT04, F-IT05, F-DA01, F-DA02, F-DA04, F-DA05, F-DA07, F-DA08, F-DA10, F-DA11
	Transição	Faltante	
	Transição	A mais	
	Transação	Errado	
	Transação	Faltante	
	Transação	A mais	
	Conteúdo dinâmico	Errado	
	Conteúdo dinâmico	Faltante	
	Conteúdo dinâmico	A mais	
	Evento	Errado	
	Evento	Faltante	
	Evento	A mais	
Design	Interação entre elementos	Errado	Não coberto pelo modelo
	Interação entre elementos	Faltante	Não coberto pelo modelo
	Interação entre elementos	A mais	Não coberto pelo modelo
	Estilo	Errado	Não coberto pelo modelo
	Estilo	Faltante	Não coberto pelo modelo
	Estilo	A mais	Não coberto pelo modelo
Conteúdo	Figura estática	Errado	F-DA04, F-DA05, F-DA07, F-DA08, F-DA10, F-DA11
	Figura estática	Faltante	F-DA04, F-DA05, F-DA07, F-DA08, F-DA10, F-DA11
	Figura estática	A mais	F-DA04, F-DA05, F-DA07, F-DA08, F-DA10, F-DA11
	Animação estática	Errado	Não coberto pelo modelo
	Animação estática	Faltante	Não coberto pelo modelo
	Animação estática	A mais	Não coberto pelo modelo
	Texto estático	Errado	F-CF01, F-CF03, F-CF05, F-DI01, F-DI02, F-DI04, F-DI05, F-DI07, F-DI08
	Texto estático	Faltante	F-CF01, F-CF03, F-CF05, F-DI01, F-DI02, F-DI04, F-DI05, F-DI07, F-DI08
	Texto estático	A mais	F-CF01, F-CF02, F-CF03, F-CF04, F-CF05, F-CF06, F-DI01, F-DI02, F-DI03, F-DI04, F-DI05, F-DI06, F-DI07, F-DI08, F-DI09

Verifica-se que, no nível de características, a maioria dos itens pode ser mapeada na classificação, com exceção da característica Entrada de Dados. Também se verifica que no nível de itens referentes ao dispositivo, os itens propostos pelo modelo MOBPI-FM não encontram referência na classificação proposta pelos autores. Já o modelo não encontra referência na classificação dos autores no que diz respeito a interface gráfica, por entender que isso é específico da aplicação.

Em linhas gerais, o modelo MOBPI-FM pode auxiliar tanto na geração de outros operadores de mutação, de forma a complementar o conjunto inicial proposto por DENG et al. (2015), e também pode auxiliar com novas classificações no trabalho de HOLL e ELBERZHAGER, (2014). De forma complementar, os trabalhos propostos contemplam itens que não constam no modelo, o que fornece ainda mais indícios de melhoria no mesmo.

Tal comparação realizou-se em nível textual. Os dois trabalhos supracitados possuem apoio ferramental que apoiam sua aplicação em seus respectivos contextos, o que mais uma vez, ressalta a necessidade do apoio ferramental para a melhor avaliação do modelo MOBPI-FM.

4.5. Considerações sobre o capítulo

O capitulo abordou a construção do modelo MOBPI-FM, sua base em características de mobilidade independente de plataforma e sua geração realizada de forma sistemática. Foi mostrado todo o processo de definição do modelo, desde a modelagem das características, passando pela utilização da técnica HAZOP e finalmente a verificação dos desvios gerados.

A utilização da técnica permite considerar as maneiras em que falhas podem ocorrer no relacionamento entre uma aplicação e o dispositivo e a plataforma. O modelo possui três níveis, sendo que os níveis de falha e erro são independentes de plataforma, já o nível de defeito é dependente onde se deseja instanciar o defeito.

Não é possível fornecer informações sobre a completude dos níveis de erro e defeito, pois uma avaliação empírica é altamente dependente de apoio ferramental, o que não foi possível conceber no decorrer do tempo da pesquisa. O apoio ferramental existente é bem avaliado empiricamente para demais plataformas. Porém, para o contexto de aplicações Android, faz-se necessário uma adaptação na inserção das falhas e na execução das aplicações que tiveram falhas inseridas.

Contudo, outro grande desafio que se apresenta é verificar se um modelo de falhas gerado sistematicamente é capaz de representar falhas de aplicações móveis reais. Para isso,

no capitulo a seguir é apresentado um estudo de viabilidade que procura avaliar a cobertura do modelo MOBPI-FM em falhas oriundas de aplicações móveis reais.

CAPITULO 5 - AVALIAÇÃO DA COBERTURA DO MODELO MOBPI-FM EM FALHAS DE APLICAÇÕES MÓVEIS REAIS

Com base nas falhas que compõem do modelo MOBPI-FM, este capitulo tem como objetivo apresentar um estudo de viabilidade, de caráter exploratório, conduzido com o intuito de verificar o quanto o modelo proposto é capaz de cobrir falhas de aplicações móveis reais.

Existem dois tipos de paradigmas de pesquisa que têm diferentes abordagens para estudos empíricos, um deles é a pesquisa exploratória (WOHLIN *et al*., 2012). A pesquisa exploratória está preocupada com o estudo de objetos em seu ambiente natural, fazendo assim que as possíveis conclusões sejam emergidas da observação. Isto implica que um projeto de pesquisa flexível é necessário para se adaptar às mudanças no fenômeno observado (WOHLIN *et al*., 2012). Nesse sentido, um estudo exploratório pode ser definido como um estudo empírico cujo objetivo principal não é encontrar uma resposta definitiva, mas sim criar um corpo de conhecimento empírico sobre a aplicação da tecnologia.

Com isso, é possível avaliar se a aplicação de determinada tecnologia condiz com o que foi especificado, ou seja, se atende de forma razoável aos objetivos inicialmente definidos, de forma a justificar (ou não) a continuação da pesquisa (SHULL *et al*., 2008).

Com base nos conceitos anteriores, este capítulo apresenta um estudo exploratório com o objetivo de verificar se o modelo MOBPI-FM é capaz de cobrir falhas que são reportadas em aplicações móveis reais. As próximas subseções descrevem as fases da definição do estudo, o planejamento do mesmo, como o estudo foi conduzido e a análise dos resultados do estudo.

5.1. Definição do estudo

5.1.1. Definição do escopo e objetivos

O propósito deste estudo foi responder à questão: "*O modelo MOPBPI-FM é capaz de cobrir falhas específicas de aplicações reais construídas para a plataforma Android?* ". Com isso, o objetivo geral do estudo é definido:

Analisar o modelo MOBPI-FM, com o propósito de caracteriza-lo em relação a cobertura de falhas específicas, do ponto de vista do pesquisador no contexto de aplicações móveis desenvolvidas para a plataforma Android.

Definido o objetivo do estudo, utilizou-se o modelo GQM (BASILI *et al*., 1994) para a definição das métricas a serem coletadas:

- *Qual o número de falhas de aplicações móveis reais que podem ser cobertas pelo modelo MOBPI-FM?* (**Métrica**: quantidade de falhas cobertas pelo modelo);

- *Qual o número de falhas cobertas pelo modelo que foram classificadas como falhas de características específicas?* (**Métrica**: quantidade de falhas de regras de negócio mapeadas);
- *Qual o número de falhas de aplicações móveis reais que não foram cobertas pelo modelo MOBPI-FM? (***Métrica**: quantidade de falhas não cobertas pelo modelo*)*.

5.2. Planejamento do estudo

Seu intuito principal é, a partir da lista de falhas reportadas para aplicações móveis reais, realizar uma associação entre elas e uma falha descrita no modelo MOBPI-FM. Para isso, as próximas subseções apresentam os itens que compõem o planejamento do estudo, seguindo diretrizes descritas em (WOHLIN et al., 2012) e (SHULL et al., 2008). Também foram selecionadas aplicações móveis reais que fossem representativas mediante às várias categorias de aplicações. Foram definidos também os instrumentos utilizados e finalmente foram definidas as ameaças à validade identificadas no estudo.

5.2.1. Formulação de hipóteses

Com o intuito de alcançar os objetivos propostos, este estudo está baseado na seguinte hipótese:

H0: Falhas de aplicações móveis reais não podem ser cobertas pelo modelo MOBPI-FM

Com o intuito refutar a hipótese nula levantada, temos a seguinte hipótese alternativa:

HA1: Falhas de aplicações móveis reais podem ser cobertas pelo modelo MOBPI-FM;

5.2.2. Seleção de aplicações móveis

Com relação às aplicações definidas para o estudo, era necessário obter conjunto que pudesse representar as muitas categorias e tipos de aplicações móveis existentes. Com isso, foi definido o seguinte critério de seleção de aplicações móveis para o estudo: Possuir uma lista de falhas reportadas por usuários ou por desenvolvedores ativos da aplicação móvel.

Com o critério definido para aplicações móveis, foi necessário definir também os critérios para selecionar as falhas reportadas para ela. Como critérios de aceitação das falhas, temos os seguintes:

- A descrição da falha descrita no idioma inglês ou português;
- A descrição deve ser de uma falha. Sugestões de melhorias ou adição de novos requisitos para a aplicação foram desconsiderados.

Definidos os critérios, a escolha das aplicações móveis partiu do repositório *FossDroid*[26], que visa agrupar e divulgar repositórios de projetos de código aberto específico para a plataforma Android. Ele divide as aplicações móveis indexados em categorias, assim

[26] FossDroid - Free and open source Android apps: https://fossdroid.com/;

como nas lojas, juntamente com os repositórios de código fonte e sua lista de falhas. Com isso, foram obtidas 58 aplicações distribuídas em 15 categorias, as quais possuem relação com as categorias da loja do *Google Play*. As aplicações e a respectiva quantidade de falhas selecionadas para cada uma são apresentadas na Tabela 26.

Tabela 26. Aplicações móveis selecionados para o estudo.

ID App	Aplicação	#Falhas	ID App	Aplicação	#Falhas
Categoria: Sistema			**Categoria: Multimídia**		
APP01	*AdAway*	287	APP30	*A2DP Volume*	210
APP02	*AndroidVNC*	202	APP31	*Vanilla Music*	16
APP03	*OSMonitor*	11	APP32	*BarcodeScanner*	21
APP04	*OI File Manager*	4	APP33	*AntennaPod*	153
Categoria: Jogos			**Categoria: Ciência**		
APP05	*Magic: The gathering*	12	APP34	*Open Training*	7
APP06	*Gobandroid*	10	APP35	*AnyMemo*	116
APP07	*FreeMiner*	18	APP36	*AnkiDroid*	84
APP08	*Solitaire*	120	APP37	*Android OBD-II Reader*	16
Categoria: Tempo			**Categoria: Desenvolvimento**		
APP09	*AlarmKlock*	48	APP38	*Github Mobile*	39
APP10	*CalDAV Sync Adapter*	39	APP39	*aLogCat*	28
APP11	*Birthday Adapter*	41	APP40	*AndLytics*	39
APP12	*A Time Tracker*	17	APP41	*VimTouch*	10
Categoria: Finanças			**Categoria: Leitura**		
APP13	*GNUCash*	22	APP42	*Web Opac*	52
APP14	*Androbank*	56	APP43	*Book Catalogue*	88
APP15	*DailyMoney*	16	APP44	*xkcdViewer*	16
APP16	*MyEnpenses*	3	APP45	*Document Viewer*	15
Categoria: Internet			**Categoria: Escritas & Notas**		
APP17	*ConnectBot*	301	APP46	*AnySoftKeyboard*	108
APP18	*Xabber*	120	APP47	*SimpleTask*	46
APP19	*K9Mail*	70	APP48	*Tasks*	38
APP20	*OwnCloud*	185	APP49	*MyOwnNotes*	18
Categoria: Navegação			**Categoria: Conectividade**		
APP21	*SatStats*	11	APP50	*NetworkMonitor*	14
APP22	*Mozilla Stumbler*	99	APP51	*SwiFTP*	10
APP23	*Vespucci*	120	APP52	*BetterWIFIOnOff*	6
APP24	*Transportr*	41	APP53	*MifareClassicTool*	6
Categoria: Segurança			**Categoria: Chamada & SMS**		
APP25	*Droidwall*	73	APP54	*Serval Mesh*	98
APP26	*Afwall*	105	APP55	*AndroidSMSPopup*	207
APP27	*Android IMSI-Catcher Detector*	63	APP56	*SMSSync*	39
APP28	*Authenticator*	8	APP57	*SMSSecure*	30
Categoria: Esporte & Saúde					
APP29	*Droidweight*	9	APP58	*Glucosio*	23

5.2.3. Seleção dos responsáveis pelo mapeamento

A realização do mapeamento das características foi realizada pelo pesquisador responsável pela pesquisa. Contudo, no intuito de tentar reduzir a ameaça da expectativa do pesquisador, foram selecionados dois desenvolvedores de aplicações móveis Android para realizar uma validação do mapeamento realizado pelo pesquisador.

Para a seleção dos desenvolvedores, primeiramente foi analisada a experiência no desenvolvimento de aplicações móveis Android em dois aspectos: (1) a quantidade de projetos na indústria no qual os desenvolvedores participaram; (2) o tempo de trabalho na função de desenvolvimento de aplicações. Finalmente, foi analisada a disponibilidade dos desenvolvedores em avaliar a quantidade significativa (3.865) de falhas levantadas nas aplicações móveis reais. Esse último fator foi o único de caráter eliminatório. A experiência foi coletada apenas para fins de caracterização. O resultado da caracterização é sumarizado na Tabela 27.

Tabela 27. Caracterização dos desenvolvedores responsáveis pela validação.

Desenvolvedor	Caracterização
D01	Possui três anos na função de desenvolvedor, com participação em quatro projetos de aplicações móveis Android, especificamente em aplicações de caráter industrial.
D02	Possui quatro anos na função de desenvolvedor, com participação em quatro projetos de aplicações móveis Android, especificamente de jogos para dispositivos móveis.

5.2.4. Instrumentos

Para realizar o estudo, os seguintes instrumentos foram definidos e formatados:

- A lista das falhas do modelo MOBPI-FM, no formato de planilha eletrônica onde eram sumarizadas as falhas, no mesmo formato apresentado na Seção 4.4.1;
- As falhas extraídas dos repositórios das aplicações móveis. Tais falhas foram extraídas de forma automática, por meio de um *web crawler*, e sumarizadas em uma planilha eletrônica, separadas por aplicação móvel real correspondente. A lista era composta por uma coluna ID, onde era apresentado um identificador para a falha; uma coluna título, que basicamente é uma descrição breve da falha; um texto explicativo, que continha uma descrição detalhada da falha, e uma coluna vazia, que deveria ser preenchida com o identificador da falha correspondente no modelo MOBPI-FM;
- Para os desenvolvedores que fizeram a avaliação das associações, foi apresentado o mesmo instrumento descrito anteriormente, porém com a associação proposta pelo pesquisador

5.2.5. Ameaças a validade

Nesta seção são discutidas as possíveis ameaças à validade dos resultados deste estudo de viabilidade.

Validade interna. Relacionada ao risco de outros fatores não identificados anteriormente, terem influenciado um eventual relacionamento de causalidade entre tratamento e resultado, sem o conhecimento prévio do experimentador.

- **Instrumentação**: os instrumentos adequados foram utilizados para todos os participantes, desde a planilhas utilizadas que passaram por revisão e foram submetidos a um mapeamento piloto, até o ajuste de implementação realizado no coletor de falhas. A medição dos resultados de mapeamento foi feita por meio de contagem das associações;
- **Seleção**: como mencionado na Seção 5.2.2, as aplicações móveis não foram selecionadas de maneira aleatória. Foram analisados tanto a distribuição por categorias relevantes, assim como uma quantidade igualitária de aplicações móveis por categoria, com exceção da categoria Esporte & Saúde, onde foram encontradas apenas duas aplicações móveis que satisfaziam os critérios de seleção;

Validade Externa. Questões relacionadas à ameaça dos resultados do estudo não serem generalizáveis a projetos reais na indústria. As principais ameaças identificadas foram:

- **Aplicações móveis Android**: foram selecionadas aplicações móveis Android que tinham um conjunto de falhas reportadas, a mesma plataforma das aplicações do estudo, como mencionado na Seção 5.2.3. Acredita-se que tal perfil possa refletir o ambiente de desenvolvimento para essa plataforma;
- **Desenvolvedores**: para a validação das associações, foram selecionados desenvolvedores de aplicações móveis Android, a mesma plataforma das aplicações do estudo, como mencionado na Seção 5.2.3. Acredita-se que tal perfil possa refletir o ambiente de desenvolvimento para essa plataforma;
- **Tempo**: em relação ao tempo de realização da verificação das associações, os dois desenvolvedores tiveram o mesmo período de tempo para a verificação (21 dias úteis). Em cada dia, os desenvolvedores avaliavam 180 associações, para que não houvesse uma desmotivação na condução de avaliação;
- **Configuração do experimento**: entende-se que o cenário utilizado nesse estudo caracteriza um ambiente real, visto que foram utilizadas aplicações móveis reais.

Validade de construção. Problemas relacionados à ameaça de generalizar os resultados do estudo à teoria que o sustenta. O risco de construção deste trabalho reside no processo utilizado na experiência. Por não se tratar de um experimento controlado, há um viés de confirmação dos resultados obtidos. As principais ameaças são:

- **Projeto do Experimento**: as falhas que compõem o modelo MOBPI-FM passaram por uma fase de concepção: mapeamento de características, aplicação do HAZOP, verificação de significância das falhas até a definição.
- **Fatores humanos (ou sociais)**: os desenvolvedores não estavam envolvidos em outros experimentos durante a realização deste estudo de viabilidade. Os participantes ainda participaram de forma espontânea como forma de adquirir conhecimento no desenvolvimento sobre falhas em aplicações móveis Android.

Validade de Conclusão. Relacionada a questões que ameaçam a habilidade de traçar conclusões corretas sobre o relacionamento entre tratamentos e resultados de um estudo experimental:

- **Confiabilidade das medidas**: os dados foram coletados a partir da associação entre falhas de aplicações móveis reais com falhas do modelo MOBPI-FM. A quantidade de falhas cobertas pelo modelo passou por uma verificação com desenvolvedores de aplicações móveis Android.
- **Confiabilidade da implementação dos tratamentos**: em relação à aplicação do tratamento não ser similar entre os diferentes participantes do estudo, o processo de associação das falhas de aplicações reais com falhas reportadas pelo modelo MOBPI-FM foi realizado rigidamente de acordo com o processo especificado, bem como a verificação por parte dos desenvolvedores.

5.3. Condução do estudo

Em poder dos instrumentos, o estudo foi conduzido conforme ilustrado na Figura 32, que mostra a realização do mapeamento das falhas reportadas no modelo MOBPI-FM. O mapeamento foi realizado verificando se a descrição da falha de uma aplicação real pode ser descrita por uma falha do modelo MOBPI-FM, especificamente realizando o seguinte procedimento:

- Definir se a determinada falha está relacionada com as regras de negócio da aplicação; caso seja, foi atribuído o código referente às falhas de regras de negócio [exemplo RN];

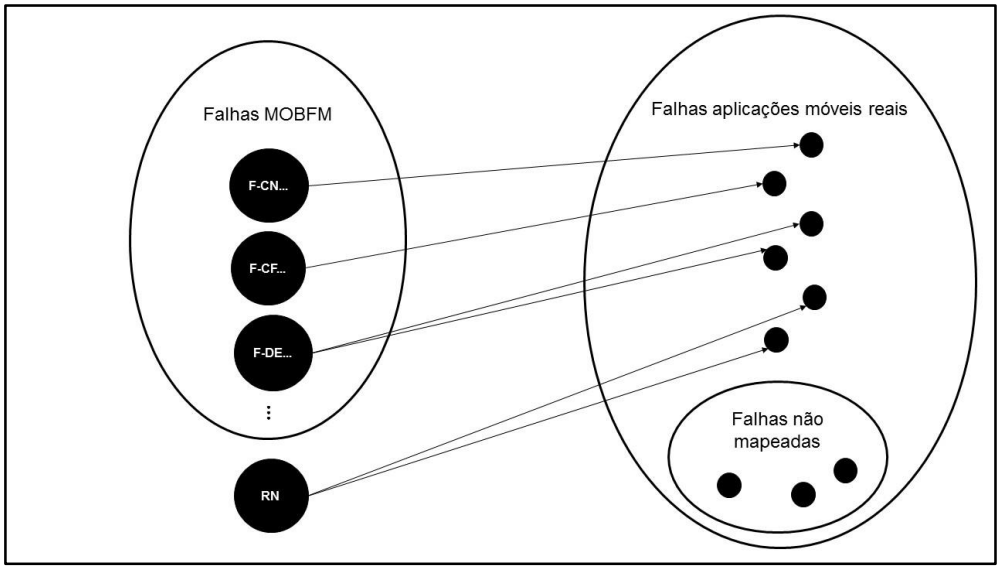


Figura 32. Representação da condução do estudo.

- Caso a descrição não encontre uma associação semelhante no modelo, a falha recebia um código sinalizando-a como “Não coberta” [e.g. NC];

Uma falha de regra de negócio pode ser definida como o não cumprimento de uma especificação de requisitos. A Tabela 28 mostra um exemplo de uma possível falha específica de aplicação móvel para uma falha de regra de negócio, no contexto de uma aplicação de calculadora.

Tabela 28. Exemplo da diferença entre uma falha RN e uma falha específica

Falha de regra de negócio	Falha específica de aplicação móvel
O campo resultado apresenta um valor incorreto para a soma dos números 2 e 2;	O campo resultado apresenta um valor correto para a soma dos números 2 e 2 quando o dispositivo está na orientação retrato. Porém ao posicionar o dispositivo na orientação paisagem, o valor da soma torna-se incorreto;

Após a execução desses passos, o conjunto de falhas mapeadas passou por um processo de avaliação por parte dos desenvolvedores selecionados. Os desenvolvedores realizaram a verificação das falhas que eram possivelmente cobertas pelo modelo, atribuindo rótulos para as associações realizadas pelo pesquisador. Tal verificação seguiu o seguinte critério:

- Se na avaliação do desenvolvedor, a associação realizada pelo pesquisador era correta conforme sua interpretação. Caso fosse, a falha recebia o rótulo “APROVADA”.
- Caso contrário, o desenvolvedor rotulava a falha como “REPROVADA”, e deveria indicar uma nova associação com a lista de falhas do modelo. Caso o mesmo não

encontrasse uma falha correspondente na lista, a falha não recebia nenhuma associação.

Para falhas que tinham dois rótulos “REPROVADA” e sem nenhuma associação de ambas as partes, tais falhas tinham seu mapeamento alterado para falhas não cobertas pelo modelo. Para falhas que tinham dois rótulos “APROVADA”, a falha era mantida com a classificação que foi previamente estabelecida pelo pesquisador.

Para falhas que receberam o rótulo “REPROVADA”, e tiveram associações definidas pelos desenvolvedores, adotou-se o seguinte procedimento:

- Se as associações de ambos os desenvolvedores fossem iguais, a falha assumia a associação proposta pelos desenvolvedores;
- Caso as avaliações fossem diferentes, os mesmos discutiam a associação, entrando assim em um consenso. Com isso, a falha recebia o resultado do consenso dos desenvolvedores. Tal discussão não contava com a participação do pesquisador.

5.4. Análise dos resultados obtidos

Os resultados obtidos são resumidos na Tabela 29, que apresenta o total de falhas da aplicação, a quantidade de falhas classificadas como falhas de regras de negócio, a quantidade de falhas específicas de características e a quantidade de falhas não cobertas pelo modelo e finalmente a taxa de cobertura (razão entre as falhas específicas e o total de falhas) por obtida por aplicação.

Verifica-se que o modelo conseguiu cobrir falhas em todas as aplicações, não em sua totalidade, mas a menor taxa de cobertura foi de 58% (APP43). Com isso, podemos considerar a hipótese alternativa definida para este estudo, de que o modelo MOBPI-FM cobre falhas específicas de aplicações móveis. Outras análises são apresentadas nas próximas subseções.

5.4.1. Cobertura geral

Analisando os dados de um modo geral de todas as falhas reportadas (3.564), 1.573 são falhas relacionadas à regra de negócio, e por isso não serão analisadas neste estudo visto que o modelo não se propõe a avaliar falhas desta categoria. As restantes (1.991) são falhas específicas de aplicações móveis e foram avaliadas neste estudo. Destas, 1.709 foram cobertas pelo modelo (representando 86% de todas as falhas específicas). Como consequência, 282 falhas específicas (14%) não foram cobertas pelo modelo, como ilustrado na Figura 33.

Tabela 29. Resultados da verificação de cobertura.

ID Aplicação	Total de falhas	Falhas RN	Falhas específicas	Falhas Cobertas	Falhas não cobertas	Cobertura
APP21	11	2	9	9	0	100%
APP45	15	4	11	11	0	100%
APP49	18	5	13	13	0	100%
APP10	39	12	27	27	0	100%
APP16	3	1	2	2	0	100%
APP57	30	12	18	18	0	100%
APP12	17	7	10	10	0	100%
APP24	41	17	24	24	0	100%
APP32	21	9	12	12	0	100%
APP37	16	8	8	8	0	100%
APP15	16	8	8	8	0	100%
APP40	39	20	19	19	0	100%
APP05	12	8	4	4	0	100%
APP52	6	4	2	2	0	100%
APP06	10	7	3	3	0	100%
APP31	16	12	4	4	0	100%
APP28	8	6	2	2	0	100%
APP11	41	15	26	25	1	96%
APP20	185	76	109	103	6	94%
APP42	52	35	17	16	1	94%
APP19	70	40	30	28	2	93%
APP25	73	29	44	41	3	93%
APP35	116	43	73	68	5	93%
APP46	108	59	49	45	4	92%
APP47	46	23	23	21	2	91%
APP33	153	74	79	72	7	91%
APP38	39	18	21	19	2	90%
APP48	38	28	10	9	1	90%
APP17	301	128	173	155	18	90%
APP54	98	60	38	34	4	89%
APP51	10	2	8	7	1	88%
APP14	56	40	16	14	2	88%
APP56	39	10	29	25	4	86%
APP22	99	29	70	60	10	86%
APP09	48	20	28	24	4	86%
APP18	120	50	70	60	10	86%
APP23	120	50	70	60	10	86%
APP27	63	23	40	34	6	85%
APP08	20	7	13	11	2	85%
APP39	28	15	13	11	2	85%
APP02	202	66	136	115	21	85%
APP26	105	33	72	60	12	83%
APP50	14	8	6	5	1	83%
APP13	22	11	11	9	2	82%
APP30	210	62	148	121	27	82%
APP29	9	4	5	4	1	80%
APP41	10	5	5	4	1	80%
APP01	287	148	139	110	29	79%
APP36	84	26	58	44	14	76%
APP04	4	0	4	3	1	75%
APP58	23	12	11	8	3	73%
APP03	11	4	7	5	2	71%
APP44	16	4	12	8	4	67%
APP53	6	3	3	2	1	67%
APP34	7	4	3	2	1	67%
APP07	18	12	6	4	2	67%
APP55	207	134	73	48	25	66%
APP43	88	21	67	39	28	58%

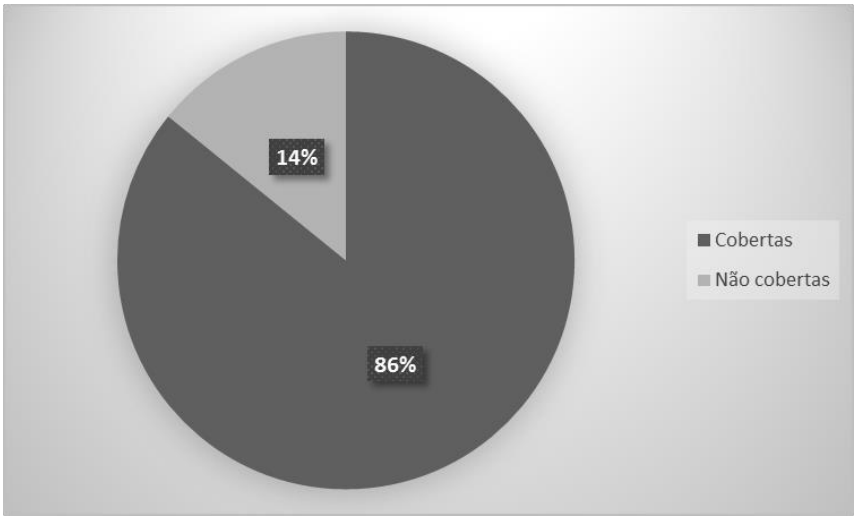


Figura 33. Distribuição dos resultados de cobertura geral.

Algumas falhas do modelo MOBPI-FM não cobriram nenhuma falha do conjunto obtido de aplicações reais, a saber:

- F-CN04, F-CN12, F-CN15, F-CN18, F-CN19, F-CN20, F-CN22, F-CN25, F-CN26 e F-CN27 para a característica de Conexão;
- F-EN02 para a característica de Energia;
- F-DI08 e F-DI12 para a característica de Interface com o Dispositivo;
- F-IT03 e F-IT06 para a característica de Interrupções, e;
- F-DA09 para a característica de Acesso a Dados.

Outras falhas do modelo cobriram apenas uma falha do conjunto de aplicações reais, a saber:

- F-CN10, F-CN16 e F-CN21 para a característica de conexão;
- F-EN01 para a característica de energia;
- F-CF02 para a característica de configuração do dispositivo;
- F-DE08 para a característica de entrada de dados;
- F-DI03, F-DI07 e F-DI10 para a característica de interface com o dispositivo, e;
- F-DA06, F-DA11 e F-DA12 para a característica de acesso a dados.

Uma análise nessas falhas mostra que 17 falhas reportadas acima (F-CN04, F-CN10, F-CN12, F-CN15, F-CN18, F-CN20, F-CN27, F-EN02, F-CF02, F-DE08, F-DI03, F-DI12, F-IT03, F-IT06, F-DA06, F-DA09 e F-DA12) são falhas que operam com eventos. Tal informação pode indicar uma baixa probabilidade dessas falhas ocorrerem. Contudo, estudos mais precisos devem ser realizados para confirmar isso, visto que várias falhas que operam com eventos foram reportadas.

As demais falhas do modelo tiveram dois ou mais falhas mapeadas. A quantidade de falhas de aplicações reais mapeadas para cada falha do modelo é apresentada na Tabela 30.

Tabela 30. Quantidade de falhas de aplicações reais para cada falha do modelo MOBPI-FM.

Ranking	ID Falha	Operação	Quantidade
1	**F-DA04**	Funcionalidade	168
2	**F-CN01**	Funcionalidade	148
3	**F-DA01**	Funcionalidade	111
4	**F-DE04**	Funcionalidade	102
5	**F-CN05**	Funcionalidade	69
6	**F-DE05**	Dados	68
7	**F-DE09**	Funcionalidade	62
8	**F-IT01**	Funcionalidade	60
9	**F-CN08**	Funcionalidade	58
10	**F-CF03**	Funcionalidade	56
11	**F-DI11**	Dados	53
12	**F-DI01**	Funcionalidade	48
12	**F-DA02**	Dados	48
14	**F-CN09**	Dados	46
15	**F-CN02**	Evento	45
16	**F-DI09**	Evento	42
17	**F-CF01**	Funcionalidade	40
17	**F-DA05**	Dados	40
19	**F-CF05**	Funcionalidade	39
20	**F-DE10**	Dados	32
21	**F-IT04**	Funcionalidade	30
22	**F-DI05**	Dados	29
23	**F-CN11**	Funcionalidade	28
24	**F-CN03**	Funcionalidade	25
25	**F-IT05**	Dados	24
26	**F-CN13**	Funcionalidade	22
26	**F-CN14**	Dados	22
26	**F-DA10**	Funcionalidade	22
29	**F-DI02**	Dados	21
29	**F-DA07**	Funcionalidade	21
31	**F-IT02**	Dados	20
32	**F-DE11**	Evento	12
32	**F-CN06**	Dados	11
34	**F-DE06**	Funcionalidade	11
35	**F-DA08**	Dados	10
36	**F-CN17**	Dados	8
36	**F-CN23**	Dados	8
38	**F-CN07**	Evento	5
38	**F-CF06**	Evento	5
38	**F-DE02**	Dados	5
38	**F-DA03**	Evento	5
42	**F-DI04**	Funcionalidade	4
43	**F-DE01**	Funcionalidade	3
43	**F-DI06**	Evento	3
45	**F-CN24**	Evento	2
45	**F-CF04**	Evento	2
45	**F-DE03**	Evento	2
45	**F-DE07**	Dados	2

Verifica-se que a falha mais ocorrida no conjunto de falhas reais é o fato de a aplicação não conseguir salvar dados no dispositivo, seguido da aplicação não realizar a conexão com

a rede e a aplicação não conseguir acessar dados salvos no dispositivo. Verifica-se que as quatro primeiras falhas possuem mais de 100 ocorrências, uma distância considerável para a quinta falha, por exemplo. Tal informação pode indicar que não há um grande exercício por meio de testes de tais características e mostrar que, ao utilizar as características de conexão e acesso a dados, uma verificação da existência de tais falhas pode ser realiza a partir da utilização do modelo.

A Figura 34 apresenta o percentual de falhas por operação que foram cobertas. Falhas referentes a funcionalidade (1132 falhas) e dados (448 falhas) obtiveram os maiores percentuais (66% e 26% respectivamente), enquanto falhas relacionadas a eventos tiveram cobertura de 8% (129 falhas). Isso indica que do conjunto de falhas cobertas, aquelas que relatam que uma funcionalidade de característica não foi executada é quase o dobro das demais. Isso pode fornecer informações para a priorização de técnicas ou estratégias de teste para aplicações móveis.

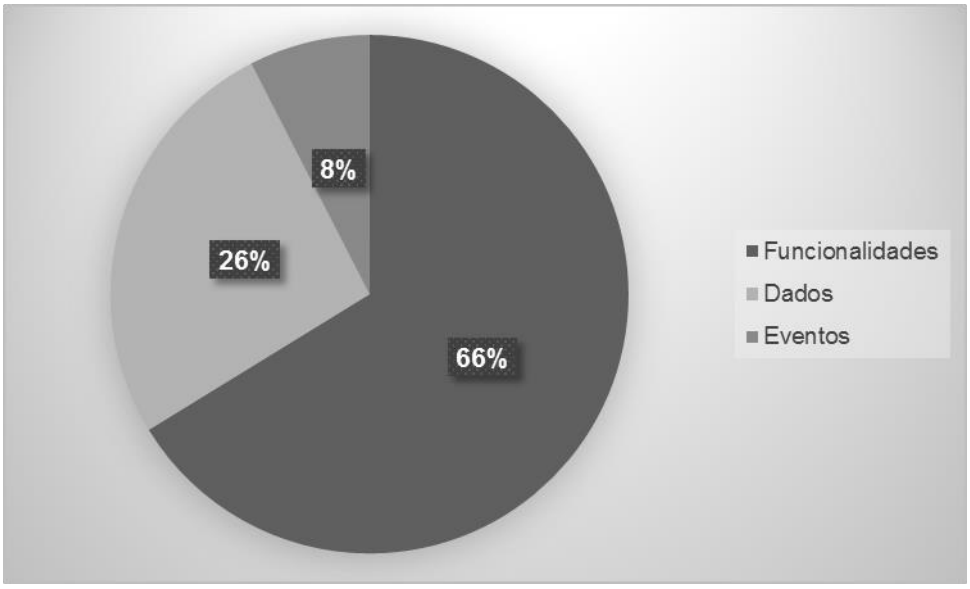


Figura 34. Distribuição das falhas por operação

Com relação às falhas específicas não cobertas no modelo, são falhas referentes à instalação e atualização das versões das aplicações móveis no dispositivo e referentes ao consumo de memória do dispositivo. Tais falhas não puderam ser mapeadas em um diagrama de estados. Com isso, os fluxos não foram identificados e, consequentemente, as possíveis falhas não foram identificadas.

Algumas amostras de falhas não cobertas pelo modelo são apresentadas a seguir, no formato *[ID_APLICAÇÃO]-[APLICAÇÃO]-[TITULO_FALHA]-[DESCRIÇÃO_FALHA]*:

“[APP04]-[AndroVNC] – APK Fails to install - Well after first uninstalling the app and then downloading it from the website directly on my phone. The installation does not work.”

No relato acima, observa-se que a aplicação falha no momento em que o sistema operacional inicia o procedimento de instalação. Falhas que relatam que determinada aplicação móvel não foi instalada corretamente no dispositivo, não são cobertas pelo modelo.

A princípio na concepção do modelo, pela visão limitada do pesquisador, não era possível mapear o relacionamento entre uma aplicação com recursos do dispositivo e da plataforma no momento que a mesma era instalada. Contudo, as amostras de falhas obtidas evidenciam a existência das falhas, e também que o engano que pode originar tal falha seja proveniente de escolha errada de versões da plataforma no código-fonte (uma definição errada da versão da plataforma, solicitar permissão para um recurso que o dispositivo não possua), ou até mesmo por uma falha proveniente do dispositivo (falta de espaço físico no dispositivo)

“[APP02]-[AlarmKlock] – Incompatibility with KitKat low-memory optimizations - I am running android 4.4.2 KitKat on an older device that only has 512 MB ram. If I enable KitKat's optimizations for low-memory devices (by setting ro.config.low_ram=false in build.prop), the Alarm Klock app misbehaves”

“[APP19]-[K9Mail] – java.lang.OutOfMemoryError when fetching mails - "Since a few days, k9mail has issues with retrieving / updating my inbox (both in background operation and when the Inbox is open).

Das amostras apresentadas acima, verifica-se que a aplicação móvel *[APP02]-[AlarmKlock]* necessita de mais memória do que o dispositivo disponibiliza para o mesmo. Na segunda amostra obtida da aplicação *[APP19]-[K9Mail]*, verifica-se que a mesma tentou utilizar mais recursos de memória do que o sistema operacional do dispositivo lhe permitiu. Falhas relacionadas ao consumo de memória pela aplicação móvel, em qualquer aspecto, também não são cobertas pelo modelo. Acredita-se que tais falhas ocorram pelo fato do dispositivo móvel não reservar determinada quantidade de memória RAM para a execução da aplicação ou a otimização de memória realizada pelas plataformas nos dispositivos pode refletir em falhas nas aplicações móveis.

Tais informações sobre falhas não cobertas evidenciam que o modelo necessita de refinamento, visto que tais falhas também são falhas específicas de aplicações móveis, pois se relacionam com a utilização de recursos limitados de memória do dispositivo, tanto a primária (memória RAM) quanto a secundária (espaço de armazenamento).

Estudos devem ser realizados para verificar o quanto uma aplicação pode manipular de tal característica. Por exemplo uma aplicação a nível de código fonte, não é capaz de verificar o quanto de espeço de armazenamento disponível um dispositivo possui, nem de verificar a memória RAM disponível para realizar uma determinada ação – como é possível realizar com o nível de bateria, por exemplo.

Portanto a princípio, são falhas que ocorrem no dispositivo e que influenciam no funcionamento da aplicação. Contudo, como apresentado na seção 4.4.2, o modelo no nível de erro apresenta determinados itens que tem sua origem no dispositivo (E-CN04 mostrado na Tabela 23). O refinamento a ser realizado no modelo pode partir desse ponto, e verificar como as falhas podem ser reportadas.

5.4.2. Cobertura por categoria de aplicações móveis

Para uma análise de cobertura do modelo em relação às categorias de aplicação móvel, os resultados são sumarizados na Tabela 31.

Tabela 31. Sumarização da cobertura de falhas por categoria de aplicação.

Categoria	Total de falhas	Falhas RN	Falhas específicas	Falhas cobertas	Falhas não cobertas	Cobertura
Tempo	145	54	91	86	5	95%
Escritas e Notas	210	115	95	88	7	93%
Desenvolvimento	116	58	58	53	5	91%
Internet	676	294	382	346	36	91%
Finanças	97	60	37	33	4	89%
Navegação	271	98	173	153	20	88%
Segurança	249	91	158	137	21	87%
Multimídia	400	157	243	209	34	86%
Ciência	223	81	142	122	20	86%
Jogos	60	34	26	22	4	85%
Conectividade	36	17	19	16	3	84%
Sistema	504	218	286	233	53	81%
Chamada & SMS	374	216	158	125	33	79%
Esporte e Saúde	32	16	16	12	4	75%
Leitura	171	64	107	74	33	69%

Verifica-se que para a categoria Tempo, o modelo atingiu a maior taxa de cobertura (95%), enquanto para a categoria Leitura o modelo atingiu 69%. De um modelo geral, o modelo consegue representar falhas da maioria das categorias de aplicações móveis. Tal informação também pode indicar a adequação do modelo para as determinadas categorias de aplicação. A Tabela 31 apresenta uma linha que separa as categorias de aplicações

móveis que tiveram cobertura de falhas acima da média geral (86%) das demais categorias. Ao total, nove categorias obtiveram cobertura acima da média, enquanto que seis categorias ficaram abaixo da média.

5.4.3. Cobertura por característica

A Figura 35 apresenta um gráfico com a distribuição da quantidade de falhas cobertas por características.

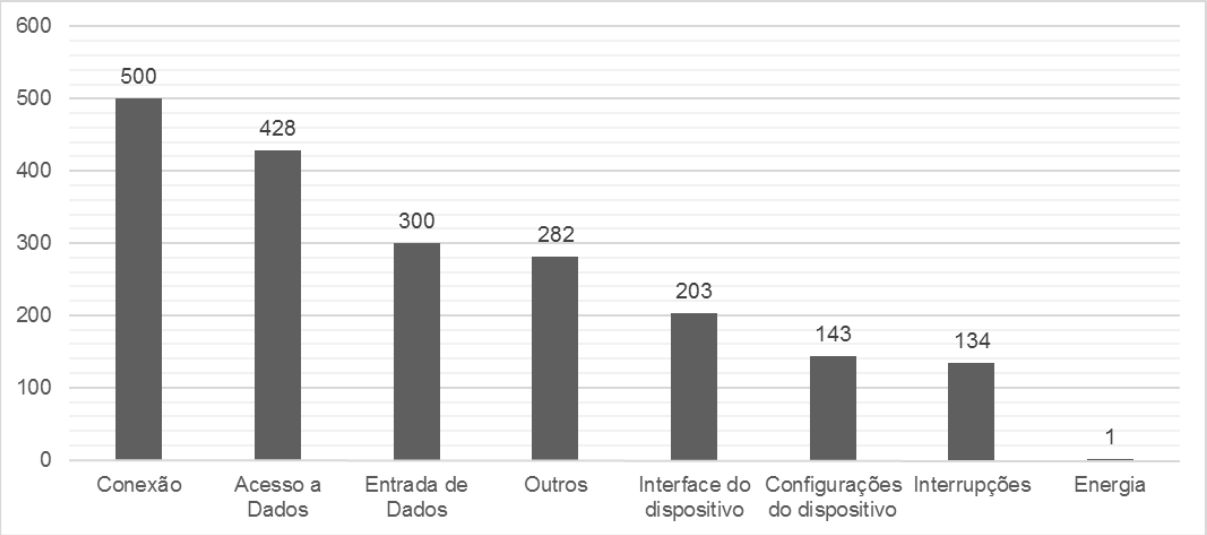


Figura 35. Distribuição das falhas por característica.

Observa-se que as características conexão (25,1% do total de falhas cobertas), acesso a dados (21,5% do total de falhas cobertas) e entrada de dados (15,1% do total de falhas cobertas) são as que mais possuem falhas reportadas, representando 61,7% das falhas cobertas pelo modelo proposto. Isso pode indicar que há uma grande utilização de tais recursos em aplicações móveis, o que, consequentemente, produz mais falhas. Interface do dispositivo (10,2% do total de falhas cobertas), configurações do dispositivo (7,2% do total de falhas cobertas) e interrupções (6,7% do total de falhas cobertas) correspondem a 24,1% das falhas cobertas pelo modelo.

Em contrapartida, a característica energia obteve apenas uma falha (0,1% do total de falhas) reportada, o que pode indicar que poucas falhas estão relacionadas a essa característica pelo fato das aplicações selecionadas não realizarem a consulta de nível de energia disponível no dispositivo. Contudo, isso pode indicar também que o poder de representação das falhas do modelo MOBPI-FM nessa característica pode ser baixo ou também que o aspecto de “consulta de nível de energia” abordado na concepção do modelo talvez não represente uma ação das aplicações escolhidas para estudo.

Tal informação indica que as características de conexão e acesso a dados são as que mais são utilizadas no contexto das aplicações móveis selecionadas, consequentemente possuem mais falhas associadas a elas. Isso pode indicar novamente a priorização de testes, específicos de conectividade para tal característica, e testes funcionais que verifiquem o acesso/manipulação a dados do dispositivo.

Houve falhas no estudo que são falhas específicas de aplicações móveis, mas que não foram cobertas pelo modelo (282 – 14,2%). Estas falhas que lidam com a limitação de memória do dispositivo e foram categorizas com o termo OUTROS. Como analisados na seção anterior, estas falhas são específicas de aplicações móveis, pois lidam com a limitação de recursos de memória do dispositivo. A quantidades de ocorrências dessas falhas reforçam um refinamento do modelo afim de incluí-las, e também reforçam a importância de tentar exercitá-las por meio de testes.

A Tabela 32 mostra o percentual de cobertura do modelo analisado por característica.

Tabela 32. Sumarização da cobertura de falhas por categoria de aplicação.

Característica	Falhas Específicas	Falhas cobertas	Falhas não cobertas	Cobertura
Conexão	500	500	0	25,1%
Acesso a Dados	428	428	0	21,5%
Entrada de Dados	300	300	0	15,1%
Interface do Dispositivo	203	203	0	10,2%
Configurações do Dispositivo	143	143	0	7,2%
Interrupções	134	134	0	6,7%
Energia	1	1	0	0,1%
Outros	282	0	282	14,2%

5.5. Considerações sobre o capítulo

Nesse capítulo foi apresentado um estudo de viabilidade com o propósito de verificar a cobertura do modelo MOBPI-FM em relação a falhas coletadas de aplicações móveis reais. Foi possível verificar que o modelo cobriu 86% das falhas relacionadas a características de aplicações móveis em um conjunto de 58 aplicações reais.

Verifica-se que a aplicação de uma metodologia sistemática para a definição de falhas a partir de uma metodologia sistemática conseguiu definir falhas representativas em um conjunto real de falhas de aplicações móveis. Tal afirmação não garante que o modelo seja robusto ou que seja completo, visto que houveram falhas específicas de aplicações móveis que não foram cobertas pelo modelo.

Com relação às falhas não cobertas, foi possível obter amostras de sua ocorrência nas falhas reportadas por desenvolvedores e verificar o comportamento que as aplicações

assumiam na ocorrência dessas falhas. Esses dados foram importantes para apontar o refinamento necessário do modelo MOBPI-FM.

Tal refinamento pode ser realizado de duas formas: incluindo essa característica no modelo, que pode ser chamada de **Interface com o SO** (sistema operacional), ou mesmo **Memória**, sendo definida como comportamento de uma aplicação móvel com a limitação de memória do dispositivo; ou unindo tal característica à de Energia, onde o resultado seria uma característica denominada **Limitação de Recursos**, onde no mapeamento do relacionamento da mesma com a aplicação, haveria um estado de definiria o comportamento da aplicação móvel mediante a escassez de memória.

Verificou-se também que, do conjunto de falhas reais obtidas, as falhas da característica conexão são as mais frequentes, mas que a falha mais encontrada foi a falha F-CN04, que reporta que uma aplicação móvel não consegue salvar dados no dispositivo. Verificou-se também

Como mostrado no Capítulo 2, testes para aplicações móveis necessitam de adaptações para serem aplicados no contexto de tal plataforma. Espera-se que as informações de cobertura obtidas nesse estudo possam auxiliar na escolha de critérios ou estratégias de teste para aplicações móveis, afim de contribuir com a prevenção dessas falhas específicas. Outro aspecto que se espera é a diminuição do esforço de teste, visto que o modelo fornece o conhecimento de falhas específicas e o estudo provê indícios de sua ocorrência. Outras contribuições do modelo são descritas no próximo capítulo, que apresenta as conclusões dessa pesquisa.

CAPÍTULO 6 – CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as conclusões obtidas nessa dissertação. Além da apresentação das principais contribuições, são apresentadas também algumas limitações da pesquisa realizada. Finalmente, alguns trabalhos futuros da pesquisa são delineados.

Teste para aplicações móveis são de grande interesse da comunidade de Engenharia de Software desde o surgimento do conceito. Há uma justificativa para isso, pois um dos elementos que podem influenciar na qualidade de um produto de aplicações móveis são os testes aplicados a ele, e nesse contexto, como discutido ao longo deste trabalho, testes baseados em falhas se apresentam como uma possível estratégia para a condução de testes em aplicações móveis, devido sua aplicação bem-sucedida em outras plataformas.

Este trabalho propõe um modelo de falhas denominado MOBPI-FM, gerado de forma sistemática e baseado em características específicas de aplicações móveis, independentemente da plataforma que tenha sido utilizada na sua construção. Esse apoio é provido a partir da disponibilização de falhas em três níveis de aplicação: o nível de falha, o nível de erros e o nível de defeitos. As principais contribuições desta pesquisa são descritas a seguir.

De acordo com o que foi descrito nessa dissertação, é possível confirmar a hipótese levantada na dissertação, apresentando o modelo MOBPI-FM (baseado totalmente em características específicas de aplicações móveis) e medindo a cobertura obtida por meio de um estudo de viabilidade, no qual verificou-se que o modelo cobriu 86% de um conjunto de falhas relacionadas a características específicas de aplicações móveis em um conjunto de 58 aplicações móveis reais.

6.1. Contribuições

Falhas baseadas em características de mobilidade. Um corpo de conhecimento pôde ser construído contendo a caracterização de 76 itens para o nível de falhas, e também itens para o nível de erro, onde esses níveis são independentes da plataforma móvel, e conforme a especificação da plataforma, um ou mais defeitos para cada erro. As falhas recebem classificações específicas conforme as características específicas da aplicação móvel. As falhas foram estruturadas por meio da aplicação da técnica de desvios HAZOP, que originou as falhas de forma sistemática. Tais itens estão diretamente ligados às características de aplicações móveis, fornecendo assim um corpo de conhecimento sobre possíveis erros de uma determinada aplicação móvel, ao utilizar os recursos do dispositivo e da plataforma.

Modelo descrito no nível de falha. Tais falhas foram avaliadas por meio de um estudo de viabilidade onde verificou-se que o mesmo é capaz de cobrir falhas reportadas em aplicações móveis reais. As falhas obtidas do modelo, aliadas aos dados empíricos obtidos do estudo de viabilidade, podem auxiliar na escolha de critérios de teste estruturais e baseados em falhas para a garantia de qualidade de aplicações móveis.

Modelo possível para erros e defeitos. Em tais níveis, o trabalho mostrou que o modelo pode compor os níveis de erro e defeito. A instanciação dos defeitos é dependente de plataforma, onde verificou-se que tais defeitos disparam erros, que produzem falhas em aplicações de exemplo da plataforma Android. Não foi possível avaliar empiricamente tais níveis por conta da falta de apoio ferramental para a injeção e detecção das falhas injetadas. Espera-se que, ao obter as informações sintáticas da plataforma, aliadas à futuras evidências empíricas, uma maior precisão na aplicação do critério de teste baseado em falhas para aplicações móveis.

6.2. Limitações da pesquisa

Entende-se que o trabalho possui algumas limitações, a principal delas diz respeito à comparação com outros trabalhos que relatam modelo de falhas e com a precisão das conclusões nos experimentos realizados. A falta de um ambiente automatizado para a inserção e execução das falhas foi o principal fator para uma não comparação.

As limitações deste trabalho estão relacionadas, principalmente, a três itens: (1) a não cobertura do modelo para falhas referentes ao consumo de memória, (2) validade do modelo nos níveis de erro e defeito (3) comparação com outros modelos de falhas.

Com relação às falhas específicas da característica de memória não cobertas pelo modelo, os dados obtidos no estudo de viabilidade foram importantes para apontar o refinamento necessário do modelo MOBPI-FM. Tal refinamento foi descrito no Capítulo 5 e será realizado seguindo um dos seguintes passos: incluindo essa característica no modelo, que pode ser chamada de Interface com o SO (sistema operacional), ou mesmo Memória; ou unindo tal característica à de Energia.

Entende-se a falta de dados empíricos sobre os níveis de erro e defeito como uma limitação desta pesquisa. Para acabar com essa limitação, primeiramente um ambiente automatizado será criado, no intuito de auxiliar na injeção dos defeitos, na execução para que a aplicação móvel acesse o erro gerado pelo defeito por meio da execução da aplicação, e fazer com que as falhas sejam apontadas na execução de testes.

Não foi possível comparar o modelo de falhas MOBPI-FM com a classificação definida por (HOLL *et al*., 2015), pois o mesmo foi planejado para ser uma abordagem que apoia a

inspeção formal. O modelo MOBPI-FM também pode apoiar inspeções formais em um futuro próximo, mas necessita de alguns refinamentos, tais como adequar os itens do modelo no nível de falhas em um formato específico (por exemplo *checklist*), bem como a adequação dos itens do modelo nos níveis de erro e defeito para verificação formal de código-fonte. Entende-se também que a avaliação proposta nessa dissertação possui limitações em virtude do baixo número de desenvolvedores utilizados na verificação das associações entre falhas de aplicações móveis reais e falhas do modelo. Contudo, pela experiência dos desenvolvedores envolvidos, acredita-se que a cobertura obtida no estudo esteja perto de um cenário real.

6.3. Trabalhos futuros

No futuro imediato, acredita-se que com as medidas descritas a seguir auxiliariam na avaliação empírica e no refinamento do modelo MOBPI-FM:

Incluir mais características específicas de mobilidade: Passa pela inclusão da característica específica de memória levantada no estudo de viabilidade. Plataformas móveis estão em constante evolução, ao ponto de dispositivos móveis comporem itens que usuários podem “vestir”. Tais dispositivos são denominados *wearables* (WILLIAMSON *et al*., 2015). Por definição, são dispositivos eletrônicos “vestíveis”. Relógios inteligentes, por exemplo, são uma categoria de *wearables* (WILLIAMSON *et al*., 2015). Além de um usuário ter a possibilidade de “vestir” um dispositivo móvel, o usuário que possui um automóvel pode acessar aplicações embarcadas em seus veículos. Companhias de tecnologia atualmente desenvolvem plataformas móveis para automóveis. Com tais plataformas é possível, entre outras funções, solicitar a exibição de mapas com trânsito em tempo real e a indicação de rotas, além de mudar a música sem tocar os painéis do carro (FLEMING, 2014). Com isso, tais evoluções na mobilidade dos dispositivos são acompanhadas de características específicas, que podem vir a originar falhas específicas.

Avaliações empíricas. Estudos empíricos devem ser executados para avaliar o quanto MOBPI-FM pode refletir com ainda mais precisão falhas ocorridas no ambiente móvel. Porém, antes de qualquer avaliação empírica, a limitação do apoio ferramental deve ser sanada. A princípio, três estudos podem ser realizados: (1) um estudo de observação com o objetivo de avaliar todas as instâncias de defeitos que determinada plataforma (a princípio na plataforma Android) pode gerar a partir do modelo, verificando também seu comportamento em relação a execução. Nessa dissertação, foram apresentadas instâncias de defeitos realizados na plataforma Android, para uma prova de conceito; (2) experimentos em laboratório comparando instâncias do modelo para diferentes plataformas de aplicações

móveis; (3) comparações empíricas com outros modelos de falhas. Alcançar o MOBPI-FM ideal será um processo iterativo de refinamento e avaliação.

Aprimoramento dos critérios e seleção de testes: Uma vez que o nível ideal de confiança no modelo seja estabelecido a partir de estudos empíricos, verificar se o mesmo pode ser usado para derivar critérios de adequação de testes para aplicações móveis.

REFERÊNCIAS

AGRAWAL, H.; DEMILLO, R. A.; HATHAWAY, B.; HSU, W.; HSU, W.; KRAUSER, E.W.; MARTIN, R. J. MATHUR, A. P. SPAFFORD, E. **Design of Mutant Operators for the C Programming Language Design of Mutant Operators for the C Programming Language**. Technical Report, p.1-74, West Lafayette, 1989.

AHMED, Z.; ZAHOOR, M.; YOUNAS, I. **Mutation operators for object-oriented systems: A survey.** Proceedings of International Conference on Computer and Automation Engineering (ICCAE), IEEE, v 2, p 614-618, 2010.

ARAUJO, R.F.; MALDONADO, J.C.; DELAMARO, M.E.; VINCENZI, A. M. R.; DELEBECQUE, F. **Devising mutant operators for dynamic systems models by applying the HAZOP study**. Proceedings of 6th International Conference on Software Engineering Advances, p. 58–64, 2011.

AVANCINI, A.; CECCATO, M. **Security testing of the communication among Android applications.** Proceedings of 8th International Workshop on Automation of Software Test, (AST), IEEE, p 57-63, 2013.

BAEKKEN, J. S. **A Fault Model for Pointcuts and Advice in AspectJ Programs**. M.Sc. Thesis, Washington State University, Washington, 2006.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. Experience factory. Book Chapter: **Encyclopedia of software engineering**. John Wiley & Sons, Inc., v. 1, p. 469–476, 1994.

BINDER, R. V. **Testing Object-Oriented Systems: Models, Patterns, and Tools**. 1º ed. Addison-Wesley Professional, 1999.

BIOLCHINI, J.; MIAN, P. G.; NATALI, A.; TRAVASSOS, G. H. **Systematic Review in Software Engineering**. Technical Report ES. System Engineering and Computer Science Department COPPE/UFRJ, v. 679, n. 05, p. 45, 2005.

DANTAS, V. L. L.; MARINHO, F. G.; DA COSTA, A.L.; ANDRADE, ROSSANA, M. C. **Testing requirements for mobile applications.** Proceedings of 24th International Symposium on Computer and Information Sciences (ISCIS), IEEE, p 555-560, 2009.

DELAHAYE, M.; DU BOUSQUET, L. **A comparison of mutation analysis tools for java.** Proceedings of the International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA), IEEE, p 187-195, 2013.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. 1º. ed. São Paulo: Elsevier, 2007.

DENG, L.; MIRZAEI, N.; AMMAANN, P.; OFFUUTT, J. **Towards Mutation Analysis of Android Apps**. Proceedings of 8th International Conference on Software Testing, Verification and Validation Workshops, p 1-10, 2015.

FLEMING, B. **Advances in Automotive Electronics [Automotive Electronics]**. IEEE Vehicular Technology Magazine, v. 9, n. 3, pp. 4-13, 2014.

HOLL, K.; ELBERZHAGER, F. **A Mobile-Specific Failure Classification and Its Usage to Focus Quality Assurance**. Proceedings of 40th Conference on Software Engineering and Advanced Applications (EUROMICRO), IEEE, pp. 385 - 388, 2014.

HOLL, K.; ELBERZHAGER, F.; VIEIRA, V. **Towards a Perspective-Based Usage of Mobile Failure Patterns to Focus Quality Assurance**. Book Chapter: Software Quality. Software and Systems Quality in Distributed and Mobile Environments. 1º. ed, Springer

International Publishing, 2015.

IBM. **Native, web or hybrid mobile-app development**. Thought Leadership White Paper, 2012.

IEEE STANDARDS ASSOCIATION. **Systems and software engineering — Vocabulary ISO/IEC/IEEE 24765:2010**. ISO/IEC/IEEE 24765:2010, v. 25021, p. 1–418, 2010.

JIA, Y.; HARMAN, M. **An analysis and survey of the development of mutation testing**. **I**EEE Transactions on Software Engineering, 37, 5 (September 2011), 649-678 2011.

JIANG, B.; LONG, X.; GAO, X.; ZHIFANG L.; CHAN, W.K. **FLOMA: Statistical fault localization for mobile embedded system**. Proceedings of 3rd International Conference on Advanced Computer Control, IEEE, pp. 296-400, 2011.

JOORABCHI, M. E.; MESBAH, A.; KRUCHTEN, P. **Real challenges in mobile app development.** Proocedins of International Symposium on Empirical Software Engineering and Measurement, IEEE, pp. 15-24, 2013.

JUST, R. **The major mutation framework: efficient and scalable mutation analysis for Java.** *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (ISSTA 2014). ACM, New York, NY, USA, 433-436, 2014.

KIM, S.-W.; CLARK, J.; MCDERMID, J. **The Rigorous Generation of Java Mutation Operators Using HAZOP.** Proceedings of the 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA), pp. 1-20, 1999.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. Technical Report. Keele University, London, 2007.

LECHETA, R. R. **Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 3th ed, Novatec, 2013.

LU, L.; HONG, Y.; HUANG, Y. **Activity page based functional test automation for android application.** Proceedings of the 3rd World Congress on Software Engineering (WCSE), IEEE, pp. 37-40, 2012.

MA, Y.-S.; OFFUTT, J.; KWON, Y.-R. **MuJava: a mutation system for Java.** *Proceedings of the 28th international conference on Software engineering* (ICSE '06). ACM, New York, NY, USA, pp. 827-830, 2006.

MAUL, W. A.; HEMMINGER, J.; OOSTDYK, R.; BIS, R. **A Generic Modeling Process to Support Functional Fault Model Development**, Technical Report, American Institute of Aeronautics and Astronautics, 2016.

MCDERMID, J. A.; PUMFREY, D. J. **A development of hazard analysis to aid software design.** Proceedings of IEEE 9th Annual Conference on Computer Assurance, IEEE, pp. 17 - 25, 1994.

MEIER, R. **Professional Android Application Development**. 1th ed. John Wiley and Sons, Inc., 2008.

MUCCINI, H.; FRANCESCO, A. DI; ESPOSITO, P. **Software testing of mobile applications: Challenges and future research directions**. *Proceedings of the 7th International Workshop on Automation of Software Test* (AST '12). IEEE Press, Piscataway, NJ, USA, pp. 29-35, 2012.

MYERS, G.; BADGETT, T.; SANDLER, C. **The Art of Software Testing**. 3th ed. John Wiley and Sons, Inc., 2012.

OFFUTT, J.; ALEXANDER, R.; WU, Y.; XIAO, Q. HUTCHINSON, C. **A fault model for**

subtype inheritance and polymorphism. Proceedings of 12th International Symposium on Software Reliability Engineering, IEEE, pp. 84-93, 2001.

PRESSMAN, R.S., MAXIM, B. R. **Software engineering: a practitioner's approach**. Palgrave Macgraw Hill, 2014.

REDMILL, F.; CHUDLEIGH, M. F.; CATMUR, J. R. **Principles underlying a guideline for applying HAZOP to programmable electronic systems**. Reliability Engineering & System Safety, Vol. 5, Issue 3, pp. 283–293, 1997.

SALVA, S.; ZAFIMIHARISOA, S. R. **Data vulnerability detection by security testing for Android applications**. Proceedins of Information Security for South Africa, IEEE, pp. 1 - 8, 2013.

SHULL, F.; SINGER, J.; SJOBERG, D. I. K. **Guide to Advanced Empirical Software Engineering**. London: Springer London, 2008.

SINGH, D.; SURI, B. **Mutation testing tools- an empirical study.** Proceedings of Third International Conference on Computational Intelligence and Information Technology (CIIT), IET, pp. 230 - 239, 2013.

SPÍNOLA, R. O.; DIAS-NETO, A. C.; TRAVASSOS, G. H. **Abordagem para Desenvolver Tecnologia de Software com Apoio de Estudos Secundários e Primários.** Proceedings of 5th Experimental Software Engineering Latin American Workshop (ESELAW), pp. 1-10, 2008.

STANDARD, B. **Hazard and Operability Studies Chemical Engineering Design**. Technical Report, London, 2012.

WASSERMAN, A. I. **Software Engineering Issues for Mobile Application Development**. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (FoSER '10). ACM, New York, NY, USA, 397-400, 2010.

WILLIAMSON, J.; QI L.; FENGLONG, L.; MORMAN, W. Kun L.; DICK, R.; LI, S. **Data sensing and analysis: Challenges for wearables.** Proceedings of 20th Asia and South Pacific Design Automation Conference, IEEE, pp. 136 - 141, 2015.

WINTER, S.; SARBU, C.; SURI, N.; MURPHY, B. **The impact of fault models on software robustness evaluations.** The impact of fault models on software robustness evaluations. In *Proceedings of the 33rd International Conference on Software Engineering* (ICSE '11). ACM, New York, NY, USA, 51-60, 2011.

WOHLIN, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A. **Experimentation in Software Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

ZHANG, P.; ELBAUM, S. **Amplifying tests to validate exception handling code**. Proceedings of the 34th International Conference on Software Engineering (ICSE '12). IEEE Press, Piscataway, NJ, USA, 595-605, 2014.

APÊNDICE I - PROTOCOLO DO MAPEAMENTO SISTEMÁTICO SOBRE TESTES ESTRUTURAIS E BASEADO EM FALHAS

PROTOCOLO

Este protocolo foi especificado seguindo as recomendações propostas nos trabalhos de (BIOLCHINI *et al*., 2005) e (KITCHENHAM e CHARTERS, 2007).

1) Objetivo

O objetivo deste estudo será esquematizado a partir do paradigma GQM (*goal, question, and metric*) descrito em (BASILI *et al*., 1994)

Tabela 33. Instanciação do paradigma GQM

Analisar	Abordagens, métodos ou técnicas de teste para aplicações móveis
Com o propósito de	Identificar
Com relação a	Abordagens, métodos ou técnicas estruturais e baseada em defeitos, nos níveis de unidade e integração
Do ponto de vista do	Pesquisador
No contexto	Pesquisas acadêmicas em Engenharia de Software

2) Formulação da Pergunta

Buscamos respostas para as perguntas a seguir:

- **[Q1]**: Como a abordagem, método ou técnica, ou a combinação delas foi aplicada no contexto móvel?
- **[Q2]**: Quais caraterísticas específicas da plataforma móvel a abordagem, método ou técnica conseguiu testar?
- **[Q3]**: Quais trabalhos utilizam a técnica de teste de mutação para testes em plataformas móveis?

3) Critério de Seleção de Fontes

Para as fontes de pesquisa foram adotados os critérios a seguir:

- Consulta de artigos em bibliotecas digitais, congressos, simpósios, revistas;
- Disponibilidade de consulta de artigos por meio da Web;
- Preferencialmente presença de mecanismos de busca por meio de palavras-chaves;
- Ter os estudos disponíveis em inglês ou português.

4) Métodos de Busca de Fontes

As fontes serão acessadas via web, por meio de máquinas de buscas. Portanto, no contexto desta revisão, não será considerada a busca manual.

5) Listagem de Fontes

As fontes serão bibliotecas digitais, disponíveis no portal CAPES, as quais estão listadas abaixo:

- IEEE Computer Science Digital Library: <http://ieeexplore.ieee.org>
- Scopus: <http://www.scopus.com>
- Science Direct - Elsevier: <http://www.sciencedirect.com/>

6) Idioma dos artigos

- **Inglês**, por ser o idioma utilizado pela maioria das editoras e também porque a maior parte dos trabalhos da área são publicados nesse idioma.

7) Critério de inclusão e exclusão de artigos

- Os artigos devem estar disponíveis na web;
- Os artigos pesquisados devem apresentar textos completos dos estudos em formato eletrônico;
- Os artigos devem estar descritos em inglês ou português;
- Os artigos devem contemplar temas relacionados testes para aplicações móveis;
- Serão excluídos os artigos que não fizerem referências aos testes para aplicações móveis.

8) Processo de seleção dos estudos primários

- O pesquisador executa a busca nas fontes selecionadas utilizando a *string* de busca elaborada.
- O conjunto de artigos é selecionado a partir da verificação dos critérios de inclusão e exclusão. Esta verificação se dará pela leitura do título do artigo, resumo e palavras-chave.
- Os artigos incluídos e excluídos são documentados no Formulário de Seleção de Estudos

9) Avaliação da qualidade dos estudos primários

Neste mapeamento, não será aplicado nenhum procedimento para avaliação da qualidade dos estudos pesquisados, visto que o mesmo tem o caráter de identificação.

10) Estratégia de extração da informação

Antes da primeira análise deverão ser eliminados os itens em duplicidade. Na primeira análise serão considerados os seguintes itens:

- **Título**

- **Resumo**
- **Palavras-Chave**

Em uma segunda análise serão extraídos os seguintes dados:

- **Título**: o título do trabalho;
- **Autor (es)**: o (s) autor (es) do trabalho;
- **Palavras-Chave**: palavras-chaves do trabalho;
- **Fonte**: onde o trabalho foi publicado;
- **Ano de publicação**: ano de publicação do trabalho;
- **Tipo de estudo**: por exemplo, estudo de caso, experimento, relatório de experiência, pesquisa de opinião, dentre outros;
- **Resumo**: uma breve descrição dos estudos;
- **Perspectivas futuras**: questão de pesquisa sugerida para trabalhos futuros;
- **Comentários**: observações gerais sobre o estudo;
- **Tipo de teste realizado**: funcional, estrutural ou baseado em erros;
- **Nível de teste** que se aplica (unidade, integração);
- **Ferramentas** usadas para modelagem;
- **Características da plataforma móvel** possivelmente alcançadas (funcionalidade, desempenho, segurança, dentre outras).

11) *String* de busca

As *strings* de busca foram definidas segundo o padrão PICO (*population, intervention, comparison, outcomes*) (KITCHENHAM e CHARTERS, 2007), conforme a estrutura abaixo:

Tabela 34. Definição do padrão PICO para a *string* do mapeamento

População	Aplicações móveis
Intervenção	Testes, nos tipos estruturais, funcionais e baseada em erros, nos níveis de unidade e integração
Comparação	Não se aplica
Resultados	Técnicas, abordagens, métodos, metodologias, ferramentas, processos

Com isso, a *string* de busca ficou a seguinte:

("mobile application" OR "mobile applications" OR "mobile app" OR "mobile apps" OR "smartphone applications" OR "smartphone application" OR "mobile device application" OR "mobile device applications" OR "device application" OR "device applications") ***AND*** *("structural testing" OR "structural test" OR "white-box testing" OR "white-box test" OR "mutation testing" OR "mutation test" OR "fault injection" OR "fault seeding" OR "unit testing" OR "unit test" OR "integration testing" OR "integration test" OR "automated test" OR "automated testing")* ***AND*** *("technique" OR "approach" OR "method" OR "procedure" OR "process" OR "tool")*

RESULTADO

Como especificado na seção anterior, em um primeiro momento foram analisados o título, o resumo e as palavras-chave de cada trabalho. Em um segundo momento, a análise foi feita procurando extrair os pontos descritos. Na Tabela, foram dispostos apenas o título do trabalho, seus autores e o ano de publicação.

Tabela 35 Resultado do mapeamento sistemático

Título	Autor	Ano
Amplifying tests to validate exception handling code: An extended study in the mobile application domain	Zhang, P. and Elbaum, S.	2014
AppACTS: Mobile App Automated Compatibility Testing Service	Jun-fei Huang	2014
Automated Generation of Oracles for Testing User-Interaction Features of Mobile Apps	Zaeem, R.N. and Prasad, M.R. and Khurshid, S.	2014
Automatic and scalable fault detection for mobile applications	Ravindranath, L. and Nath, S. and Padhye, J. and Balakrishnan, H.	2014
Automating UI tests for mobile applications with formal gesture descriptions	Hesenius, M. and Griebe, T. and Gries, S. and Gruhn, V.	2014
Caiipa: Automated large-scale mobile app testing through contextual fuzzing	Liang, C.-J.M. and Lane, N.D. and Brouwers, N. and Zhang, L. and Karlsson, B.F. and Liu, H. and Liu, Y. and Tang, J. and Shan, X. and Chandra, R. and Zhao, F.	2014
Efficiently, effectively detecting mobile app bugs with appDoctor	Hu, G. and Yuan, X. and Tang, Y. and Yang, J.	2014
Automated testing with targeted event sequence generation	Jensen, C.S. and Prasad, M.R. and MÃ¸ller, A.	2013
Considering context events in event-based testing of mobile applications	Amalfitano, D. and Fasolino, A.R. and Tramontana, P. and Amatucci, N.	2013
Data vulnerability detection by security testing for Android applications	Salva, S. and Zafimiharisoa, S.R.	2013

Mobile application testing-Challenges and solution approach through automation	Kirubakaran, B. and Karthikeyani, V.	2013
MZoltar: Automatic debugging of android applications	Machado, P. and Campos, J. and Abreu, R.	2013
Real Challenges in Mobile App Development	Joorabchi, M.E. and Mesbah, A. and Kruchten, P.	2013
Security testing of the communication among Android applications	Avancini, A. and Ceccato, M.	2013
A novel approach of automation testing on mobile devices	Nagowah, L. and Sowamber, G.	2012
A whitebox approach for automated security testing of Android applications on the cloud	Mahmood, R. and Esfahani, N. and Kacem, T. and Mirzaei, N. and Malek, S. and Stavrou, A.	2012
Activity page based functional test automation for android application	Lu, L. and Hong, Y. and Huang, Y. and Su, K. and Yan, Y.	2012
RoboLIFT: Engaging CS2 students with testable, automatically evaluated android applications	Allevato, A. and Edwards, S.H.	2012
Software testing of mobile applications: Challenges and future research directions	Muccini, H. and Di Francesco, A. and Esposito, P.	2012
Testing conformance of life cycle dependent properties of mobile applications	Franke, D. and Kowalewski, S. and Weise, C. and Prakobkosol, N.	2012
A GUI crawling-based technique for android mobile application testing	Amalfitano, D. and Fasolino, A.R. and Tramontana, P.	2011
A model-driven approach for automating mobile applications testing	Ridene, Y. and Barbier, F.	2011
Automating testing of service-oriented mobile applications with distributed knowledge and reasoning	Edmondson, J. and Gokhale, A. and Neema, S.	2011
FLOMA: Statistical fault localization for mobile embedded system	Jiang, B. and Long, X. and Gao, X. and Liu, Z. and Chan, W.K.	2011
Vision: Automated security validation of mobile apps at app markets	Gilbert, P. and Chun, B.-G. and Cox, L.P. and Jung, J.	2011
Adaptive random testing of mobile application	Liu, Z. and Gao, X. and Long, X.	2010
Hermes: A tool for testing mobile device applications	She, S. and Sivapalan, S. and Warren, I.	2009
Performance testing based on test-driven development for mobile applications	Kim, H. and Choi, B. and Yoon, S.	2009
Performance testing of mobile applications at the unit test level	Kim, H. and Choi, B. and Wong, W.E.	2009
Automated integration tests for mobile applications in Java 2 micro edition	Weiss, D. and Zduniak, M.	2007
MobileTest: A tool supporting automatic black box test for software on smart mobile devices	Jiang, B. and Long, X. and Gao, X.	2007
A strategy to perform coverage testing of mobile applications	Delamaro, M.E. and Vincenzi, A.M.R. and Maldonado, J.C.	2006