UNIVERSIDADE FEDERAL DO PAMPA

Marcus Vinicius Norberto Jácome

Framework para Teste de Desempenho em Dispositivos Móveis

Alegrete
2019

Marcus Vinicius Norberto Jácome

Framework para Teste de Desempenho em Dispositivos Móveis

Projeto de Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Dr. Maicon Bernardino da Silveira

Coorientador: Prof. Dr. Fábio Paulo Basso

Alegrete
2019

Marcus Vinicius Norberto Jácome

Framework para Teste de Desempenho em Dispositivos Móveis

Projeto de Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Projeto de Trabalho de Conclusão de Curso defendido e aprovado em ..... de ..... de .....
Banca examinadora:

Prof. Dr. Maicon Bernardino da Silveira
Orientador
UNIPAMPA

Prof. Dr. Fábio Paulo Basso
Coorientador
UNIPAMPA

Prof. Dr. Elder de Macedo Rodrigues
UNIPAMPA

Bel. Guilherme Legramante Martins
UNIPAMPA

AGRADECIMENTOS

Gostaria de agradecer primeiramente minha mãe Marleusa e meu pai Alexandre por me apoiar a todo momento durante a minha graduação, aos meus tios Marco e Meire, minha vó Maria Helena e aos meus irmãos por terem me acompanhado durante está jornada.

Agradeço ao meu orientador, professor Maicon Bernardino e ao meu coorientador Fábio Paulo Basso pela paciência , pelas oportunidades, pelas orientações e principalmente por terem me mostrado o "caminho das pedras".

Deixo também meus agradecimentos aos meus amigos Lukas, Giliardi, Jonnathan e Luiz Paulo por tornarem a vida melhor na colonia e por me proporcionar várias historias no qual vou rir bastante no futuro. Aos meus queridos amigos da 303 agradeço a todos os momentos que compartilhamos juntos durante esses últimos semestres.

Por fim gostaria de agradecer a todos os professores que tive durante a minha graduação e a todos funcionários e a comunidade da Unipampa.

RESUMO

Com a evolução tecnológica das redes sem fio e com o avanço tecnológico dos dispositivos móveis, os celulares estão cada vez mais presentes na vida das pessoas, tornando-se quase que indispensáveis em seu cotidiano. Assim, devido à popularização dos dispositivos móveis, os desenvolvedores de *software* estão empenhados em desenvolver aplicações confiáveis, corretas, seguras e performáticas, objetivando agradar seus usuários finais. Contudo, existe uma demanda alta no desenvolvimento de aplicações móveis, exigindo um grande esforço por parte dos desenvolvedores de *software*, para garantir a qualidade desses serviços. Uma das práticas que auxiliam a garantir um bom desempenho do Aplicativos Móveis (APPs) é por meio do teste de desempenho. No entanto, a literatura é limitada no sentido de fornecer uma visão geral e suporte para este tipo de abordagem. O objetivo deste estudo é oferecer um suporte para a realização de teste de desempenho em APPs, por meio da elaboração de um *framework*. Para embasar os conceitos relacionados ao estudo deste projeto, um protocolo foi formulado e executado de acordo com as diretrizes para a realização de um Mapeamento Multivocal da literaturas (MLMs) em Engenharia de *Software*. Este estudo apresenta um *framework* que oferece suporte na realização de teste de desempenho em APPs. Além disso identifica, por meio de um MLMs, ferramentas, *frameworks*, estratégias, abordagens, métodos e processos de testes de desempenho em APPs. Desta forma, vale destacar os resultados sobre classificação de métricas de desempenho, problemas reportados em testes de desempenho em APPs e o desenvolvimento de um *framework* para realizar teste de desempenho em APPs.

Palavras-chave: Teste de desempenho. Aplicativos móveis. *framework* para teste de desempenho.

ABSTRACT

With the technological evolution of wireless networks and the technological advancement of mobile devices, mobile phones are increasingly present in people's lives, becoming almost indispensable in their daily lives. Thus, due to the popularization of mobile devices, developers are committed to developing reliable, correct, secure and performative applications to please their end users. However, there is a high demand for mobile application development, requiring a lot of effort from developers to ensure the quality of these services. One of the practices that help ensure APPs performance is through the benchmarking approach. However, the literature is limited in providing an overview and support for this type of approach. The aim of this study is to provide better support for performing in APPs performance testing by proposing a framework. To support the concepts related to the study of this project, a protocol was formulated and executed according to the guidelines for the realization of an MLM in Software Engineering. This study presents a framework that supports in APPs performance testing. It also identifies, through a MLM, app performance testing tools, frameworks, strategies, approaches, methods and processes. Thus, it is worth highlighting the results on performance metrics classification, problems reported in APPs performance testing and the development of a framework for performing in APPs performance testing.

Key-words: Performance Testing. Mobile Application. framework for performance testing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ADB *Android Debug Bridge*

Android Sistema Operacional da *Google*

API *Application Programming Interface*

APK Android Application Pack

APP Aplicativo Móvel

CPU Unidade de Processamento Central

CQ Critérios de Qualidade

DE itens de extração de dados

EC Critérios de Exclusão

EFG modelos construídos através de gráficos de fluxo de eventos

ES Engenharia de *Software*

GB Giga Byte

GPS Sistema de Posicionamento Global

GQM *Goal Question Metric*

GUI Interface Gráfica do Usuário

IC Critérios de Inclusão

IDE Ambiente de Desenvolvimento Integrado

IOS Sistema Operacional do *Iphone*

KB Kilobyte

MB Megabyte

MLM Mapeamento Multivocal da literatura

RNF Requisitos Não Funcionais

RQ *Research Questions*

SMS *Systematic Mapping Study*

TaaS Teste com Serviço

TCC Trabalho de Conclusão de Curso

TCC1 Trabalho de Conclusão de Curso um

TCC2 Trabalho de Conclusão de Curso dois

SUMÁRIO

1 INTRODUÇÃO

Com a evolução tecnológica das redes sem fio e com o avanço tecnológico dos dispositivos móveis, os *smartphones* estão cada vez mais presentes na vida das pessoas, tornando-se quase que indispensáveis em seu cotidiano. Nos últimos anos, os dispositivos móveis passaram de *gadgets* de entretenimento para mídias populares, com uma ampla gama de usos, de aplicativos sociais a negócios, medicina e outros. O estudo de Bayindir e Winther (2018) demonstra um crescimento desde 2015, no uso de *smartphones*.

Assim, devido à popularização dos dispositivos móveis, os desenvolvedores estão empenhados em desenvolver aplicações confiáveis, corretas, seguras e com bom desempenho, objetivando agradar seus usuários finais. Contudo, existe uma demanda alta no desenvolvimento de APPs, exigindo um grande esforço por parte dos desenvolvedores, para garantir a qualidade desses serviços. Conforme mencionado por Muccini, Francesco e Esposito (2012), Wasserman (2010), Santos e Correia (2015), os APPs têm algumas características únicas que o diferem dos sistemas tradicionais, exigindo assim novas técnicas de teste ou abordagens modificadas para garantir eficácia e eficiência dos mesmos.

Para que tais sistemas obtenham o nível desejado de qualidade, os desenvolvedores tendem a utilizar técnicas de teste de software. Porém, existem limitações nas soluções manuais e automatizadas, as quais impedem uma abordagem mais abrangente, eficaz e prática para realizar testes automatizados nesse contexto. Portanto, quando os desenvolvedores adotam técnicas de testes automatizados em dispositivos móveis, consequentemente há aumento dos custos e do esforço empregado na produção dessas aplicações (JOORABCHI; MESBAH; KRUCHTEN, 2013) (KOCHHAR et al., 2015).

Diversas técnicas e tipos de testes podem ser empregados em aplicações móveis, inclusive em diferentes etapas de um processo de desenvolvimento de software. Todavia, de acordo com os resultados do estudo de Zein, Salleh e Grundy (2016), dentre os testes mais realizados em dispositivos móveis, destacam-se teste de usabilidade, teste de compatibilidade, teste de *interface*, teste de serviços, teste de segurança e teste de desempenho.

Nesse cenário, o teste de desempenho tem como objetivo verificar as metas especificadas de desempenho do sistema como por exemplo, tempo de execução, capacidade de resposta, uso de memória, consumo de energia, uso da rede ou métricas adicionais (AMALFITANO et al., 2013). Entretanto, aplicar esse tipo de teste não é uma tarefa trivial devido às particularidades deste tipo de aplicação, diferentemente de como é realizado o teste de desempenho em aplicações Web.

Há na literatura alguns relatos das principais dificuldades em aplicar teste no contexto dos dispositivos móveis. As taxas de tráfego na rede utilizada pelas aplicações podem influenciar no desempenho das mesmas (GOEL et al., 2013), bem como a diversidade de configurações existentes nos variados dispositivos móveis em que a aplicação se encontra instalada e as possíveis restrições de *hardware* (Malini et al., 2014) (NANDAKUMAR; EKAMBARAM; SHARMA, 2013). Outro fator importante é a interdependência

da aplicação com demais serviços externos, os quais também podem ter influência neste cenário.

1.1 Motivação

O estudo de Joorabchi e Kruchten (2013) aponta que a aplicação de testes em APPs não é uma preocupação recente dos desenvolvedores. Pois, diminui a chance da aplicação apresentar problemas antes da aplicação chegar ao usuário final.

Conforme os resultados apresentados da execução do MLM presente neste estudo, notou-se que existem vários estudos estudos que apresentam técnicas para ajudar os desenvolvedores a melhorar o teste em APPs. Porém há pouco suporte por parte da academia em testes de desempenho para APPs. Sahinoglu, Incki e Aktas (2015) diz que os problemas de desempenho serão de grande interesse em dispositivos incorporados com recursos restritos. Os autores também acreditam que o teste de desempenho pode fornecer oportunidades no campo de pesquisa de testes de APPs.

Liu e Cai (2014) afirmam que uma das limitações e problemas enfrentados pelos APP atualmente é a falta de suporte técnico para os testes de segurança, de desempenho e de compatibilidade. Pois, o atual desenvolvimento de tecnologias para teste de APP é relativamente lento, não tendo boas soluções para tais tipos de testes.

Apesar de já existirem técnicas e abordagens para teste de desempenho em APP, ainda não há muitos trabalhos que realizam estudos em cima dessa prática.

1.2 Objetivo

O objetivo geral deste estudo é desenvolver um *framework* capaz de oferecer suporte ao teste de desempenho em APPs. Para alcançar este objetivo, foram definidas outras metas. As metas são descritos com base no *template* do GQM proposto por Basili, Caldiera e Rombach (1994). O modelo é apresentado na Tabela 1, o qual apresenta uma estrutura que deve ser seguida para formular um objetivo (meta).

- **Meta 1 - Análise** do estado atual das pesquisas na área de testes de desempenho em APPs, **no intuito de** caracterizar o estado da arte deste tópico de pesquisa, **no que diz respeito ao** desempenho dos APPs **na perspectiva dos** pesquisadores e engenheiros de teste, **no contexto da** área acadêmica em computação.

- **Meta 2 - Análise** do estado atual da pratica na área de testes de desempenho em APPs, **no intuito de** caracterizar o estado da pratica deste tópico de pesquisa, **no que diz respeito ao** desempenho dos APPs **na perspectiva dos** pesquisadores e engenheiros de teste, **no contexto da** área comercial em computação.

- **Meta 3 - Análise** das dificuldades para a realização de teste de desempenho em APPs, **no intuito de** compreender os problemas que impossibilitam essa prática,

no que diz respeito a medição do desempenho dos APPs em um dispositivo móvel **na perspectiva dos** pesquisadores e engenheiros de teste, **no contexto da** área acadêmica em computação.

- **Meta 4 - Análise** das métricas coletadas durante o teste de desempenho em um APP, **no intuito de** caracterizá-los, **no que diz respeito ao** desempenho dos APPs **na perspectiva dos** pesquisadores e engenheiros de teste, **no contexto da** área acadêmica em computação.

Tabela 1 – *Template* de meta do paradigma GQM.

Analisar	Objeto sob análise
No intuito de	Finalidade.
No que tange a	Característica de qualidade do objeto.
Na perspectiva do	Pessoas interessadas.
No contexto do	Ambiente no qual a medida se encontra.

Fonte: Basili, Caldiera e Rombach (1994).

1.3 Contribuição

As principais contribuições deste trabalho estão relacionadas aos resultados obtidos com a condução do estudo empírico realizado e o desenvolvimento de um *framework*.

- Este estudo apresenta uma proposta de um solução que oferece suporte para testes de desempenho em APPs;
- Este estudo identifica, por meio de um *Systematic Mapping Study* (SMS), os principais problemas relacionados ao teste desempenho em APPs;
- Este estudo classifica, por meio de um MLM, as ferramentas e *frameworks* existentes que oferecem suporte a teste de desempenho em APPs;
- Este estudo apresenta uma categorização das principais métricas que devem ser coletadas durante um teste de desempenho em APPs.

1.4 Organização deste trabalho

O restante deste estudo esta organizado da seguinte forma. O Capítulo 2 apresenta a metodologia seguida para a produção deste estudo. O Capítulo 3 apresenta a fundamentação teórica do mesmo, na qual, são explanados os conceitos principais sobre teste de desempenho em APPs. O Capítulo 4 apresenta o MLM realizado para obter o entendimento abrangente da área de teste de desempenho em APPs. O Capítulo 5 mostra a proposta de solução para oferecer suporte para teste de desempenho em APPs.

O Capítulo 6 apresenta uma avaliação da proposta deste estudo. Por fim, o Capítulo 7 apresenta as considerações finais e trabalhos futuros deste estudo.

2 METODOLOGIA

Este capítulo aborda a metodologia adotada para a execução deste estudo. A Seção 2.1 apresenta e classifica qual o tipo de pesquisa que este estudo se enquadra. A Seção 2.2 apresenta todas as atividades foram seguidas para o TCC1 e para o TCC2.

2.1 Classificação de Pesquisa

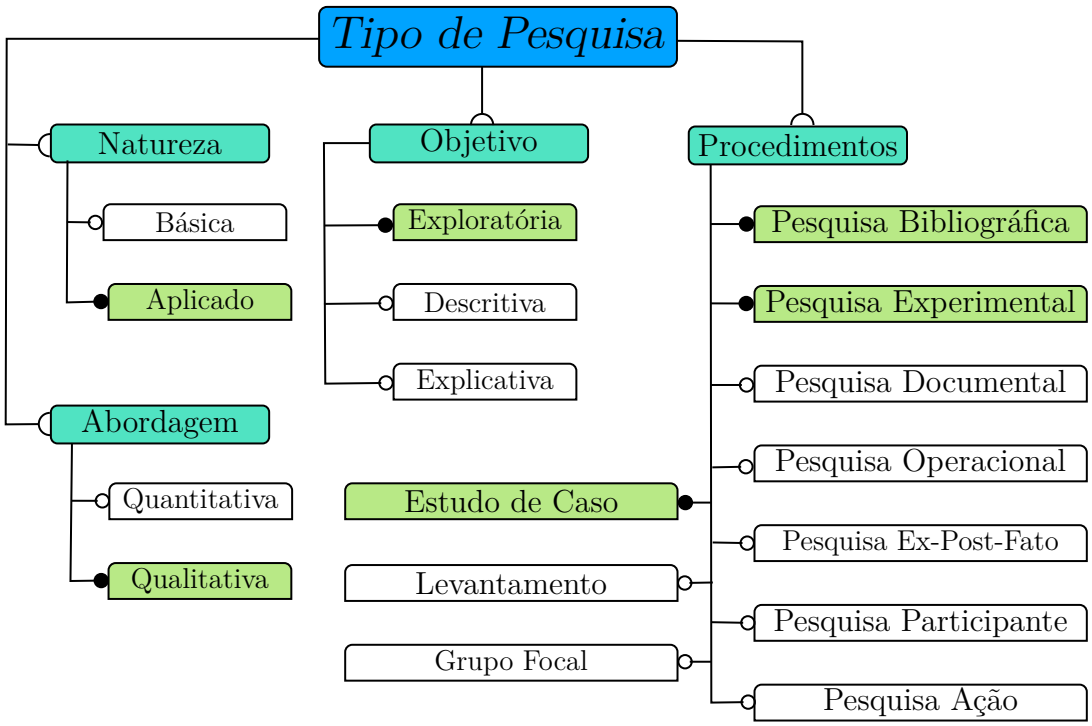
De acordo com Köche (2007), o planejamento de uma pesquisa depende tanto do problema a ser estudado, da sua natureza e situação espaço-temporal em que se encontra, quanto da natureza e nível de conhecimento do pesquisador. Isso indica que existe varias formas de classificar a pesquisa. Logo, é essencial classificar qual tipo de pesquisa se trata este estudo para guiar o planejamento do mesmo.

A Figura 1 identifica qual a classificação de pesquisa este estudo se enquadra mediante a sua natureza, objetivo, abordagem e procedimentos. Para Prodanov e Freitas (2013) nenhum tipo de pesquisa é autossuficiente, sendo necessário mesclar todos, acentuando um ou outro tipo.

Do ponto de vista da natureza de uma pesquisa, pode ser classificada em dois (2) tipos: básica e aplicada. Como este estudo tem o intuído de desenvolver um *framework* para oferecer suporte a testes de desempenho em APPs. Portanto, este estudo pode ser categorizado de natureza aplicada.

No que se refere ao objetivo, uma pesquisa pode ser classificada em três (3) ti-

Figura 1 – Tipos de pesquisa.



Autor: Adaptação de Prodanov e Freitas (2013).

pos: descritiva, exploratória e explicativa. Assim, este estudo pode ser classificado como exploratório, uma vez que encontra-se em sua fase de concepção.

No ponto de vista de abordagem, uma pesquisa pode ser classificada em dois (2) tipos: quantitativo e qualitativo. Este estudo esta classificado como abordagem qualitativa. Pois o foco da pesquisa é a qualidade (natureza e essência) do resultado final.

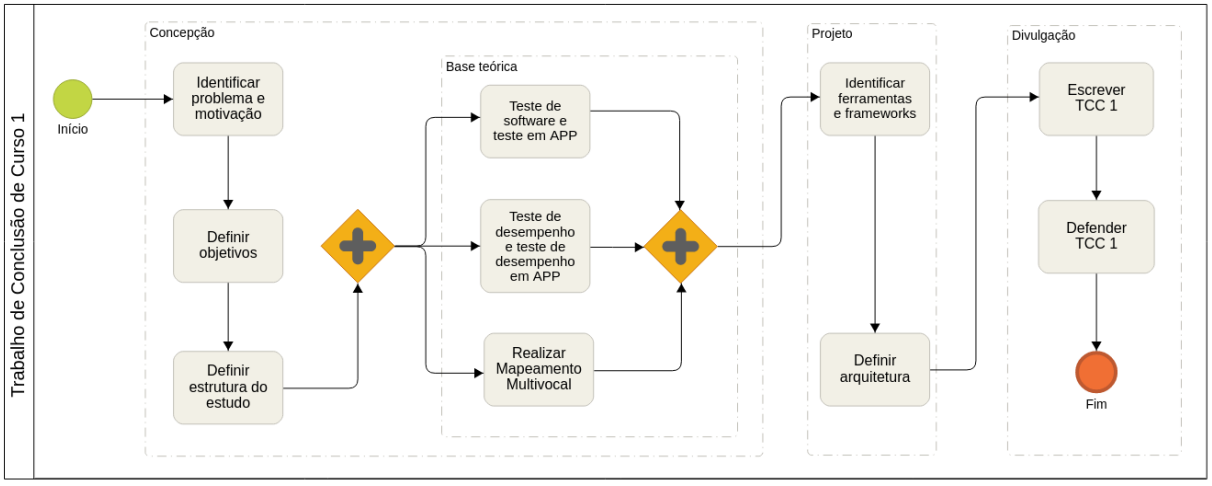
Tendo em vista os procedimentos, pode-se classificar em dez (10) tipos: pesquisa bibliográfica, pesquisa experimental, pesquisa documental, pesquisa operacional, pesquisa *ex-post-facto*, pesquisa participante, pesquisa-ação, grupo focal, levantamento e estudo de caso. Este estudo enquadra-se em três (3) classificações: pesquisa bibliografia, pesquisa experimental e estudo de caso. Pois, foram realizadas atividades para o levantamento da sua base teórica e o controle de variáveis para condução do MLM. Pretende-e realizar um estudo de caso da solução em um ambiante real para averiguar a relevância da mesma.

2.2 Desenho de Pesquisa

Para a condução deste estudo, foi definido um desenho de pesquisa para o TCC1 e TCC2, em que apresenta atividades bem definidas.

A Figura 2 apresenta todas as atividades seguidas para o TCC1. Primeiramente, é identificado os problemas e a motivação do estudo que é apresentado na Capítulo 1, com base nisso, os objetivos são definidos. Logo após, é realizado o planejamento do estudo como um todo.

Figura 2 – Desenho de pesquisa TCC1.



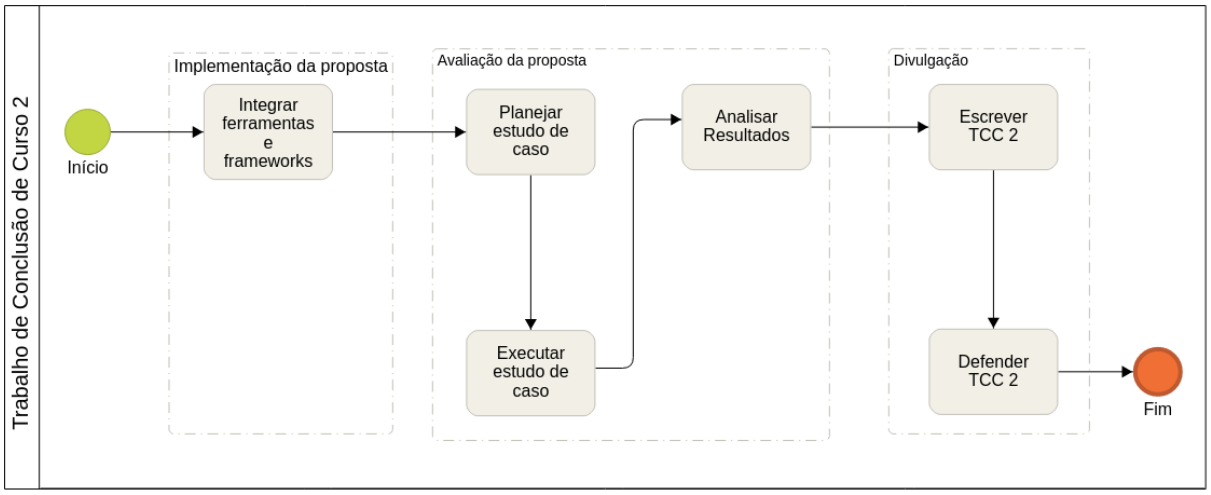
Fonte: O autor.

Em seguida, é desenvolvido toda a base teórica do estudo, como teste de *software* e teste em APPs, teste de desempenho e teste de desempenho em APPs, e por fim é realizado um Mapeamento Multivocal da Literatura, do inglês *Multivocal Literature Mapping* (MLM).

O próximo passo é a fase de projeto, na qual, foram identificados as ferramentas e *frameworks* para a proposta. Adiante defini-se a arquitetura da proposta do *framework*. Por fim, é realizada a escrita e defessa do Trabalho de Conclusão de Curso (TCC).

A Figura 3 apresenta todas as atividades que devem ser seguidas para a elaboração do TCC2. Primeiramente, é realizado a implementação da proposta, na qual, são integradas as ferramentas e *frameworks* escolhidos. A próxima atividade prevista, é à avaliação da proposta, na qual, será planejado um estudo de caso. Depois, o estudo de caso será executado e em seguida os resultados serão oletados e analisados. E por fim, sera realizado a entrega e defesa do TCC2.

Figura 3 – Desenho de pesquisa TCC2.



Fonte: O autor.

Para ajudar a compreender melhor os processos descritos nesta seção a Tabela 2 apresenta uma breve síntese deste trabalho, na qual os mesmos estão sendo executados.

Tabela 2 – Síntese deste trabalho.

Assunto	Teste de desempenho.
Tópico	Teste de desempenho em aplicativos móveis.
Questão de pesquisa	Como é realizado o teste de desempenho em aplicativos móveis?
Objetivo	Proposta de um *framework* para a realização de teste de desempenho em aplicativos móveis.

Fonte: O autor.

2.3 Lições do Capítulo

Neste capítulo foram expostas as atividades seguidas para o TCC1 e as atividades realizadas para o TCC2. Também é realizada a classificação da pesquisa em que este estudo se enquadra.

O próximo capítulo apresenta toda a fundamentação teórica necessária para uma melhor compreensão deste trabalho.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por finalidade apresentar os principais conceitos em que este trabalho está inserido e os trabalhos relacionados ao mesmo. A Seção 3.1 apresenta o conceito geral sobre testes em produtos de *software*. A Seção 3.2 mostra a importância e algumas dificuldades da realização teste em APPs. A Seção 3.3 mostra a diferença entre teste para APPs e para aplicações convencionais. A Seção 3.1.1 apresenta o conceito geral e conceitua teste de desempenho. A Seção 3.2.1 demonstra uma visão geral de teste de desempenho em dispositivos móveis. E por fim, a Seção 3.5 apresenta um resumo deste capítulo.

3.1 Teste de Software

Para Delamaro e Jino (2004) a construção de um *software* não é uma tarefa simples, dependendo do caso, pode se tornar extremamente complexa. Alguns fatores como características e dimensões do sistema podem aumentar ou diminuir a complexidade do desenvolvimento do software. Ainda seguindo a mesma linha do autor, a existência de um nível de dificuldade considerável no desenvolvimento de *software*, uma aplicação está sujeita a diversos tipos de problemas que acabam resultando na obtenção de um produto diferente daquele que se esperava. Esses problemas podem ser causados por diversos fatores, mas a maioria deles tem uma única origem: erro humano. O mesmo autor afirma que o desenvolvedor de um *software* depende principalmente da habilidade, da interpretação e da execução dos envolvidos no processo. Por isso, erros acabam surgindo, mesmo com a utilização de métodos, abordagens, técnicas e ferramentas de Engenharia de *Software* (ES).

Geralmente, esses problemas são chamados de "*bug*" de *software*. Conforme Patton (2000) um "*bug*" de *software* ocorre quando uma ou mais das cinco regras são verdadeiras:

1. O *software* não faz algo que a especificação do produto diga que deve fazer;
2. O software faz algo que a especificação do produto diz que não deveria fazer;
3. O software faz algo que a especificação do produto não menciona;
4. O software não faz algo que a especificação do produto não menciona, mas deve fazer;
5. O software é difícil de entender, difícil de usar ou lento, seja aos olhos do testador de software ou visto pelo usuário final dando a impressão que algo não está certo.

Delamaro e Jino (2004) afirmam que para tais *bugs* não sejam descobertos depois que o *software* for liberado para os usuários finais, é utilizada uma série de atividades da ES, chamadas de validação, verificação e teste. Estas atividades têm o objetivo de

garantir que o modelo pelo qual o software está sendo desenvolvido quanto o produto de software estejam em conformidade com o especificado.

De acordo com Myers, Sandler e Badgett (2011) o teste de software é um processo, ou uma série de processos, projetado para garantir que o código do computador faça o que foi projetado e, inversamente, que ele não faça nada não intencional, com o propósito de executar um programa com a intenção de encontrar erros. O processo referido, consiste em basicamente quatro etapas: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes.

Myers, Sandler e Badgett (2011) afirmam ainda que, em geral, é impraticável, quase que impossível, encontrar todos os erros em um programa. Para que se possa testar os produtos de *software*, deve-se adotar técnicas para se testar o mesmo. O mesmo autor afirma que as principais estratégias adotadas são testes de caixa preta e caixa branca.

A definição dada por Myers, Sandler e Badgett (2011) para teste de caixa preta e branca, é descrita abaixo:

Teste de caixa preta : Se preocupam apenas com as entradas e saídas do *software*, não sendo necessário saber o comportamento interno e a estrutura do programa;

Teste de caixa branca : Essa estratégia deriva os dados de teste de um exame da lógica do programa, ou seja, tem uma perspectiva mais interna do sistema (código fonte).

Com base nos testes de caixa preta e branca, somos capazes de executar a grande maioria dos tipos de testes. Para ajudar na execução dos testes de *software* Myers, Sandler e Badgett (2011) identificou um conjunto de dez (10) princípios ou diretrizes, vitais para o teste de *software*. Sendo elas:

1. Uma parte necessária de um caso de teste é uma definição da saída ou resultado esperado;
2. Um programador deve evitar tentar testar seu próprio programa;
3. Uma organização de programação não deve testar seus próprios programas;
4. Qualquer processo de teste deve incluir uma inspeção completa dos resultados de cada teste;
5. Os casos de teste devem ser escritos para condições de entrada inválidas e inesperadas, bem como para as que são válidas e esperadas;
6. Examinar um programa para ver se ele não faz o que deve fazer é apenas metade da batalha; a outra metade está vendo se o programa faz o que não deveria fazer;
7. Evite casos de teste descartáveis, a menos que o programa seja realmente um programa descartável;

8. Não planeje um esforço de teste sob a suposição tácita de que nenhum erro será encontrado;

9. A probabilidade da existência de mais erros em uma seção de um programa é proporcional ao número de erros já encontrados naquela seção;

10. O teste é uma tarefa extremamente criativa e intelectualmente desafiadora.

3.1.1 Teste de Desempenho

O desempenho é uma qualidade bem difundida dos sistemas de softwares, pois desde o software em si até suas camadas inferiores (como sistema operacional, *middleware*, *hardware* e redes de comunicação) podem afetar o desempenho Woodside, Franks e Petriu (2007). Um sistema com bom desempenho é aquele que permite ao usuário final realizar uma determinada tarefa sem atraso ou irritação percebida indevida Molyneaux (2009).

De acordo com Woodside, Franks e Petriu (2007), a Engenharia de Desempenho de *Software* é o campo responsável pelo estudo de atividades de Engenharia de *Software* e análises relacionadas usadas ao longo do ciclo de desenvolvimento de *software* e direcionadas aos requisitos de desempenho. Segundo estes autores, existem duas abordagens possíveis dentro da Engenharia de Desempenho de *Software*: baseada em medição ou em modelos. A abordagem de medição inclui atividades de teste, diagnóstico e otimização. Estas atividades são realizadas apenas no final do ciclo de desenvolvimento, visto que o sistema real precisa ser executado para que possa ser medido medido. Já a abordagem de modelos, usa os resultados quantitativos dos modelos para ajustar a arquitetura e o projeto com o objetivo de atingir os requisitos de desempenho.

Tendo conceituado essa área de estudo, pode-se entender melhor a definição de teste de desempenho dada por Meier et al. (2007). O autor diz que o teste de desempenho determina a capacidade de resposta (*responsiveness*), vazão (*throughput*), confiabilidade (*reliability*) e/ou escalabilidade (*scalability*) de um sistema sob uma dada carga (*workload*). Geralmente, o teste de desempenho tem o objetivo de:

- Definir as características de desempenho do sistema;
- Encontrar problemas de desempenho, *i.e.* gargalos (*bottlenecks*);
- Permitir a otimização do sistema;
- Estimar uma configuração de *hardware* adequada para a aplicação;
- Melhorar a escalabilidade e o desempenho do sistema.

De acordo com Meier et al. (2007) o teste de desempenho inclui as seguintes atividades:

1. **Identificação do ambiente de teste**: Identificação do ambiente de teste físico e o ambiente de produção, bem como as ferramentas e recursos disponíveis para a equipe de teste;
2. **Identificar os Critérios de Aceitação de Desempenho**: Identificação das metas e restrições de tempo de resposta, taxa de transferência e utilização de recursos. Além disso, identificar os critérios de sucesso do projeto que podem não ser capturados por essas metas e restrições;
3. **Planejar e Projetar os Testes**: Identificar os principais cenários, determine a variabilidade entre os usuários representativos e como simular essa variabilidade, defina os dados de teste e estabeleça as métricas a serem coletadas;
4. **Configurar o Ambiente de Teste**: Preparação do ambiente de teste, ferramentas e recursos necessários para executar cada estratégia, à medida que recursos e componentes se tornem disponíveis para teste;
5. **Implementar o Projeto de Teste**: Desenvolvimento dos testes de desempenho de acordo com o projeto de teste;
6. **Executar o Teste**: Execução e monitoramento dos testes;
7. **Analisar os Resultados, Reportar e o Retestar**: Consolidação e compartilhamento dos dados de resultados. Análise dos dados individualmente e como uma equipe multifuncional.

O teste de desempenho pode receber diferentes classificações de acordo com seus objetivos, como, teste de resistência, teste de pico e entre outros. A nomenclatura definida por Meier et al. (2007), é a seguinte:

- Teste de desempenho: determina ou valida a velocidade, a escalabilidade e a estabilidade do sistema sob teste. O desempenho está relacionado ao tempo de resposta, vazão e utilização. O autor ainda afirma que, o teste de desempenho é considerado como o superconjunto de todas as outras subcategorias de testes relacionados ao desempenho;
- Teste de carga: Subcategoria do teste de desempenho focada em avaliar o desempenho do sistema quando submetido a uma carga equivalente ao ambiente de produção;
- Teste de estresse: Subcategoria do teste de desempenho focada em avaliar o desempenho do sistema quando submetido a uma carga superior a de produção. Também pode avaliar o sistema sob condições estressantes como insuficiência de espaço em disco, limitação de memória, etc. Este tipo de teste é projetado para avaliar sob quais condições uma aplicação vai falhar, como será essa falha e quais indicadores podem ser monitorados para avisar que uma falha está prestes a ocorrer.

3.2 Teste em Aplicativos Móveis

A computação é uma área bastante dinâmica, visto que a mesma está em constantes mudanças. Em pouco tempo, o computador passou do *desktop* para o laptop e agora para o dispositivo móvel.

Um APP é vagamente definido como um aplicativo executado em dispositivos móveis e/ou recebendo informações contextuais de entrada Chen e Kotz (2000). Com a finalidade de melhorar esta definição, dois autores apresentam seus conceitos de APPs: Um APP é um programa baseado em rede que é executado em um dispositivo móvel Myers, Sandler e Badgett (2011). Um aplicativo é considerado móvel se for executado em um dispositivo eletrônico que pode ser movido (por exemplo, leitores de mp3, câmera digital, telefones celulares) Muccini, Francesco e Esposito (2012).

De acordo com Myers, Sandler e Badgett (2011) a maioria dos profissionais de teste de software considera o teste em APPs muito desafiador, mais do que qualquer outro tipo de software ou plataforma. Na realidade, são os dispositivos e o ambiente móvel, mais do que a aplicação móvel que impõe o desafio. Esses dois componentes adicionam muitas variáveis e complexidades que podem distorcer ou mascarar problemas em seu aplicativo, o que dificulta a criação de um plano de teste robusto. Resumidamente, é preciso considerar o desempenho e a confiabilidade da rede, *interfaces* de usuário consistentes, mudanças de contexto, diversidade de dispositivos e plataformas de recursos limitados.

Myers, Sandler e Badgett (2011) afirma que a chave para criar planos de teste bem-sucedidos para seus APPs é entender o ambiente de computação móvel. Contudo, o autor identificou várias áreas cruciais que devem ser exploradas e investigadas ao projetar planos de teste para APPs. A Tabela 3 mostra as áreas que devem ser exploradas segundo Myers.

3.2.1 Teste de Desempenho em Aplicativos Móveis

APPs, explicitamente ou implicitamente, devem satisfazer um número de Requisitos Não Funcionais (RNF) diferentes. Os RNF comuns incluem desempenho, compatibilidade, acessibilidade, usabilidade e segurança. A verificação de cada requisito não funcional requer o projeto de atividades de teste com objetivos específicos.

Nos APPs, o teste de desempenho tem como objetivo verificar as metas especificadas de desempenho do sistema, como tempo de execução, capacidade de resposta, uso de memória, consumo de energia, uso da rede ou metas adicionais Amalfitano et al. (2013).

Os testes de desempenho, em última instância, podem ser executados em ambientes reais por dispositivos reais para garantir a precisão das medidas de desempenho. No entanto, os testes de desempenho podem ser executados em ambiente simulado, tanto no nível do sistema quanto no nível de unidade Kim, Choi e Wong (2009). Porém, os testes de desempenho em ambientes simulados só permitem atividades como a comparação de

Tabela 3 – Considerações sobre *design* de teste de ambiente móvel.

Área	Comentários
Conectividade	Provisionamento de dispositivos. Velocidade de rede. Latência da rede. Disponibilidade de rede em áreas remotas. Confiabilidade do serviço.
Diversidade de dispositivos	Vários navegadores da web para testar. Várias versões de tempos de execução para Java ou outros línguas.
Restrições de dispositivos	Memória ou processador limitado. Tamanho de tela pequeno. Múltiplos Sistemas Operacionais. Capacidades de multitarefa. Tamanhos de cache de dados.
Dispositivos de entrada	Ecrãs tácteis. *Stylus* (Canetas touch screens). *Mouse*. Botões.
Instalação e Manutenção	Instalação e desinstalação. Patching. Atualização.

Fonte : Myers, Sandler e Badgett (2011)

alternativas de implementação, ineficiências na detecção de gargalos de desempenho no início do ciclo de vida.

Os problemas descritos na Seção 3.2, afetam também os testes de desempenho nos APPs. Pois, vários fatores influenciam esse tipo de teste, como o desempenho e confiabilidade da rede dos dispositivos, desempenho de *hardware* de diversos modelos e a plataforma na qual a aplicação será instalada. Nem toda aplicação móvel, é executada localmente, alguns de seus serviços podem ser fornecidos por outro serviço externo, deixando a aplicação dependente de outros serviços para seu funcionamento.

Os teste de desempenho em APPs devem levar todos os problemas citados anteriormente em consideração antes da realização dos teste. Para ajudar nesse contexto, Molyneaux (2009) definiu-se algumas etapas para executar casos de teste de desempenho, valendo tanto para sites quanto para dispositivos móveis e aplicativos para dispositivos móveis. As etapas são descritas a seguir:

1. Identifique os casos de uso e os requisitos de dados de teste;

2. Crie um nível de *Application Programming Interface* (API) e *scripts* no dispositivo. Isso pode ser possível usando o mesmo conjunto de ferramentas, mas suponha que você precisará de ferramentas de *scripts* diferentes;

3. Basear o desempenho do tempo de resposta dos *scripts* no dispositivo para um único usuário sem carga de segundo plano na seguinte sequência:

- Apenas no dispositivo. *Stub* e chamadas de API externas, conforme apropriado.
- Dispositivo com API ativa.

4. Crie um cenário de teste de desempenho de volume incorporando, sempre que possível, a automação em segundo plano e os *scripts* no nível da API;

5. Execute apenas os *scripts* no dispositivo;

6. Execute os *scripts* no nível da API simultaneamente com os *scripts* no dispositivo.

3.3 Teste em Aplicativos Móveis e suas Diferenças em Relação aos Testes *Desktop* e *Web*

O desenvolvimento de software teve sua origem com as tradicionais aplicações *desktop*, as quais se caracterizam pela execução direta em um computador, sem a necessidade de conexão com a Internet. Com a evolução da tecnologia e o surgimento da Internet, as aplicações *Web* ganharam mais evidência e, da mesma maneira, hoje ganham destaque os APPs.

Dependendo da plataforma em que uma aplicação é executada, pode-se classificá-la em três (3) categorias: *desktop*, *Web* e móvel. Essas plataformas diferem entre si, por características propiás. Conforme a plataforma, causam impactos direto no teste, e devendo ser levadas em consideração.

O objetivo do teste, independentemente da plataforma utilizada, é o mesmo: encontrar o maior número de problemas possíveis no *software*. Porém, devido às diferenças entre as plataformas, os testes são realizados especificamente de acordo com as necessidades e características da mesma.

Embora as aplicações *desktop*, *web* e móvel tenham suas similaridades, ainda apresentam diferenças. Por sua vez, os APPs são altamente orientados a eventos, aceitam tarefas dos usuários e mudanças no contexto ambiental, facilitado por um conjunto diversificado de sensores e componentes de *hardware* que fornecem *interfaces* para gestos baseados em toque, medições de temperatura, Sistema de Posicionamento Global (GPS) e entre outros serviços. Esses diversos cenários de entrada são difíceis de serem emulados pelos desenvolvedores em ambientes de teste controlados.

O estudo de Linares-Vásquez, Moran e Poshyvanyk (2017) apresenta sete (7) conjuntos de problemas abertos e distintos, servindo como fatores que impedem a criação de uma solução abrangente e automatizada para APPs, e por sua vez, que o diferem bem das demais aplicações *desktop* e *Web*. Sendo elas:

Fragmentação : Grande número de dispositivos com diferentes configurações, e para cada dispositivo há várias versões da mesma plataforma para o mesmo aparelho (*e.g*: Sistema Operacional da *Google* (Android) 2.0, 2.2, 4.0, 4.4, 8.1, 9.0 e Sistema

Operacional do *Iphone* (IOS) 7, 8, 9, 10, 11). Linares-Vásquez, Moran e Poshyvanyk (2017) afirmam que a fragmentação das plataformas causam uma complexidade para os testes em APPs. Pois, para cada configuração de dispositivo é necessário testar com mais de uma versão da plataforma;

Test Flakiness : Muitos APPs dependem muito de serviços de *back-end* para delegar operações que não podem ser executadas no dispositivo (por exemplo, cálculos e armazenamento longos) e acessar serviços de terceiros, como autenticação e geolocalização. Quanto maior a dependência, mais vulnerável o aplicativo a chance de ser afetado pela perda de serviço devido à falta de conectividade, tempos de resposta e problemas de integridade de dados quando grandes quantidades de dados são transferidos para um dispositivo. Assim, este é o chamado Fenômeno de *Test Flakiness* Luo et al. (2014);

Modelo de Falha Específico para Dispositivos Móveis e sua Aplicação : Os APPs utilizam um modelo de programação diferente de outras plataformas mais tradicionais, como a Web, principalmente devido a fatores específicos de dispositivos móveis, como eventos contextuais, comportamento orientado a eventos e interação orientada por gestos. Portanto, os APPs apresentam problemas diferentes das outras plataformas, em que sua maioria ocorrem *bugs* focados principalmente em desempenho e questões relacionadas à energia. O estudo de Linares-Vásquez et al. (2017) apresenta uma taxonomia de *bugs* para APPs;

Falta de Conhecimento Histórico em Casos de Teste : O reconhecimento de histórico em casos de teste para APPs são explorados principalmente de duas maneiras: modelos construídos através de gráficos de fluxo de eventos (EFG), ou modelos de linguagem. EFGs modelam o comportamento da Interface Gráfica do Usuário (GUI) como uma máquina de estados finitos vários estados da GUI (por exemplo, janelas / telas) com transições (entre os estados) definidos como eventos de entrada (por exemplo, clique no botão OK). No caso de modelos de linguagem, a memória é explicitamente modelada com distribuições de probabilidade condicionais que geram o próximo evento em uma sequência de teste com base em sua probabilidade condicionada à ocorrência dos eventos anteriores. Ambos, EFG e modelos de linguagem não são capazes de reconhecer recursos de alto nível (isto é, casos de uso) em casos de teste; os modelos geram sequências de eventos, mas sem reconhecer quais recursos ou casos de uso os eventos pertencem, portanto, os casos de teste gerados são sequências de eventos em vez de sequências de casos de uso. Além disso, o problema de desordem de teste descrito anteriormente torna a geração de um caso de teste ciente do histórico problemática;

Dificuldades em Evolução e Manutenção de *Scripts* e Modelos GUI : Por causa

da fragmentação das plataformas móveis, o testador precisa desenvolver vários *scripts* para cada modelo e versão de um aparelho. Quando ocorre mudanças na GUI do APP o testador deve alterar todos os *scripts* de teste, tornando-se uma atividade muito onerosa;

Ausência de Oráculos de Teste Específicos para Dispositivos Móveis : Um oráculo é um conjunto de condições ou resultados esperados que definem quando um caso de teste "falhou". As ferramentas e práticas atuais dependem fortemente de desenvolvedores e testadores para verificar manualmente os resultados esperados, e codificar manualmente os oráculos via assertivas ou exceções ao usar as APIs de automação;

Ausência de Suporte para Teste Automatizados com Múltiplos Focos : Para garantir aplicativos de alta qualidade que funcionem corretamente, são necessários testes com vários focos distintos, tais como: regressão, funcional, segurança, localização, energia, identificação de erros, teste de jogos e desempenho. Porém, atualmente não existe nenhuma ferramenta que dê suporte para todas essas metas.

3.4 Trabalhos Relacionados

É possível encontrar diversos trabalhos que fornecem *frameworks* como solução para aplicar testes em APPs. Porém, poucos focam em testar o desempenho da aplicação em um dispositivo móvel.

Como resultado da SMS discutido no Capítulo 4, foram encontrados quatro (4) estudos que propõem *frameworks* para realização de teste em APPs. No entanto, não foram desenvolvidos especificamente para teste de desempenho em APP, mas oferecem suporte para a coleta de dados de *hardware*, rede e tempo de execução.

O estudo de Malini et al. (2014) tem como objetivo superar os testes convencionais em APPs, em seu estudo, implementou-se uma estrutura de teste móvel como um serviço no ambiente de nuvem, chamada de MTAAS. Usando a estrutura do MTAAS, muitos APPs podem ser testados em diferentes dispositivos portáteis e diferentes plataformas móveis. O teste de APPs usando o MTAAS fornece resultados mais realistas, pois inclui a velocidade real da rede. Neste estudo é realizado um experimento na estrutura do MTAAS e os resultados dos testes mostram que o MTAAS pode reduzir efetivamente a complexidade dos testes móveis em diferentes dispositivos inteligentes.

No estudo de Rajan, Malini e Sundarakantham (2014), propõem um *framework* de testes, o "*Test My APP*", no intuito de testar os APPs *on-line*. A estrutura de teste proposta mede o tempo de resposta sob várias condições de dispositivo e rede. Os resultados obtidos são então processados usando o método de avaliação de desempenho baseado no método matemático em *chi-square* para fornecer uma medida de desempenho precisa do tempo de resposta.

Nandakumar, Ekambaram e Sharma (2013) apresenta o *Appstrument*, uma estrutura unificada para instrumentar o APPs para torná-los prontos para testes funcionais, de desempenho e de acessibilidade. Essa estrutura permite instrumentar o aplicativo para prepará-lo para uma única categoria de teste ou uma combinação de duas ou mais dessas categorias, com vários recursos opcionais para cada categoria. Além disso, dado um *script* de teste, a estrutura também suporta a reprodução automatizada de aplicativos instrumentados. O *Appstrument* foi implementado e testado em relação a alguns aplicativos populares do *Google Play* (aplicativos *Android*) e alguns aplicativos *iOS* internos da IBM. Os resultados indicam que essa estrutura é capaz de instrumentar com êxito um número considerável de aplicativos e reproduzir com eficiência casos de teste definidos pelo usuário automaticamente para coletar métricas ou resultados relevantes correspondentes a cada categoria de teste.

O estudo de Prathibhan et al. (2014) propõe uma estrutura de teste móvel no ambiente de nuvem que visa fornecer testes automatizados de APPs em vários dispositivos móveis. Essa estrutura de testes tem uma ferramenta de teste automatizada, a Ferramenta MAT (*Mobile Application Testing*) integrada a ela que executa testes funcionais, de desempenho e de compatibilidade de APPs.

Embora existam boas propostas de *frameworks* que apoiam o teste de desempenho em APPs, nenhum dos trabalhos citados informam se suas soluções são *open source* ou pagas. Também não apresentam nenhum *link* das ferramentas utilizadas que compõem suas soluções para que possam ser efetivamente avaliadas.

3.5 Lições do Capítulo

Neste capítulo é apresentada a fundamentação teórica deste trabalho. Esta fundamentação serviu como base de apoio às demais partes do trabalho. Os principais pontos abordados neste capítulo são:

- Visão geral de teste de *software*;
- Visão geral de teste em dispositivos móveis;
- Conceito de teste de desempenho;
- Conceito de teste de desempenho em dispositivos móveis;
- Principal diferença entre os testes em APPs com as demais tipos de software;
- Apresentação e discussão dos trabalhos relacionados com este estudo.

4 MAPEAMENTO MULTIVOCAL DA LITERATURA

Este capítulo descreve os procedimentos elaborados para a realização de uma investigação da literatura realizada neste estudo. Com este propósito a ser alcançado, foi executado um mapeamento multivocal da literatura, do inglês *Multivocal Literature Mapping* (MLM). Para isso, foram aplicadas diretrizes estabelecidas conceitualmente pelo autores Petersen et al. (2008) e Nakagawa et al. (2017).

A definição do protocolo do MLM descrita na Seção 4.1. A Seção 4.2 apresenta a o protocolo definido para executar a busca na literatura cinza. A Seção 6.2 mostra como foi realizado a condução da SMS deste estudo. A Seção 4.4 apresenta os resultados obtidos através da execução do MLM. A Seção 4.5 apresenta as ameças à validade deste estudo e como foram tratadas. Em seguida a Seção 4.6 mostrar o direcionamento das pesquisas para teste de desempenho em APPs. A Seção 4.7 apresenta os estudos similares a este MLM. A Seção 4.8 apresenta as considerações finais do MLM. E por fim Seção 4.9 descreve as lições aprendidas do MLM.

4.1 Protocolo do Mapeamento

Para se obter resultados confiáveis e reproduzíveis em um mapeamento, é essencial que um protocolo seja bem definido. Em linhas gerais, um protocolo visa definir todas as etapas e atividades necessárias para a condução do estudo. De acordo com Biolchini et al. (2005) e Nakagawa et al. (2017), as etapas de um protocolo devem contemplar a definição de questões de pesquisa, critérios de seleção, bibliotecas digitais a serem utilizadas para a identificação dos estudos, além de definir os critérios de qualidade e a estratégia de extração dos dados.

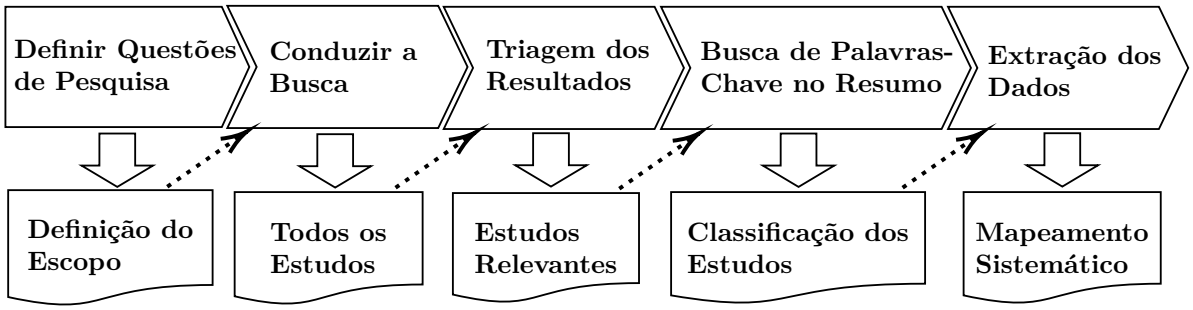
Um MLM é uma forma de Mapeamento Sistemático de Literatura, o qual inclui a literatura cinza. Os MLMs são úteis para os pesquisadores e profissionais, pois fornecem uma visão do estado da arte e da prática de uma determinada área.

Para executar o MLM, este estudo utilizou como referência o processo de SMS em Engenharia de Software de Petersen et al. (2008) e as diretrizes propostas por Garousi, Felderer e Mäntylä (2019) para pesquisa na literatura cinza. As entradas e saídas de cada etapa podem ser observadas na Figura 4 da SMS.

Em termos gerais, o estudo foi conduzido da seguinte forma:

- Identificação das necessidades de um MLM;
- Formulação do protocolo para o MLM;
- Execução do protocolo piloto;
- Execução da MLM.

Figura 4 – Representação do processo de mapeamento sistemático



Fonte: Petersen et al. (2008)

A motivação para realizar esse mapeamento sistemático surgiu a partir da identificação de poucas iniciativas para realizar este tipo de estudo. E as iniciativas existentes se limitam a ausência de um protocolo bem definido, além da grande parte dos estudos secundários focarem na identificação de estudos primários de testes em geral em APPs, não se atendo somente a testes de desempenho para APPs.

Em seguida, um estudo piloto foi executado para verificar e validar a qualidade do protocolo definido.

No estudo piloto, a *string* de busca foi utilizada nas seis (6) bibliotecas digitais, definidas na Tabela 4. Após a execução do piloto, foram identificados 6 (seis) estudos, sendo eles: Kim, Choi e Wong (2009), Kim, Choi e Yoon (2009) Nandakumar, Ekambaram e Sharma (2013), Amalfitano et al. (2013), Willocx, Vossaert e Naessens (2016), Zein, Salleh e Grundy (2016). Esses estudos foram discutidos com um especialista da área, e após aprovados foram definidos como obrigatórios para a execução do SMS. Em outras palavras, esses estudos, obrigatoriamente deveriam retornar nas *strings* de busca, além de serem incluídos ao final da execução do SMS.

Para realizar o estudo, adotou-se a estratégia de revisão em pares, definindo assim que todas as etapas de revisão deveriam ser executadas em pares. Também adotou-se uma fase de leitura do título, resumo e palavras-chave das obras selecionadas, incluindo os trabalhos promissores e eliminando os demais. Após essa etapa, realizou-se a análise da qualidade dos estudos remanescentes, a fim de medir a qualidade dos estudos primários. Finalmente, foi realizada a análise dos dados para geração dos gráficos e tabelas, então as questões de pesquisa foram respondidas.

Desta maneira levando em conta o propósito de um protocolo, foi utilizado o paradigma GQM de Basili, Caldiera e Rombach (1994) para executar esse conceito de forma mais estruturada. Assim, foi estabelecido uma Meta (**G**oal), e para esta meta foram descritas quatro (4) Questões (**Q**uestions) e suas respectivas Métricas (**M**etrics).

Para o protocolo deste estudo, foi utilizado a **Meta 1** descrita na Seção 1.2 como meta principal.

4.1.1 Questões de Pesquisa

Com o objetivo de atender a **Meta 1** as seguintes *Research Questions* (RQ) foram definidas:

RQ1. *Quais problemas são enfrentados durante o teste de desempenho em APPs?*

RQ2. *Quais ferramentas oferecem suporte para teste de desempenho em APPs?*

RQ3. *Quais processos ou frameworks foram definidos na literatura para apoiar o teste de desempenho em APPs?*

RQ4. *Quais são os tipos de métricas usados para medir o desempenho de um APP?*

4.1.2 Critérios de Inclusão e Exclusão

Com o objetivo de eliminar estudos primários não relacionados ao contexto delimitado, e apenas incluir estudos relevantes a esse mapeamento, foram definidos Critérios de Inclusão (IC) e Critérios de Exclusão (EC) para a etapa de seleção dos estudos primários.

IC01: *O estudo deve reportar, explicitamente, estudos sobre teste de desempenho em aplicativos móveis;*

IC02: *O estudo deve demonstrar alguma ferramenta/ técnica/ método/ métrica/framework processo para realizar teste de desempenho em aplicativos móveis;*

EC01: *Estudo com menos de 4 páginas, e.g. resumo expandido;*

EC02: *Estudo não disponível para download;*

EC03: *Estudo não escrito em inglês;*

4.1.3 Critérios de Qualidade

À qualidade de um SMS está diretamente ligado a relevância dos estudos primários selecionados para compor seus resultados. Assim, buscando elevar a qualidade dos resultados desse estudo, após a etapa de seleção final e considerando o paradigma GQM, em que as métricas devem ser usadas para responder às questões de pesquisa definidas, foram definidos quatro (4) Critérios de Qualidade (CQ) como métricas dentro do paradigma GQM. Desta forma, combina-se os resultados do controle de qualidade e o formulário de extração de dados para responder às questões de pesquisa da SMS.

Além disso, define-se uma pontuação para cada CQ e, depois de respondê-las, calcula-se a pontuação total de cada estudo.

Este estudo define duas (2) alternativas de respostas para cada critérios de qualidade:

A. Atende: *o estudo ganhará 1 ponto;*

N. Não Atende: *o estudo não pontuará (zero).*

Desta forma, a fim de apenas incluir estudos com qualidade ao SMS, definiu-se uma pontuação miníma de um (1) ponto, portanto, tornando assim os critérios de qualidade exclusivos. Assim, estudos que não alcançaram a pontuação miníma foram eliminados do SMS. As perguntas utilizadas para a avaliação dos critérios de qualidade e suas respectivas regras de pontuação foram definidas da seguinte maneira:

CQ1. *O estudo aborda alguma ferramenta que provê suporte para o teste de desempenho em APPs?*
N: O estudo não apresenta nenhuma ferramenta que tenha suporte para o teste de desempenho em APPs;
A: O estudo apresenta uma ou mais ferramentas que provêm teste de desempenho em APPs.

CQ2. *O estudo apresenta um processo ou framework o qual apoia o teste de desempenho em APPs?*
N: O estudo não aborda nenhum processo de apoio a teste de desempenho em APPs;
A: O estudo aborda um processo que apoia testes de desempenho em APPs.

CQ3. *O estudo apresenta alguma métrica para medição do desempenho dos dispositivos móveis ou APPs?*
N: O estudo não apresenta nenhuma métrica;
A: O estudo apresenta uma ou mais métricas.

CQ4. *O estudo relata problemas relacionados a testes de desempenho em APPs?*
N: O estudo não reporta nenhum problema enfrentado em testes de desempenho em APPs;
A: O estudo reporta um ou mais problemas enfrentados em testes de desempenho em APPs.

4.1.4 Critérios de Extração de Dados

Além dos critérios de qualidade, foram definidos itens de extração de dados (DE), sendo eles:

DE1. *Nome da ferramenta, framework, processo ou métrica.*

DE2. *Ferramentas de apoio: foram consideradas ferramentas propostas especificamente para apoiar uma abordagem ou outras ferramentas adaptadas ou utilizadas para uma tarefa durante a execução.*

DE3. *Problemas ou limitações enfrentadas.*

DE4. *Origem da proposta: comercial ou acadêmica.*

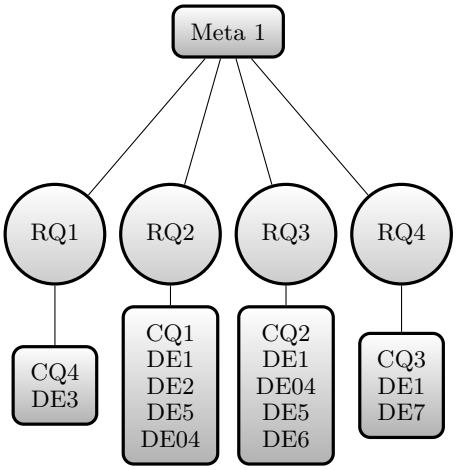
DE5. *Tipo de teste.*

DE6. *Tipo de contribuição: framework ou processo.*

DE7. *Intuito da métrica.*

Por fim, combina-se CQ e DE para definir às métricas com o propósito de responder as RQs. A Figura 5 apresenta o esquema destas associações com base na estrutura GQM proposta.

Figura 5 – Estrutura GQM



Fonte: O autor.

4.2 Pesquisa na Literatura Cinza

Com o objetivo de alcançar a **Meta 2** descrita na Seção 1.2, foi utilizado o motor de busca *Google*. Para isto, foi realizado combinações das *strings* de buscas definidas na Tabela 5 e adequando-as para o contexto da busca na *Web*.

- *Mobile Application Performance Testing*;
- *Testing Mobile Application Performance Tool*;
- *Mobile APP Performance Testing*;
- *Testing Mobile APP Performance Tool*;
- *Device Application Performance*;
- *Testing Device Application Performance Tool*;

- *Mobile Application Load Testing*;
- *Mobile Application Load Testing Tool*;
- *Stress Testing on Mobile Applications*;
- *Stress Testing on Mobile Applications Tool*.

A verificação dos resultados limitou-se cinco (5) páginas no motor de busca para cada *string* de busca. As aceitações das ferramentas ou *frameworks* no MLM é sujeita a um processo de seleção baseado em alguns requisitos. Primeiramente, a ferramenta ou *framework* deve estar disponível para acesso, para que possa ser avaliado suas funcionalidades. Necessita oferecer algum tipo de suporte para teste de desempenho em APP. Ao final da seleção, devem apresentar algumas informações como o tipos de teste oferecem, qual o tipo de monitoramento suportam e seu tipo de licença, para que possa ser feita sua categorização.

4.3 Condução do Mapeamento Multivocal

A condução da SMS deste estudo se baseou nas diretrizes de mapeamento sistemática da literatura em Engenharia de Software proposto por (PETERSEN et al., 2008).

4.3.1 Bibliotecas Digitais

Para encontrar os estudos primários foi realizado um processo de busca, no qual é necessário o uso de bases de dados online que (i) tenha um mecanismo de busca baseado na Web; (ii) tenha um mecanismo de busca capaz de usar palavras-chave, e; (iii) contenha estudos referentes a área da ciência da computação. As bibliotecas selecionadas e os seus tipo estão apresentadas na Tabela 4.

Tabela 4 – Bases de dados.

Base	Tipo	URL
ACM	Híbrida	<dl.acm.org>
Engineering Village	Motor de busca	<engineeringvillage.com>
IEEE Xplore	Base bibliográfica	<ieeexplore.ieee.org>
ScienceDirect	Base bibliográfica	<sciencedirect.com>
SpringerLink	Base bibliográfica	<link.springer.com>
Scopus	Motor de busca	<scopus.com>

Fonte: O autor.

Para a elaboração das palavras chaves utilizadas para a busca dos estudos primários relacionados ao estudo, foram definidas duas palavras-chave **Mobile Application** e **Performance Test**, na qual ambas foram definidas como termos principais, devido ao fato de refletirem melhor a população dos estudos desejados a sofrerem intervenção desse

SMS. Além disso, com o intuito de aumentar a abrangência desse estudo, buscando identificar a maior quantidade de estudos da área de testes de desempenho em dispositivos móveis, foram elaborados sinônimos derivados dos 2 (dois) termos principais, os quais podem ser observados na Tabela 5.

Tabela 5 – Definição das *strings* de busca.

Termos	**Sinônimos**
Mobile Application	Mobile App, Mobile Software, Device Application, Device App, Device Software
Performance Testing	Performance Test, Load Testing, Load Test, Stress Testing, Stress Test, Workload Testing, Workload Test

Fonte: O autor.

Na etapa da condução de um mapeamento sistemático são realizadas atividades que competem a triagem dos resultados, no qual as especificações descritas na Seção 4.1, são colocadas em prática com o intuito de identificar estudos a fim de responder as questões de pesquisa inicialmente definidas.

4.3.2 Busca Primária de Estudos

Para realizar a pesquisa utilizou-se as bibliotecas descritas na Tabela 4. Em cada uma das bibliotecas digitais foram utilizadas as palavras-chave e seus sinônimos, apresentados na Tabela 5, na qual retornou um conjunto inicial de 1414 estudos. Com o objetivo de aferir a qualidade da *string* de busca proposta, aplicou-se a abordagem denominada *Search-Based String Generation (SBSG)*, proposta por Souza et al. (2018), no qual se baseia no cálculo dos índices de precisão e sensibilidade. A precisão é obtida pelo cálculo do montante de estudos irrelevantes, por meio da razão entre os estudos retornados e a soma entre os estudos retornados e a diferença entre os estudos relevantes e os estudos retornados. Quanto maior o resultado, menor a chance da *string* retornar estudos irrelevantes ao estudo. Enquanto que a sensibilidade é alcançada por meio da razão dos estudos relevantes pela soma dos estudos retornados com a lista dos estudos piloto. Podendo assim indicar a possibilidade da *string* retornar estudos relacionados ao nosso contexto. Essa abordagem aplica uma técnica de Inteligência Artificial, por meio do algoritmo de *Hill Climbing* proposto por Russell e Norvig (2016), permitindo a aferição dos índices de precisão e sensibilidade para um conjunto de palavras-chave e um conjunto inicial de trabalhos selecionados. Neste contexto, a *string* proposta foi submetida com base em 6 (seis) estudos pré-selecionados (estudos piloto). Assim, os resultados alcançados foram 91,30% de precisão e 89,36% de sensibilidade,indicando que obtivesse uma alta precisão reforçada por uma alta sensibilidade, demonstrando que nossa *string* está bem formulada. De acordo com SOUZA et al. (SOUZA et al., 2018) os resultados passam a ser aceitáveis a parti dos percentuais de 80-99% de sensibilidade e 20-60% de precisão. Também vale

ressaltar que está técnica não garante a mesma chance de inclusão dos estudos em relação aos critérios de seleção.

Para cada uma das bibliotecas, as *strings* tiveram que sofrer ajustes, mas sempre mantendo todas as palavras-chaves, além de manter a coerência com as demais bibliotecas. As *strings* utilizadas para cada base, podem ser vistas na Tabela 6. As *strings* utilizadas nas bibliotecas *Engineering Village*, *Scopus* e *SpringerLink* possibilitaram o uso de um filtro para limitar as buscas para apenas estudos relacionados a área da Ciência da Computação. Os resultados são apresentados na Figura 6.

Tabela 6 – Strings de busca por base de dados.

Base	String de busca
ACM Digital Library	(“Mobile Application” “Mobile App” “Mobile Software” “Device Application” “Device App” “Device Software”) AND (“Performance Test” “Performance Testing” “Load Testing” “Load Test” “Stress Testing” “Stress Test” “Workload Testing” “Workload Test”)
Compendex	((("Mobile Application"OR "Mobile App"OR "Mobile Software"OR "Device Application"OR "Device App"OR "Device Software") AND ("Performance Test"OR "Performance Testing"OR "Load Testing"OR "Load Test"OR "Stress Testing"OR "Stress Test"OR "Workload Testing"OR "Workload Test")) WN ALL) + english WN LA
IEEE Xplore	("Mobile Application"OR "Mobile App"OR "Mobile Software"OR "Device Application"OR "Device App"OR "Device Software") AND ("Performance Test"OR "Performance Testing"OR "Load Testing"OR "Load Test"OR "Stress Testing"OR "Stress Test"OR "Workload Testing"OR "Workload Test")
Science Direct	("Mobile Application"OR "Mobile App"OR "Mobile Software"OR "Device Application"OR "Device App"OR "Device Software") AND ("Performance Testing"OR "Load Testing"OR "Stress Testing"OR "Workload Testing")
Springer Link	'("Mobile Application"OR "Mobile App"OR "Mobile Software"OR "Device Application"OR "Device App"OR "Device Software") AND ("Performance Test"OR "Performance Testing"OR "Load Testing"OR "Load Test"OR "Stress Testing"OR "Stress Test"OR "Workload Testing"OR "Workload Test")'
Scopus	TITLE-ABS-KEY ("Mobile Application"OR "Mobile App"OR "Mobile Software"OR "Device Application"OR "Device App"OR "Device Software") AND ("Performance Test"OR "Performance Testing"OR "Load Testing"OR "Load Test"OR "Stress Testing"OR "Stress Test"OR "Workload Testing"OR "Workload Test") AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (LANGUAGE , "English"))

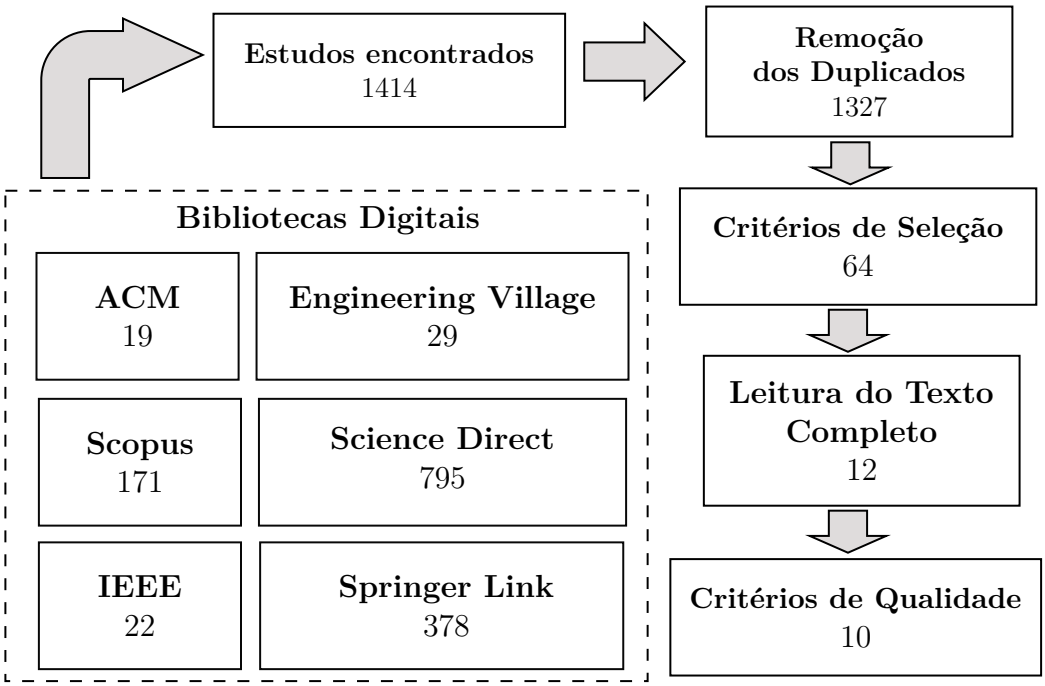
Fonte: O autor.

4.3.3 Seleção dos Estudos

Inicialmente, os estudos duplicados foram retirados restando 1327 estudos, em seguida cada revisor aplicou os critérios de seleção descritos na Subseção 4.1.2 separadamente obtendo assim 64 estudos. Os critérios de seleção foram aplicados em todos os estudos remanescentes da remoção das duplicatas a fim de eliminar todos os estudos que não estavam de acordo com os critérios de inclusão definidos. Por último uma leitura do texto completo foi realizada para incluindo os trabalhos promissores que estejam

de acordo com o assunto e eliminando os demais. O resultados dessa etapa podem ser avaliados na Figura 6.

Figura 6 – Representação do processo da seleção de estudos.



Fonte: O autor

4.3.3.1 Avaliação da Qualidade do Estudo

Após à aplicação dos critérios de seleção, os critérios de qualidade foram aplicados como objetivo de verificar o nível de relevância de cada estudo. Assim, como as etapas anteriores dessa condução, a avaliação dos critérios qualidade dos estudos primários foram realizadas separadamente, em que cada revisor realizou individualmente sua avaliação para cada estudo.

Em seguida calculou-se o coeficiente de *KAPPA* proposto por Cohen (1960), com o objetivo de medir a concordância entre os avaliadores nas notas determinadas para cada critério de qualidade. A Tabela 7 demonstra o número de ocorrências houveram concordâncias e discordâncias entre os avaliadores. Ao aplicar a fórmula e realizar o cálculo, obteve-se como resultado o valor para o coeficiente de *KAPPA* de 0,66. O que de acordo com os valores definidos por Landis e Koch (1977), para a interpretação do coeficiente obtido, os avaliadores se encontravam em nível de concordância substancial, o que é um bom sinal. Ainda de acordo com Landis e Koch (1977), em casos que resultado da interpretação é menor que 0,21, demonstra que os avaliadores que conduziram o estudo, não obtiveram o nível mínimo de concordância, sendo necessário refazer a etapa de condução. A Tabela 8 mostra como o coeficiente de *KAPPA* é interpretado.

Tabela 7 – Contingência dos resultados da avaliação de qualidade dos revisores.

Avaliador 1 \ Avaliador 2	Atende	Não Atende	Total
Atende	20	4	24
Não atende	4	20	24
Total	24	24	48

Fonte: O autor.

Tabela 8 – Interpretação do coeficiente de KAPPA.

Coeficiente KAPPA	Nível de Concordância
< 0	Não existe Concordância
0 – 0,20	Concordância Mínima
0,21 – 0,40	Concordância Razoável
0,41 – 0,60	Concordância Moderada
0,61 – 0,80	Concordância Substancial
0,81 – 1,00	Concordância Perfeita

Fonte: Landis e Koch (1977)

Após as avaliações de qualidade individuais, e a aplicação do coeficiente de *KAPPA*, nos casos que houveram divergências em relação às notas determinadas pelos avaliadores, um especialista foi consultado, agindo como mediador para a decisão sobre a nota final dos estudos divergentes. O resultado final é apresentado na Tabela 9. Porém, dois estudos não atingiram a nota miníma definida de um (1) ponto, portanto fora eliminados deste estudo, já que os itens de critério de qualidade também foram usamos como itens exclusivos.

Tabela 9 – Classificação geral dos artigos.

Classificação	Estudo	Nota
1	Malini et al. (2014)	4
2	Nandakumar, Ekambaram e Sharma (2013)	3
3	Prathibhan et al. (2014)	3
4	Liu, Hu e Cai (2014)	2
5	Rajan, Malini e Sundarakantham (2014)	2
6	Goel et al. (2013)	2
7	Kim, Choi e Wong (2009)	2
8	Kim, Choi e Yoon (2009)	2
9	Amalfitano et al. (2013)	1
10	Willocx, Vossaert e Naessens (2016)	1
11	Deka et al. (2017)	0
12	Kim (2012)	0

Fonte: O autor.

4.3.4 Busca na Literatura Cinza

A execução do protocolo na literatura cinza, retornou ao todo cinquenta e sete (57) ferramentas[1]. Ferramentas que não puderam ser vistas suas principais funcionalidades foram excluídas desta busca. Após a sumarização realizou-se a exclusão das ferramentas que não apresentação explicitamente algum suporte para teste de desempenho em APP, restando trinta e nove (39) ferramentas. Ao final os resultados foram categorizados.

4.4 Resultados e Discussões

Esta seção discute os resultados da pesquisa para responder as RQs apresentadas na Seção 4.1.1.

RQ1. *Quais os problemas enfrentados durante os testes de desempenho em dispositivos móveis?*

O intuito desta questão é responder a **Meta 3** descrito na Seção 1.2. Kim, Choi e Wong (2009) e Kim, Choi e Yoon (2009) reportam a dificuldade de aplicar testes de desempenho em APPs. Essa dificuldade é justificada ao fato de que esse tipo de teste exige um nível alto de processamento devido ao grande número de execuções ao mesmo tempo além do fato de que os dispositivos móveis, em sua grande maioria, possuírem recursos de *hardware* limitados. Outra dificuldade destacada pelo mesmo autor é que há uma dificuldade em encontrar o tempo de resposta de um método por meio de depuração ou teste convencionais.

De acordo com o estudo de Goel et al. (2013), um dos desafios enfrentados pelas plataformas de medição em redes móveis para medir o desempenho da rede do celular é como garantir capacidade de recursos suficientes para os casos de teste. O recurso limitador descrito são dados móveis, os quais na sua grande maioria são sujeitos a cobranças e limites mensais.

O estudo de Malini et al. (2014) confirma esse tipo de problema, no qual reafirma as dificuldades similares em medir o tráfego de rede dos dispositivos móveis.

Nandakumar, Ekambaram e Sharma (2013) assevera que a complexidade crescente dos APPs ocasiona um alto volume de casos de testes e que os mesmos têm ciclos de lançamento mais curtos e a frequência de atualização é alta, assim, tornando necessário que o testador realize testes adicionais com bastante frequência.

Os autores Malini et al. (2014), Nandakumar, Ekambaram e Sharma (2013) e Prathibhan et al. (2014) apresentam problemas similares. Assim, com a alta demanda por dispositivos móveis, atualmente existem diversos modelos e configurações de dispositivos, além de diversas plataformas. Desta forma, faz-se necessário o desenvolvimento de casos de testes de desempenho em diversos dispositivos implementados com diferentes

[1] Lista das ferramentas analisadas: <http://bit.ly/2WA1132>

tecnologias e arquiteturas. Sendo assim, muitas vezes é necessário alterar um único caso de teste para que o mesmo funcione em cada dispositivo.

Ainda nesse contexto com base no problema reportado por Willocx, Vossaert e Naessens (2016), o qual demostra a grande variedade de tecnologias de desenvolvimento disponíveis para o desenvolvimento de APPs, pode implicar na necessidade da realização de uma etapa de projeto anterior à especificação dos casos de teste. Essa etapa teria como intuito de verificar o escopo e definir a ferramenta que melhor se adéqua para o escopo identificado. A principal motivação para isso é que a ferramenta definida para a criação e execução dos casos de teste muitas vezes pode estar restrita com a tecnologia em que o aplicativo foi desenvolvido. Sendo assim, dependendo da escolha, poderá ser necessário a utilização de uma ferramenta específica para os testes de desempenho da aplicação.

RQ2. *Quais ferramentas oferecem suporte aos testes de desempenho em APPs?*

O objetivo desta questão é mapear as ferramentas utilizadas ou propostas por estudos científicos que suportam testes de desempenho em APPs.

Para responder essa questão de pesquisa foram encontradas seis (6) ferramentas que oferecem suporte a testes de desempenho em APPs. A Tabela 10 apresenta uma listagem das ferramentas encontradas, as ferramentas que servem de apoio para a ferramenta principal, podendo ser elas *plugins*, os tipos de teste suportados pelas mesmas e finalmente a sua origem, podendo ser ela acadêmica ou comercial.

Ao realizar uma análise das ferramentas encontradas foi possível perceber que atualmente existem poucas ferramentas para o suporte ao teste de desempenho em dispositivos móveis. Reafirmando uma necessidade já identificada por Zein, Salleh e Grundy (2016), o qual descreve a carência de ferramentas para a realização de testes de desempenho em APPs. Também pode ser identificado que em sua grande maioria as ferramentas

Tabela 10 – Lista das ferramentas.

Estudo	Nome	Ferramenta de Apoio	Tipo de Teste	Origem
KIM; CHOI; WONG	PJUnit	MobilePBDB e SUN WTK	Teste de desempenho	Acadêmica
GOEL et al.	MITATE	CPLEX	Teste de desempenho do tráfego de rede	Acadêmica
WILLOCX; VOSSAERT; NAESSENS	DDMS, ADB, Instruments tool, Visual Studio Console		Teste de desempenho	Acadêmica e Comercial
AMALFITANO et al.	The Monkey		Teste de estresse	Comercial

Fonte: O autor.

encontradas pela SMS são de origem acadêmica, resultado esse que já era esperado, devido ao *gap* existente entre o estado da arte e da prática. Fato de que muitas vezes as ferramentas de origem comercial não são publicadas em eventos científicos.

Além das ferramentas foi possível identificar também um *plugin* da plataforma Eclipse, denominado de PJunit, desenvolvido por Kim, Choi e Wong (2009). O seu principal objetivo foi fornecer suporte à testes de desempenho no nível de unidade.

Já em relação aos tipos de testes suportados por cada uma das ferramentas foi identificado que o principal foco está voltado a realização de testes de desempenho de uma forma geral, seguidas por testes de desempenho com foco no tráfego de rede apresentado por Goel et al. (2013) e testes de desempenho voltados à técnicas de testes de estresse, abordados por Amalfitano et al. (2013).

Por outro lado, a busca realizada na literatura cinza agregou mais valor a esta MLM, trazendo mais ferramentas para serem analisadas. A Tabela 13 apresenta todas as ferramentas encontradas. Percebe-se que a maioria das ferramentas possuem licença gratuita para uso equem em sua maioria oferecem suporte para teste de desempenho. Porem, nem todas oferecem algum tipo de monitoramento do APP sendo executado no dispositivo móvel.

RQ3. *Quais processos ou frameworks foram definidos na literatura para apoiar o teste de desempenho em APPs?*

O proposito desta RQ é encontrar na literatura *frameworks* e processos que ofereçam suporte em testes de desempenho em APPs. Por meio deste SMS foi possível identificar quatro (4) *frameworks* que realizam o apoio para testes de desempenho em APPs. Na Tabela 11 são apresentados os *frameworks* e seus tipos de testes suportados, além de sua origem, podendo ela ser comercial ou acadêmica.

Tabela 11 – Lista dos *frameworks* identificados.

		Tipo de Teste				
Estudo	**Nome**	**Nuvem**	**Desempenho**	**Funcional**	**Compatibilidade**	**Origem**
Malini et al. (2014)	MTAAS	✓	✓	✓		Acadêmica
Rajan, Malini e Sundarakantham (2014)	Test My App	✓	✓			Acadêmica
Nandakumar, Ekambaram e Sharma (2013)	AppStrument		✓	✓	✓	Comercial
Prathibhan et al. (2014)	ATaaS		✓	✓		Acadêmica

Fonte: O autor

Por meio dos estudos classificados foi possível identificar que a maioria dos *frameworks* possui foco no suporte nos tipos de testes de desempenho e testes funcionais.

Além disso é possível visualizar que os *frameworks* em sua grande maioria possuem focos distintos, sendo eles voltados a identificar o desempenho de aplicações por meio de Teste com Serviço (TaaS) em nuvem ou verificar o desempenho da compatibilidade.

Já em relação à existência de processos para o apoio de testes de desempenho em APPs não foram identificados processos que apoiassem esse tipo de teste de desempenho. Assim, demonstrando a possibilidade de estudos inéditos relacionados a esse tópico de pesquisa.

Por meio da busca na literatura cinza, diversos *frameworks* foram encontrados, os quais são apresentados na Tabela 13. A maioria dos *frameworks* encontrados oferecem TaaS, na qual, quase todas são pagas, com exceção a *openSTF*. Todos os *frameworks* oferecem algum tipo de suporte para monitoramento de desempenho enquanto são executado em um dispositivo móvel. Também nota-se que menos da metade estão preocupadas em medir o desempenho de um APP, estando focados em sua grande maioria em testes funcionais.

RQ4. *Quais os tipos de métricas necessários para medir o desempenho de uma aplicação móvel?*

O propósito desta questão é responder a **Meta 4** descrito na Seção 1.2.

Para responder essa questão de pesquisa foram identificados doze (12) tipos de métricas, as quais foram classificadas em três (3) categorias distintas. A Tabela 12 ilustra as categorias e suas métricas, assim como que cada estudo primário contempla cada uma das métricas.

As métricas classificadas no grupo `Execução` se caracterizam por serem métricas que quantificam o tempo na execução de atividades em APPs. Na literatura para esse tipo de medição são apresentadas métricas de execução do aplicativo, tempo de carregamento de cada página do aplicativo e o tempo de retorno a página anterior. Além de buscar medir o tempo de execução após o aplicativo já ter sido compilado, os estudos de Kim, Choi e Wong (2009) e Kim, Choi e Yoon (2009), preocuparam-se em medir o menor nível, ou seja, o teste unitário. No qual, medem o tempo em que cada função demora para finalizar a execução de cada unidade.

Já as métricas voltadas a medir o desempenho dos componentes físicos de um dispositivo móvel foram classificas em uma categoria denominada de `Hardware`. Nessa categoria foram alocadas as métricas referentes ao consumo de recursos físicos do dispositivo, tais como: *CPU*, memória, disco e o consumo de bateria. Para a medição do espaço ocupado no disco físico pelo aplicativo, o estudo de Willocx, Vossaert e Naessens (2016) realizou uma sub-categorização, a qual se preocupou em dividir e medir o espaço utilizado pelo aplicativo em duas (2) medidas. A primeira medida se preocupa em mensurar o tamanho do aplicativo em seu formato compilado, arquivos no formato Android

Tabela 12 – Categorização das métricas identificadas.

Estudo	Execução				Hardware					Rede		
	TdeEdoMetodo	TdeEdoAPP	TdeCarrPgn	Retonar Página	Uso de CPU	Mem. Alocada	Espaço Inst.	Tamanho APP	Consumo Bateria	Taxa de Trans.	Latência	Tempo Resp.
Kim, Choi e Wong (2009)	✓											
Kim, Choi e Yoon (2009)	✓											
Goel et al. (2013)												✓
Malini et al. (2014)												✓
Rajan, Malini e Sundarakantham (2014)												✓
Willocx, Vossaert e Naessens (2016)		✓	✓	✓	✓	✓	✓	✓	✓			✓
Nandakumar, Ekambaram e Sharma (2013)	✓				✓	✓					✓	
Prathibhan et al. (2014)										✓	✓	✓

Fonte: O autor.

Legenda: **TdeEdoMetodo**: Tempo de método. **TdeEdoAPP**: Tempo de execução do aplicativo. **TdeCarrPgn**: Tempo de carregamento da página. **Mem. Alocada**: Memória alocada. **Espaço Inst.**: Espaço de instalação. **Taxa de Trans.**: Taxa de Transferência. **Tempo Resp.**: Tempo de resposta.

Application Pack (APK). Já a segunda medida se ateve a medição do aplicativo quando o mesmo está instalado no disco físico do dispositivo móvel.

Finalmente, as métricas propostas para monitorar o tráfego de informações fornecidas por meios externos ao aplicativo também foram categorizados. Sendo as métricas de taxa de transferência, latência e tempo de resposta, as quais foram classificadas na categoria de `Rede`. Essas métricas são importantes em situações que se faz necessário obter informações referentes a quantidade de dados externos que estão sendo consumidos pelo aplicativo, como por exemplo atualizações ou consumo de informações de *Web services*, os quais são geralmente monitorados quando o aplicativo irá consumir dados de redes móveis. Além desta categorização foi possível identificar também que os estudos que propõem métricas para medir o desempenho do APPs, geralmente, preocupam-se em analisar categorias específicas, como por exemplo, os estudos de Kim, Choi e Wong (2009) e Kim, Choi e Yoon (2009) voltadas unicamente a testes unitários e os estudos de Goel et al. (2013), Malini et al. (2014), Prathibhan et al. (2014) e Rajan, Malini e Sundarakantham (2014) que focaram essencialmente em métricas na categoria `Rede`.

As exceções são os estudos de Willocx, Vossaert e Naessens (2016) e Nandakumar, Ekambaram e Sharma (2013). O primeiro estudo se preocupou em medir a execução do aplicativo e o *hardware*, enquanto que o segundo estudo contemplou parcialmente as três categorias classificadas nesse estudo.

Tabela 13 – Lista de Ferramentas e *frameworks* encontradas na literatura cinza.

#	Nome	Tipo de Teste			Tipo		Monitor		Licença	
		Nuvem (TaaS)	Desempenho	Funcional	*Framework*	Ferramenta	*Hardware*	Rede	Gratuita	Paga
1	TestFairy	✓		✓	✓		✓	✓		✓
2	Appium		✓	✓	✓		✓	✓	✓	
3	Robustest				✓		✓	✓		✓
4	Genymotion Cloud	✓	✓	✓	✓		✓	✓		✓
5	Pcloudy	✓		✓	✓		✓	✓		✓
6	Bitbar	✓		✓	✓		✓	✓		✓
7	TestObject	✓		✓	✓		✓	✓		✓
8	AWS Device Farm	✓		✓	✓		✓	✓		✓
9	Visual Studio Appcenter	✓	✓	✓	✓		✓	✓		✓
10	Jamosolutions	✓		✓	✓		✓	✓		✓
11	Test Studio Mobile	✓	✓	✓	✓		✓	✓		✓
12	Android Studio		✓	✓		✓	✓	✓	✓	
13	AppExperience	✓	✓	✓	✓		✓	✓		✓
14	XCUITest		✓	✓		✓	✓	✓	✓	
15	Tornimo	✓		✓	✓		✓	✓		✓
16	Robotium		✓	✓		✓			✓	
17	WebLOAD	✓		✓		✓	✓	✓		✓
18	Kobiton	✓		✓	✓		✓	✓		✓
19	Experitest	✓		✓	✓		✓	✓		✓
20	MonkeyTalk		✓	✓		✓			✓	
21	UI Automator		✓	✓		✓			✓	
22	Silk Mobile	✓		✓	✓		✓	✓		✓
23	SOASTA TouchTest	✓		✓	✓		✓	✓		✓
24	Testdroid (bitbar)	✓		✓	✓		✓	✓		✓
25	Espresso		✓	✓		✓			✓	
26	Firebase Performance Monitoring	✓	✓	✓	✓		✓	✓		✓
27	Blazemeter (molibe)	✓	✓	✓	✓		✓	✓		✓
28	Apteligent	✓	✓	✓	✓		✓	✓		✓
29	NeoLoad (mobile)	✓	✓	✓	✓		✓	✓		✓
30	Test Io	✓		✓	✓		✓	✓		✓
31	JMeter (mobile)		✓	✓		✓			✓	
32	Appvance	✓	✓		✓					✓
33	Dynatrace	✓		✓	✓		✓	✓		✓
34	Newrelic	✓		✓	✓			✓		✓
35	Firebase Test Lab	✓		✓	✓		✓	✓		✓
36	Load Mobile Application Servers	✓	✓		✓			✓		✓
37	OpenSTF	✓			✓			✓	✓	
38	Airtest IDE		✓	✓		✓			✓	
39	Airlab (airtest)	✓		✓	✓		✓	✓		✓

Fonte: O autor.

4.5 Ameaças à Validade do Estudo

Ameaças ao resultado do estudo foram identificadas na MLM realizada, as quais foram categorizadas conforme Hyman (1982) e Wohlin (2014):

Validade de Construção: Pode haver uma possível exclusão de estudos relevan-

tes. Para mitigar esse problema foram realizadas revisões em pares, em que as avaliações de cada etapa da condução e extração dos dados desse estudo foram realizadas por no mínimo dois avaliadores para evitar o viés de cada avaliador. Além disso, os avaliadores atuaram de forma independente durante o processo de condução e avaliação dos estudos a fim de evitar viés ao processo. Também foi utilizado varias bibliotecas digitais para realizar as buscas dos estudos.

Validade Interna: Possíveis ameaças internas podem ter surgido por meio do uso de métodos de busca incorretos, o que poderia levar à exclusão de estudos relevantes. Para mitigar esta ameaça o protocolo foi previamente avaliado por meio de um estudo piloto da SMS, a fim de verificar a relevância dos estudos incluídos nas buscas. Além disso, foi realizada a avaliação da *string* de busca usando os índices de precisão e sensibilidade propostos por Souza et al. (2018). As *strings* de busca definidas para a busca na literatura cinza foram testados para verificar se estavam retornando resultados coerentes com o esperado.

Validade Externa: Questões externas como a indisponibilidade de estudos foram resolvidas por pesquisas nos indexadores de literatura cinza, como o Google Acadêmico, Google, *ResearchGate* e contato com os autores.

Validade de Conclusão: Ameaças ao MLM podem surgir dos pesquisadores que realizaram este estudo, ao fato de que eles desconheciam seu viés durante a condução e extração dos dados desse estudo. Assim, podendo impactar em uma possível precisão na extração de dados, ameaçando a conclusão dos resultados do estudo. Para mitigar essas ameaças, um especialista na área foi consultado durante toda a condução do SLM. Além de realizar todos os estágios da condução, desde a seleção dos estudos até a extração dos dados foram realizados por no mínimo dois pesquisadores independentes, nos casos de divergências, sempre um especialista na área foi consultado.

4.6 Perspectivas de Pesquisa

De acordo com a análise realizada nos estudos presentes na literatura, relacionados ao contexto de teste de desempenho para dispositivos móveis, alguns tópicos emergem como possíveis perspectivas de pesquisa.

No estudo de Willocx, Vossaert e Naessens (2016) é evidenciado a necessidade de conduzir pesquisas que analisem o desempenho de um determinado APP produzido em diferentes plataformas.

No estudo de Kim, Choi e Yoon (2009) é sugerida a pesquisa na área de teste de desempenho, porém em um nível de granularidade menor, possibilitando o teste em nível unitário. Neste tipo de teste poderiam ser feitas as medições padrões de um teste de desempenho só que obtendo resultados para uma determinada funcionalidade ou método do APP.

Malini et al. (2014) apresenta uma perspectiva de pesquisa em que a técnica de

virtualização seja explorada para o teste de desempenho em APPs, seguindo o modelo utilizado para testes de desempenho Web.

De acordo com os resultados apresentados, não foi possível encontrar nenhum processo que de suporte para aplicação de teste de desempenho em APPs. Assim, possibilitando estudos inéditos nesta área de pesquisa.

Uma outra perspectiva de pesquisa a ser ressaltada como tendência, inspirada por outras áreas, há terceirização dos testes por meio de servidores na nuvem. Esta abordagem é conhecida como TaaS, em que algumas, ou todas, em que as atividades do teste de desempenho ficam a cargo de terceiros. Sendo assim, estudos que analisem desde a viabilidade, quanto ao impacto e viés que esse tipo de abordagem pode haver em relação ao teste de desempenho realizado de forma mais convencional.

4.7 Trabalhos Relacionados do Mapeamento Multivocal da Literatura

Esta seção apresenta os trabalhos relacionados a este estudo.

Santos Valéria L .L. Dantas (2012), em seu estudo realizou uma revisão sistemática da literatura, na qual identificou artigos que abordam métodos, abordagens, ferramentas e *frameworks* que apoiam os testes em APPs. Porém, a busca ficou restrita a um motor de busca apenas e algumas buscas manuais em trabalhos em congressos nacionais.

Os APPs de computação na nuvem estão se tornando amplamente utilizados para atenuar as limitações dos dispositivos móveis. A partir dessa motivação, o estudo de Al-Ahmad, Aljunid e Sani (2014) apresenta uma revisão sistemática da literatura sobre testes de aplicações para dispositivos móveis na nuvem, buscando identificar recursos e modelos para esse contexto. Como principais resultados, o autor discute os recursos e modelos de teste para computação móvel na nuvem em um nível conceitual, não mostrando nenhum exemplo prático.

Nos últimos anos, houve um aumento no interesse na área de teste para dispositivos móveis, incluindo teste de desempenho. Isso pode estar relacionado com o fato de que os APPs são diferentes das aplicações tradicionais como soluções Web e *desktop* e, cada vez mais, estão migrando para o uso em domínios críticos, exigindo uma abordagem diferente para a qualidade e a confiabilidade destes aplicativos.

O estudo de Zein, Salleh e Grundy (2016) executa um mapeamento sistemático para categorizar e estruturar as evidências de pesquisa que foram publicadas em técnicas de teste para APPs, bem como os desafios que eles relataram. Setenta e nove (79) estudos empíricos foram mapeados para um esquema de classificação. Diversas lacunas de pesquisa são identificadas e problemas específicos de teste para os profissionais da área são expostos. Porém, o estudo encontrou apenas dois trabalhos voltados a testes para dispositivos móveis, além de buscar categorizar de uma forma ampla os tipos de testes para aplicações em geral.

Braun, Elberzhager e Holl (2017) em seu estudo, apresenta um esquema de clas-

sificação de ferramentas para teste em dispositivos móveis em cinco categorias: testes funcionais, testes de desempenho e carga, teste de cobertura e teste de aceitação, testes em dispositivos em nuvem e ferramentas para análise e medição de testes com experiência do usuário. Para a classificação, 76 ferramentas de teste para APPs foram avaliadas e categorizadas de acordo com suas funcionalidades. Contudo, o estudo apenas se preocupou em realizar a classificação das ferramentas. Vale destacar que nenhum estudo empírico foi utilizado para avaliar cada uma das ferramentas identificadas. O estudo também identificou as ferramentas apenas na literatura cinza, o que não é uma má prática. Essa estratégia é amplamente utilizada no estudo da prática, além de servir como estudo complementar em estudos sistemáticos. No entanto, Nakagawa et al. (2017) descreve que apenas fundamentar os resultados apenas na literatura cinza pode colocar a reprodutividade do estudo em risco, sendo que a forma e o conteúdo que é apresentado podem sofrer alterações a qualquer momento, além de não se ter certeza da confiabilidade dos estudos encontrados no uso dessa prática.

A Tabela 14 apresenta um resumo dos trabalhos relacionados citados nesta seção.

Tabela 14 – Resumo dos trabalhos relacionados.

Conceito	**O estudo**	**Ismayle**	**Al-Ahmad**	**Zein**	**Braun**
Tipo	SLM	SLR	Revisão Ad-Hoc	SLM	Revisão Ad-Hoc
Intervalo	Até março de 2019	Não especificado	Não especificado	Até 2015	Não especificado
Identificar	Problemas, ferramentas, *frameworks*, processos e métricas	Processos, ferramentas, problemas e teste de usabilidade	Teste em nuvem	Técnicas para teste em APPs	Ferramentas para teste

Fonte: O autor.

4.8 Considerações Finais do Mapeamento Multivocal da Literatura

Com a realização desse mapeamento multivocal da literatura foi possível identificar as principais contribuições e lacunas existentes na literatura capazes de sugerir pesquisas futuras relacionadas aos principais problemas enfrentados durante a realização de testes de desempenho em APPs. Nesse estudo também é verificada a existência de *frameworks* que suportam os testes de desempenho em dispositivos móveis, bem como as ferramentas e as métricas utilizadas para elaborar e medir o desempenho desse tipo de aplicação. As contribuições presentes nesse estudo poderão ser utilizadas para beneficiar a indústria de *software* e a academia no apoio a tomada de decisões relacionadas ao que pode ser adotado para testar suas APPs, além de fornecer artefatos para o início de diversos outros estudos complementares.

As principais lacunas identificadas relacionadas aos problemas evidenciados, descrevem que a maioria dos problemas reportados na literatura, as quais não possuem

soluções triviais. Os problemas identificados geralmente estão relacionados à rápida evolução e mudança dos dispositivos móveis, exigindo assim que testadores de uma forma geral, realizem constantes refatorações em suas propostas, a fim de se adequarem as mudanças. Outro problema recorrente é a limitação de *hardware* da maioria dos dispositivos, em que muitas vezes para testes de desempenho se tornam um grande limitador, visto que esse tipo de teste exige constantes iterações no aplicativo a fim de identificar a aceitabilidade do aplicativo. Para a solução desses problemas, alguns dos autores buscaram resolver os impasses abordados. Contudo, buscaram elaborar suas próprias soluções para seus próprios problemas, havendo assim a necessidade do desenvolvimento de soluções sistemáticas que busquem resolver os problemas apresentados.

Acerca das ferramentas propostas na literatura para o apoio no teste de desempenho para APPs foram identificadas nove (9) ferramentas e um (1) *plugin* da plataforma Eclipse. As ferramentas identificas em geral buscaram contemplar testes de desempenho focados em alguma categoria do aplicativo, como por exemplo, testes de desempenho focados no tráfego de rede, ou testes de desempenho com foco em estressar o dispositivo.

Na literatura cinza foram encontradas nove (9) ferramentas. As ferramentas identificadas, em sua maioria são ferramentas mais robustas, como Ambiente de Desenvolvimento Integrados (IDEs). Não tem foco principal no teste de desempenho em APPs, sendo esse tipo de teste apenas uma de suas funcionalidades.

Em relação aos *frameworks* foram identificadas quatro (4) propostas pela SMS, o que em relação aos *frameworks* existentes para outros tipos de testes de desempenho em outros tipos de aplicação, como aplicações Web, o número atingido é bastante reduzido. Por outra lado, a busca pela literatura cinza se demostrou muito mais efetiva, retornando trinta (30) *frameworks*, mas em sua grande maioria são serviços pagos. Com os *frameworks* identificados foi descoberto que existem duas abordagens distintas que são geralmente utilizadas pelas propostas. Sendo elas voltadas a teste de desempenho TaaS ou abordagens voltadas a verificar a desempenho da compatibilidade das APPs.

Já em relação as métricas geralmente utilizadas para a realização das medidas de desempenho de APPs foi possível por meio desse estudo identificar e categorizar doze (12) métricas em três (3) categorias. Sendo elas, as categorias de Execução, *Hardware* e Rede. Ao categorizar cada uma das métricas foi possível identificar que nenhum estudo cobre todas as categorias em sua plenitude. Além disso, apenas dois (2) estudos realizaram medições em mais de uma categoria distinta. Portanto, é possível que estudos futuros busquem contemplar todas as categorias identificadas nesse estudo.

Como trabalho futuro será realizado o *snowballing* dos estudos já incluídos nesse mapeamento.

4.9 Lições do Capítulo

Este capítulo apresenta a condução do MLM para a obtenção do entendimento e os estado da arte da área de teste de desempenho em APP. Os principais pontos abordados neste capítulo são:

- Detalhamento do protocolo utilizado para execução do MLM;
- Execução do MLM;
- Apresentação e discussão dos resultados obtidos por meio do MLM;
- Apresentação de trabalhos relacionados que propõem mapeamentos semelhantes.

O próximo Capitulo apresenta a proposta do *framework* para teste de desempenho em APP.

5 PROPOSTA DE *FRAMEWORK*

Este capítulo apresenta proposta de um *framework* para teste de desempenho em APPs, bem como os requisitos que foram impostos para a tomada de decisões de projeto desta proposta. A Seção 5.1 mostra os requisitos de software que a solução deve atender. A Seção 5.2 apresenta as decisões de projeto tomadas para a escolha das ferramentas. A Seção 5.3 demonstra a arquitetura e o funcionamento geral da proposta. A Seção 5.4 apresenta o processo geral de realização de testes de desempenho no APP. A Seção 5.5 apresenta o projeto de desenvolvimento do *framework*. A Seção 5.6 apresenta um exemplo de utilização do *framework*. E por fim, a Seção 5.5 apresenta os pontos principais abordados neste capítulo.

5.1 Requisitos de Software

Esta seção lista os requisitos de software que foram definidos com base nos resultados do MLM, e também do conhecimento prévio dos pesquisadores envolvidos no projeto. Estes requisitos de software são relacionados diretamente com as decisões de projeto.

RQ1. A solução deve ser disponibilizada sob uma licença de código aberto. Como o foco é oferecer uma solução que dê suporte para teste de desempenho em APPs tanto para a academia e para os profissionais da área, a solução deve ser um *software* livre. Outro vantagem que esse requisito proporciona é sua possível evolução, em que a própria comunidade pode propor melhorias.

RQ2. A solução deve oferecer suporte para medição das métricas de *hardware*, rede e execução. Como resultado da execução do MLM foi possível categorizar os três (3) tipos de métricas essenciais em um teste de desempenho. Sendo assim, a solução deve ser capaz de medir os tipos de métricas para oferecer um suporte melhor para indústria e academia.

RQ3. A solução deve suportar a realização de teste de desempenho automatizado. Visto que a maioria das ferramentas encontradas pelo MLM são para testes automatizados, é desejável que esta solução oferecer este tipo de suporte.

RQ4. A solução deve suportar a realização de teste de desempenho em múltiplos dispositivos móveis, emulados ou reais. Um problema recorrente no teste em APP é a fragmentação das plataformas móveis, que por sua vez, afetaria os teste de desempenho em APP. Neste sentido, a solução deve minimizar o efeito deste problema.

RQ5. A solução deve oferecer TaaS. Levando em consideração a tendência TaaS notada pelos resultados do MLM, seria relevante atender essa tendência do mercado.

5.2 Decisões de Projeto

Esta seção lista as decisões de projeto tomadas para o desenvolvimento deste *framework* para que seja possível criar uma solução de código aberto que ofereça suporte para teste de desempenho em APPs.

DP1. A solução deve adotar apenas ferramentas e *frameworks* de código aberto no auxilio da implementação da proposta (RQ1). Os *frameworks openSTF*[1] e *Appium*[2] possuem código aberto. Uma de suas vantagens, é sua comunidade bem ativa, podendo facilitar a manutenção da solução.

DP2. A solução deve adotar ferramentas e *frameworks* que capturam dados de desempenho do APP em um dispositivo móvel, referentes às métricas de *hardware*, execução e rede (RQ2). O *framework Appium* atende a esse quesito, pois possui uma API[3] capaz de capturar os recursos do dispositivo móvel enquanto executa um *script* de teste, como por exemplo Unidade de Processamento Central (CPU), memoria principal, rede e bateria do aparelho. Além disso permite também a execução de comandos *shell script* que permite a coleta de dados de desempenho usando o *Android Debug Bridge* (ADB).

DP3. A solução deve adotar ferramentas e *frameworks* que ofereçam suporte para teste automatizado em múltiplos dispositivos de forma paralela (RQ3 e RQ4). Novamente o *framework Appium* foi adotado, pois o mesmo é capaz de realizar testes automatizados em APPs. O *framework openSTF* também ajuda neste quesito, pois oferece suporte para executar diversos dispositivos ao mesmo tempo podendo ser emulados ou físico, na qual o *Appium* consegue executar *scripts*. Além disso o *Open STF* possui uma API[4] que permite a conecção entre os dois *framework*.

DP4. A solução deve adotar ferramentas e *frameworks* que ofereçam suporte para TaaS (RQ5). O *framework openSTF* foi adotado pois é capaz de criar um serviço que contem vários dispositivos, recebendo apenas comandos do *Appium* para executar todos os *scipts* nos dispositivos conectados com o servidor.

5.2.1 *Appium*

O *Appium* é um framework para automação de testes para aplicativos mobile nativas ou hibridas tanto para dispositivos Android quanto IOS. Este framework utiliza a API *WebDriver* do *Selenium* para enviar os comandos de teste, por causa disso os

[1] OpenSTF: <https://openstf.io/>
[2] Appium: <http://appium.io/>
[3] API: <http://appium.io/docs/en/commands/device/performance-data/get-performance-data/>
[4] API OpenSTF: <https://github.com/openstf/stf/blob/master/doc/API.md>

testes podem ser escritos em diversas linguagens suportadas pelo *WebDrive*, incluindo Java, Objective-C, JavaScript, PHP, Python, Ruby, C#, Clojure, Perl, Haskell, e em alguns *frameworks* de testes como: JUnit, RSpec, PHPUnit, Nose, Mocha, Cucumber, Capybara, Vows e entre outros. Porem para este estudo optou-se por utilizar a linguagem java.

5.2.2 *OpenSTF*

O *OpenSTF* (ou *Smartphone Test Farm*) é um aplicativo web para depurar *smartphones*, *smartwatches* e outros *gadgets* remotamente, por meio do navegador. O *OpenSTF* se conecta com o dispositivos Android emulados ou dispositivos reais por meio de uma conexão com fio, podendo controlá-los e gerenciá-los. Um dos principais motivos para a adoção do *OpenSTF*, deve-se por causa de suas funcionalidades, pois permite criar uma "fazenda" de dispositivos móveis Android. Com esta "fazenda" de dispositivos, pode-se executar *scripts* de teste por meio do *Appium* em diversos dispositivos emulados ou reais que estejam conectados no serviço.

5.2.3 Android Debug Bridge

O ADB[5] é uma ferramenta de linha de comando versátil que permite a comunicação com um dispositivo. O comando ADB facilita uma variedade de ações do dispositivo, como instalar e depurar APP, e fornece acesso a um *shell Unix* que pode ser usado para executar diversos comandos em um dispositivo.

5.3 Arquitetura Geral

A arquitetura geral da estrutura do *framework* e os passos do processo para a realização dos teste de desempenho em APPs do mesmo é apresentado na Figura 7.

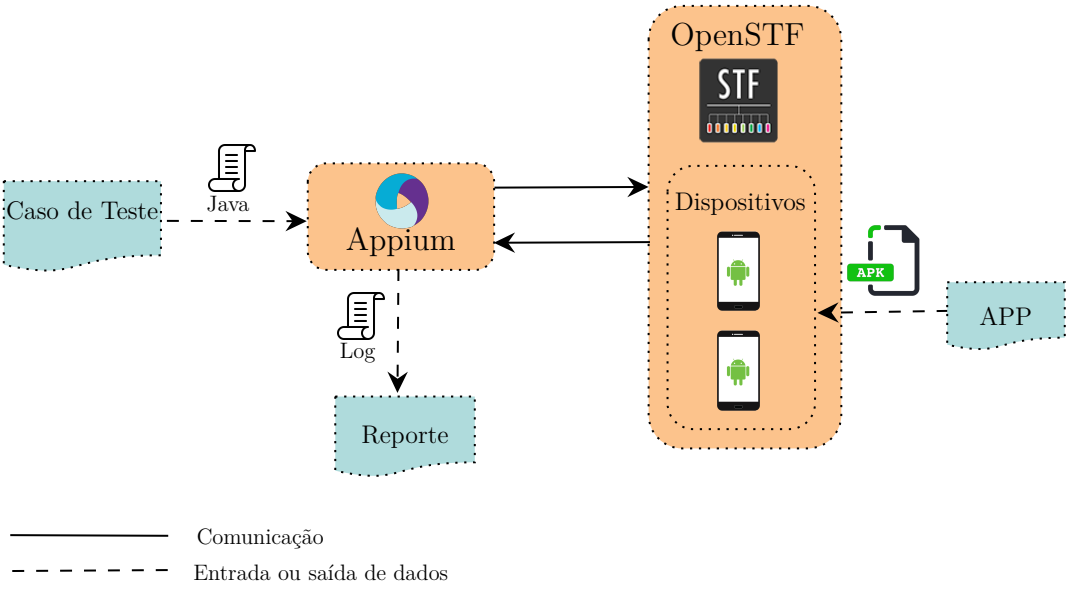
1. A primeira etapa é o *framework Appium*, o qual recebe como entrada um *script* de teste Java com todos os passos necessários que o dispositivo móvel deve executar. Em seguida, o *Appium* solicita ao *OpenSTF* acesso aos dispositivos que se deseja executar os teste de desempenho utilizando a API que foi desenvolvida para se comunicar com o *OpenSTF*;

2. Em seguida o *OpenSTF* enviar os dados necessários dos dispositivos para o *appium* executar os *scripts* de teste;

    - Enquanto o teste estiver sendo executado no dispositivo móvel, o *Appium* acompanha todo o processo de execução, enquanto coleta as métricas de desempenho de *hardware* e rede dos dispositivos.

[5] Conexao ADB: <https://developer.android.com/studio/command-line/adb?hl=pt-br>

Figura 7 – Arquitetura da proposta de *framework*.



Fonte: O autor.

3. Por fim, o *Appium* apresenta o *Log* com os dados coletados da execução do teste e o *status* da execução dos *scripts* em cada dispositivo.

5.4 Processo de Teste de Aplicativos Móveis com o *Framework* Proposto

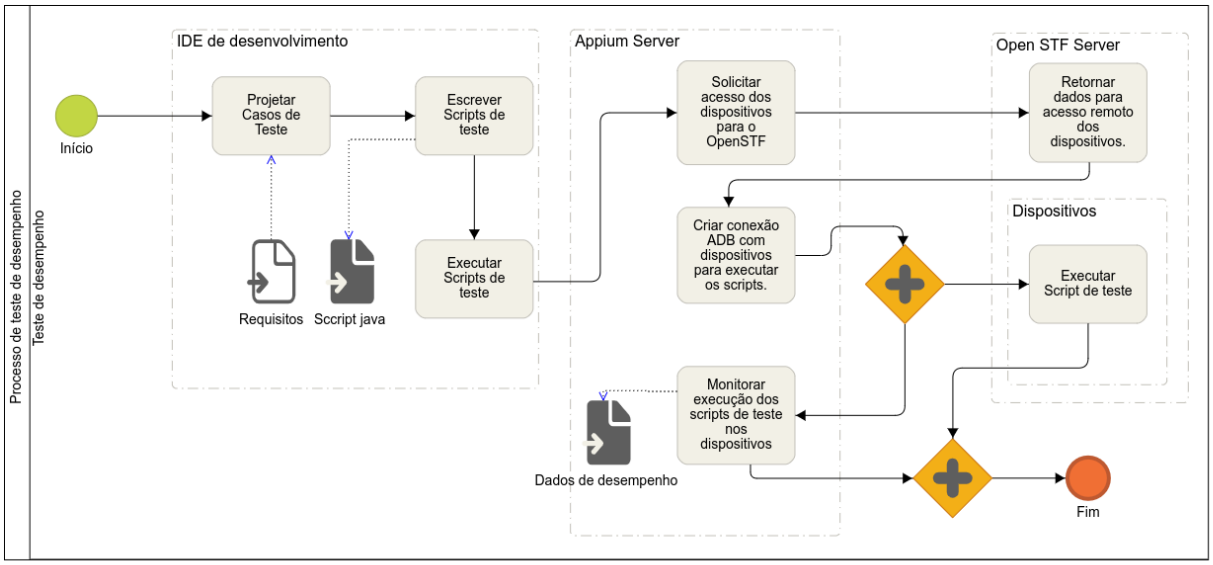
A Figura 8 apresenta o funcionamento do processo de teste de APPs com o *framework* proposto, o qual é dividido em oito (8) etapas. Antes de iniciar o processo de execução dos testes, o engenheiro de teste criará o projeto *maven* [6] em uma IDE e realizar toda a configuração necessária para que o *Appium* e o *OpenSTF* trabalhem corretamente.

Inicialmente, o engenheiro de teste deve projetar os casos de teste para que possam ser convertidos em *scripts* de teste na linguagem Java. A próxima atividade é a execução dos *scripts* de teste nos dispositivos móveis, na qual devem ser disparados pela IDE que estiver sendo utilizada.

Assim que o *Appium* receber os *scripts* de teste, solicitará ao *OpenSTF* acesso aos dispositivos que deseja executar o teste. Em seguida o *OpenSTF* dará acesso ao *Appium* aos dispositivos desejados. Com o acesso liberado aos dispositivos o *Appium* criará uma conexão via ADB com os dispositivos e executará os *scripts* de teste. Enquanto os testes estiverem sendo executados nos dispositivos, o *Appium* estará realizando a coleta de dados de desempenho nos aparelhos. Ao término deste processo o *Appium* persistirá estes resultados em um arquivo de *log*, o qual pode ser analisado pelo testador.

[6] Ferramenta de automação de compilação utilizada primariamente em projetos Java

Figura 8 – Processo de Teste de APPs com o *Framework* Proposto.



Fonte: O autor.

5.5 Detalhes da Implementação

Nesta seção são apesentadas evidências da implementação do *framework*, como a criação de classes para realização dos testes de desempenho, desenvolvimento da API para permitir a comunicação entre o *Appium* e o *OpenSTF*, geração do relatório e criação de associações entre classes.

5.5.1 Arquitetura do Projeto

Para construir o projeto foi utilizado o *Apache Maven* para automatizar a injeção de dependências. Para realizar os testes de desempenho foram adotadas uma serie de ferramentas para ajudar a desenvolver este *framework*. As ferramentas complementares utilizadas são :

- TestNG[7];
- Okhttp3[8];
- Appium Java Client[9];
- AventStack[10].

[7] TestNG: <https://testng.org/doc/>
[8] Okhttp: <https://github.com/square/okhttp/tree/master/okhttp/src/main/java/okhttp3>
[9] Appium Java Client: <https://github.com/appium/java-client>
[10] AventStack: <http://extentreports.com/>

TestNG fornece uma estrutura de teste para a linguagem de programação Java inspirada em JUnit e NUnit. O objetivo do TestNG é cobrir uma ampla gama de categorias de teste: unitário, funcional, ponta a ponta, integração, etc.

Devido às suas características, é possível criar suítes de casos de casos de teste que auxiliam a criação de vários casos de testes em uma mesma classe de teste. Desta forma, esta estratégia foi adotada para auxiliar o engenheiro de teste a criar os casos de testes e facilitar a legibilidade do código. A Figura 9 apresenta um exemplo de uso na prática.

Figura 9 – Exemplo de Uso do TestNG.

```
@Test
public void realizaTutorial() {...}

@Test(dependsOnMethods = "editaPerfil")
public void realizaCadastro() {...}

@Parameters({"email", "senha"})
@Test(dependsOnMethods = "realizaTutorial")
public void realizaLogin(String email, String senha) {...}
```

Fonte: O autor.

O *Okhttp3* é um cliente que realiza requisições *http*. Esta biblioteca foi utilizada, por apresentar um bom desempenho em realizar requisições *http* na linguagem de programação *Java*. Sua utilização foi no desenvolvimento da API que permite a comunicação entre o *Appium* e o *OpenSTF*.

Com a decisão de adotar o *Appium* para realizar os testes, é necessário o uso do *Appium Java Client* para realizar a comunicação com o servidor do *Appium*, o qual realiza os testes.

AventStack é um *framework* que permite a criação de relatórios de testes usando a linguagem de programação *Java*, podendo acrescentar informações personalizadas no relatório.

5.5.2 Diagrama de Classes do *Framework*

A estrutura do diagrama de classe que o *framework* suporta é apresentado na Figura 10. Neste diagrama existem três classes abstratas, `ConfigAppium`, `ExtentReport` e `ConfigGetPerformance`. A classe `ConfigAppium` é responsável por realizar toda configuração necessária para criar o servidor do *Appium*. A classe `ConfigGetPerformance` implementa toda lógica da coleta de dados de desempenho do dispositivo móvel. A `ExtentReport` é responsável por gerar os relatórios ao final da execução dos casos de testes. É possível observar também que foi adotado o uso de classes abstratas para a solução como um todo, esta abordagem foi utilizada para modularizar a arquitetura do *framework* permitindo com que o modelo viabilize uma estrutura com baixo Acoplamento. Além disso, a classe `ConfigAppium` estende a classe `ConfigGetPerformance` que

por sua vez estende também a classe `ExtentReport`, permitindo assim o desacoplamento das funcionalidades de coleta de dados ou a geração do relatório com poucas alterações. Facilitando também o uso e a manutenção do *framework*.

Um dos métodos mais importantes da classe `ConfigGetPerformane`, responsável por coletar as métricas de desempenho dos dispositivo é o método `snapShotRecursos`. Este método recebe como parâmetro uma *string* que contém o nome do pacote da aplicação que se deseja monitorar no dispositivo, permitindo assim capturar apenas o que esta aplicação esta utilizando do dispositivo. Para realizar a coleta de dados foi utilizado o método `executeScript`[11], fornecido pelo próprio *Appium*, para executar comandos nativos do dispositivo em *shell script*. Contudo é possível utilizar os comandos do *dumpsys*[12], fornecido pelo ADB. Para realizar a coleta foram utilizados os seguintes pacotes do *dumpsys* para realizar a coleta:

- *DumpSys cpuinfo*: Para coletar os dados de desempenho da CPUs;
- *DumpSys batteryinfo*: Para coletar os dados de consumo da bateria;
- *DumpSys meminfo*: Para coletar dados de desempenho da memória;
- *DumpSys netstat*: Para coletar dados de consumo de Internet;
- *DumpSys procstats*: Para coletar dados de consumo de memória principal.

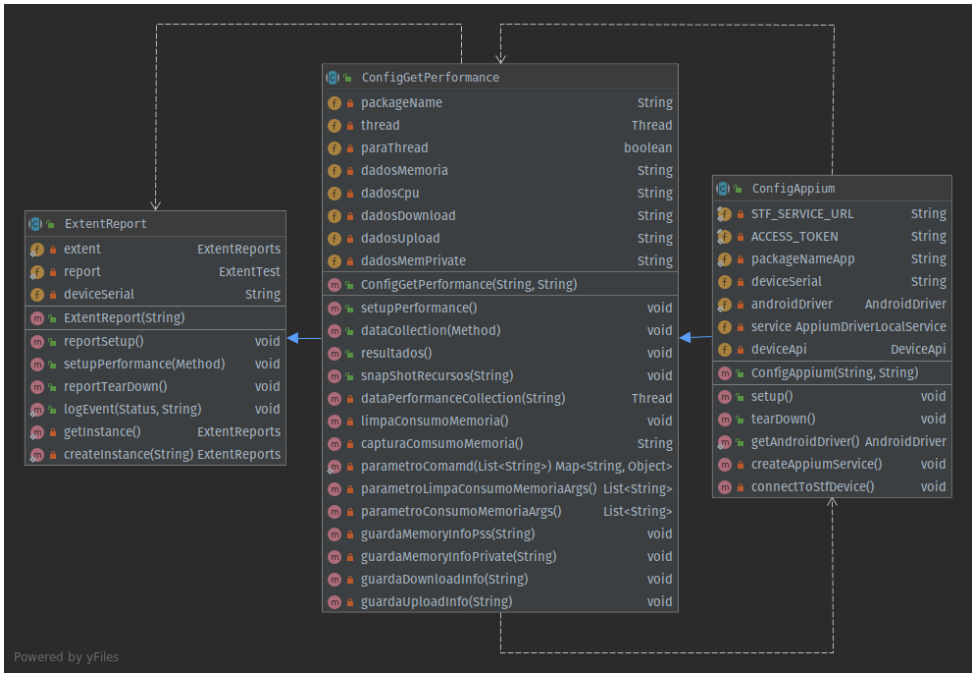
Para realizar o monitoramento do dispositivo em tempo real, enquanto o dispositivo executa o *script* de teste, o método `snapShotRecursos` é chamado dentro de outro método: o `dataPerformanceCollection`, o qual cria uma *Thread* que realiza a coleta assim que o teste é iniciado, encerrando quando o teste é finalizado.

5.5.3 Diagrama de Classes da API

Para estabelecer uma comunicação com os dispositivos locados pelo *OpenSTF* para reservar e liberar qualquer dispositivo é necessário utilizar uma a API *RestFull* fornecida pelo próprio *OpenSTF*. Então foi desenvolvida uma APIs genérica, por meio da linguagem de programação *Java*, capaz de utilizar o serviço. Para isto foram criadas duas classes: `DeviceApi` e `STFService`. O objeto `STFService` contem duas informações apenas: *url*, o qual o serviço do *OpenSTF* está sendo executado, e o *Token* de acesso. Já a classe `DeviceApi` realiza as requisições para solicitar acesso ao dispositivo que se deseja, por meio do método `connectDevice`. A Figura 11 apresenta o diagrama de classe da API.

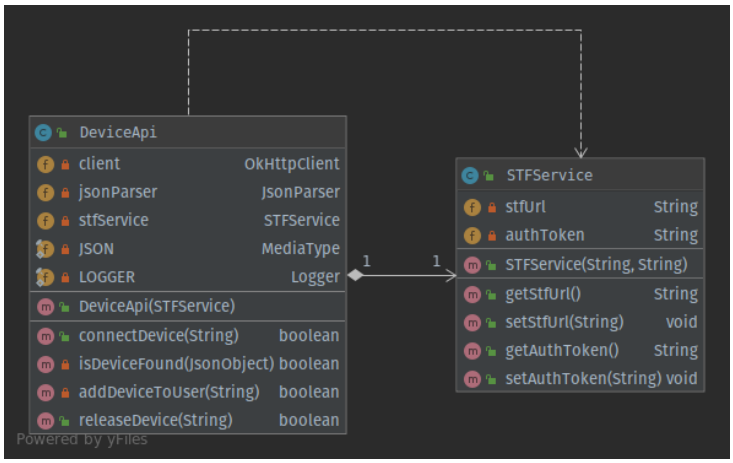
[11] ExecuteScript : <http://appium.io/docs/en/commands/mobile-command/>
[12] DumpSys : <https://developer.android.com/studio/command-line/dumpsys>

Figura 10 – Diagrama de Classe do *Framework*.



Fonte: O autor.

Figura 11 – Diagrama de Classe da API.



Fonte: O autor.

5.6 Exemplo de Uso do *Framework*

Após a instalação de toda infraestrutura necessária, bem como a criação do projeto com suas respectivas dependências, já é possível executar os testes. Primeiramente, deve-se criar uma classe de teste que estenda a classe do *framework* `configAppium`. Após a importação a IDE o obigará a implementa o construtor da classe e inserir os parâmetros exigidos pela classe `configAppium`. Os parâmetros exigidos são o id do dispositivo e o nome do pacote da aplicação que deve ser observada durante os testes. Caso se deseje realizar uma execução dos teste em múltiplos dispositivos é necessário usar o estereótipo "@Factory" sobre o construtor. Com isso o *TestNG* criará uma instância da classe de teste

para cada cada dispositivo. A Figura 12 apresenta o exemplo de código do construtor da classe de teste.

Figura 12 – Construtor da Classe de Teste.

```
public class Agendei extends ConfigAppium {


    @Factory(dataProvider = "parallelDp", dataProviderClass = DataDeviceId.class)
    public Agendei(String deviceSerial) {
        super(deviceSerial, packageName: "com.agendeiquadras");
    }
```

Fonte: O autor.

Contudo, para permitir a execução em paralelo, é necessário criar uma nova classe que tenha tenha os objetos referentes aos dispositivos que se deseja executar os casos de teste. Nesta nova classe é necessário criar um método, como exemplificado na Figura 13, anotado com o esteriótipo "@DataProvider" e que retorne uma matriz de objetos com os dados à serem processados.

Figura 13 – Classe de Dados dos Dispositivos.

```
public class DataDeviceId {

    @DataProvider (name = "parallelDp")
    public Object[][] parallelDp() {
        return new Object[][]{
                {"420077960157947b"},
        };
    }
}
```

Fonte: O autor.

Uma última configuração necessária para que seja possível executar os testes é composta das seguintes itens: 1) inserir as informações do *link* do local onde o serviço do *OpenSTF* esta sendo executado; 2) inserir o *token* de acesso e; 3) acrescentar o caminho que se encontra o APK na classe `ConfigAppium`. A Figura 14 apresenta o exemplo de código das configurações necessárias.

Figura 14 – Configurações da Classe `ConfigAppium`.

```
private static final String STF_SERVICE_URL = "http://10.0.2.15:7100";
private static final String ACCESS_TOKEN = "cf5a46faa8bd4f48baa8ffddf02722890611c1a7138c4a6f8cc05a63de2c033a";
@BeforeClass
public void setup() throws MalformedURLException, URISyntaxException {
    DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
    desiredCapabilities.setCapability(MobileCapabilityType.DEVICE_NAME, value: "ANDROID");
    desiredCapabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, value: "ANDROID");
    desiredCapabilities.setCapability(MobileCapabilityType.UDID, this.deviceSerial);
    desiredCapabilities.setCapability(MobileCapabilityType.APP, new File( pathname: "src/test/resources/aplicacao.apk").getAbsolutePath());
    connectToStfDevice();
    createAppiumService();
    androidDriver = new AndroidDriver(this.service.getUrl(), desiredCapabilities);
}
```

Fonte: O autor.

Com essas configurações, já é possível realizar os testes de desempenho nos APPs. Para realizar os testes, basta criar um método sem retorno e acrescentar o esteriótipo "@Test" sobre o mesmo. Dentro deste método, deve-se inserir os *scripts* de teste do *Appium* escritos na linguagem de programação Java. Além disso o *framework* disponibiliza dois métodos chamados `LogEvent` e `getAndroidDriver`. O método `LogEvent` permite o testador colocar alertas dentro do *scripts* de teste para que se saiba no relatório gerado pelo *framework* o que esta acontecendo em determinado tempo. Este método possuí dois parâmetros, o primeiro é qual o tipo de alerta e o segundo uma *string* quem tenha a mensagem que deseja apresentar no relatório. Com relação ao primeiro parâmetro, existem 3 tipos de alertas, sendo eles: *PASS* (Passou), *INFO* (Informação) e *FAIL* (Falhou). Outro método que é frequentemente usado é o `getAndroidDriver`, o qual retorna o *driver* do *Appium*, para que seja possível criar os *scripts* de teste. A Figura 15 apresenta um exemplo de método de teste.

Figura 15 – Caso de Teste.

```
@Test
public void realizaTutorial() {
    logEvent(Status.INFO, text: "Iniciou o teste");
    getAndroidDriver().manage().timeouts().implicitlyWait( l: 5000, TimeUnit.MILLISECONDS);
    logEvent(Status.PASS, text: "Iniciou o tutorial");
    MobileElement el1 = (MobileElement) getAndroidDriver().findElementByXPath( using: "//*[@text='"+"Próximo"+"']");
    el1.click();
    el1.click();
    el1.click();
}
```

Fonte: O autor.

5.7 Lições do Capítulo

Este capítulo apresenta um relato de como foi projetado e desenvolvido o *framework* proposto. Os principais pontos abordados neste capítulo são:

- Principais requisitos, levados em consideração no escopo da proposta;
- Decisões de projeto, tomados com base nos requisitos para construir a solução;
- Arquitetura geral e funcionamento da solução.
- Exemplo de uso do *framework*.

Por meio do relato de um estudo de caso, o próximo capítulo apresenta a avaliação do *framework*.

6 ESTUDO DE CASO

A fim de avaliar o uso do *framework* proposto, este capítulo tem por propósito descrever a avaliação do estudo de caso. A Seção 6.1 apresenta o planejamento do estudo de caso que foi realizado. A Seção 6.2 mostra como foi realizada a condução do estudo de caso. A Seção 4.4 apresenta os resultados obtidos no estudo de caso e uma discussão sobre o mesmo. A Seção 6.4 discute as limitações que houveram durante o estudo de caso. A Seção 4.5 descreve as principais ameaças à validade do estudo.

6.1 Planejamento

Uma vez que o *framework* foi desenvolvido, foi planejada a realização de um estudo de caso com o propósito de analisar como a proposta reagiria em um ambiente real. Para isso, o estudo de caso seguiu como orientação um *template* base proposto por Brereton et al. (2008).

Este estudo de caso seguiu três fases:

1. **Fase 1**: Treinamento do *framework*;
2. **Fase 2**: Utilização do *framework* para realizar teste de desempenho;
3. **Fase 3**: Entrevista pós-conhecimento.

A Fase 1 tem o objetivo de apresentar o funcionamento geral do *framework* para os integrantes da empresa participante do estudo. Na Fase 2 a execução dos casos de teste e toda a configuração de ambiente foi realizada. Por último, na Fase 3 são apresentados os resultados obtidos anteriormente na Fase 2, que são apresentados na Seção A.2. Ainda na Fase 3 é realizado uma entrevista com os participantes para analisar a viabilidade de uso do *framework*.

6.1.1 Contexto e Unidade de Análise

A empresa participante do estudo de caso é a DevPampa [1], a qual foi escolhida por trabalhar com desenvolvimento de APPs, que se encaixam ao contexto de uso do framework.

A aplicação que foi selecionada para a realização dos testes de desempenho trata-se de uma aplicação de reserva de quadras esportivas: o Agendei Quadras[2]. Esta aplicação disponibiliza funcionalidades relacionadas ao gerenciamento de quadras esportivas, como por exemplo:

- Realizar *login*;

[1] DevPampa: <https://devpampa.com>

[2] Agendei Quadras: <https://play.google.com/store/apps/details?id=com.agendeiquadras&hl=pt_BR>

- Criar perfil;
- Editar perfil;
- Reservar quadras;
- Visualizar horário das quadras;
- Cancelar reservas e etc.

A empresa disponibilizou para os participantes do estudo de avaliação uma "imagem" de uma máquina virtual contendo todas as ferramentas necessárias instaladas e testadas. Assim, os envolvidos com o processo de avaliação, preocuparam-se apenas com o uso do *framework*. Esta máquina virtual dispunha de 4 Giga Byte (GB) de memória principal e uma CPU de 2 núcleos apenas.

Participaram deste estudo de caso 3 integrantes da equipe de desenvolvimento da DevPampa. Dois são desenvolvedores *Full Stack* o outro atua como gerente de projetos da equipe.

6.1.2 Protocolo

Com o objetivo de executar um estudo de caso de forma mais estruturada, foi aplicado novamente o GQM para conduzir o estudo. Como se trata de uma avaliação exploratória, este Estudo de Caso teve como foco obter resultados qualitativos. Com base nisso, foi elaborado o objetivo, as questões de pesquisa e as métricas de coleta.

6.1.2.1 Objetivo

Com o objetivo de verificar o uso do *framework* desenvolvido em um ambiente real, e seguindo o *template* do GQM, foi estruturada a seguinte meta para estudo.

- Meta: **Analisar** o uso do *framework* para a realização de teste de desempenho em aplicações móveis, **no intuito** de avaliar sua viabilidade de uso, **no que diz respeito** ao teste de desempenho em aplicativos móveis **na perspectiva** dos avaliadores e desenvolvedores de APP **no contexto** da indústria.

Nas seções seguintes, serão descritas as questões de pesquisa e os critérios de qualidade que fazem parte da estrutura do protocolo.

6.1.2.2 Questões de Pesquisa

Com base no objetivo do estudo de caso, foram formuladas as seguintes RQs que respondem a meta do estudo de caso.

- **RQ1.** O *framework* oferece suporte para a realização de teste de desempenho em APPs?

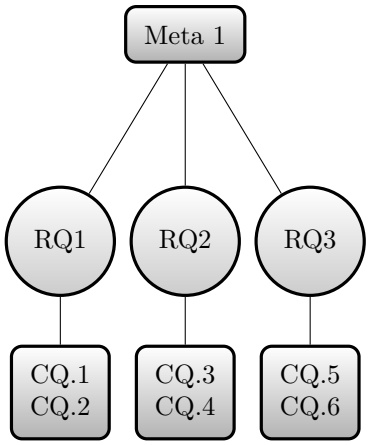
- **RQ2.** Qual a complexidade na utilização deste *framework* para realizar teste de desempenho em APPs?

- **RQ3.** Os resultados gerados pelo *framework* ajudam na compreensão do desempenho do APP em relação aos casos de testes?

6.1.2.3 Critérios de Qualidade

Para medir a qualidade dos artefatos gerados e a viabilidade de uso do *framework*, foi criado dois CQs para cada RQ. Cada critério pode ter três pontuações possíveis: Total (T) que pontua 1, Parcial (P) que pontua 0,5 e Nada (N) que pontua 0 (zero). Para responder as questões de pesquisa as CQ foram transformadas nas métricas que respondem as RQs. A Figura 16 apresenta a estrutura geral do GQM adotada para o estudo de caso.

Figura 16 – Estrutura GQM do Estudo de Caso.



Fonte: O autor.

Os CQs são descritos a seguir:

- **CQ1.** Métricas de desempenho coletadas durante o teste de desempenho são úteis para a organização?
  T: Totalmente úteis para a organização;
  P: Parcialmente úteis para a organização;
  N: Não agrega valor algum para a organização.

- **CQ2.** As métricas de desempenho coletadas e os relatórios gerado pelo *framework* são suficientes para dar suporte a teste de desempenho em APPs?
  T: Oferece suporte totalmente na realização de testes de desempenho nos APPs;

P: Oferece suporte parcialmente na realização de testes de desempenho nos APPs;
N: Não oferece suporte algum para realizar testes de desempenho nos APPs.

- **CQ3.** Qual o nível de complexidade para criar os casos de teste de desempenho e executá-los de forma automatizada?
  T: Nível baixo de complexidade para criar os casos de teste e executá-los de forma automatizada;
  P: Nível médio de complexidade para criar os casos de teste e executá-los de forma automatizada;
  N: Nível Alto de complexidade para criar os casos de teste e executá-los de forma automatizada.

- **CQ4.** Qual o nível de complexidade em realizar as configurações necessárias para realizar o teste de desempenho?
  T: Nível baixo de complexidade para realizar as configurações;
  P: Nível médio de complexidade para realizar as configurações;
  N: Nível Alto de complexidade para realizar as configurações.

- **CQ5.** As métricas apresentadas nos relatórios são suficientes para analisar o desempenho do APP?
  T: São suficientes para analisar o desempenho do APP;
  P: Atendem parcialmente a analise de desempenho do APP;
  N: Não é possível analisar o desempenho do APP.

- **CQ6.** Os dados apresentado nos relatórios são de fácil entendimento e compreensão?
  T: Fácil compreensão dos resultados;
  P: Compreensão parcial dos resultados;
  N: Difícil compreensão dos resultados.

6.2 Condução

Na Fase 1 foi realizado um treinamento com os participantes do estudo. Inicialmente foi apresentada uma visão geral do *framework*, em seguida foram mostradas todas as dependências do projeto e a infraestrutura necessária para suportar o *framework*. O próximo passo foi ensinar como criar os métodos de teste e apresentar todo o ferramental que auxilia a criação dos *scripts* de teste. Por fim, foram passadas as configurações

que devem ser feitas nos *scripts* para que o *framework* funcione adequadamente. Este treinamento teve duração aproximada de 30 minutos.

Para realizar a Fase 2, o testador seguiu todo o processo descrito na Seção 5.4. Antes de projetar os casos de teste foi analisada uma lista contendo vários casos de testes referentes a aplicação sob teste. Com base nesta lista, foi elaborado um conjunto de casos de teste para que fosse possível testar a aplicação. Contudo, vale lembra que o foco do estudo não era testar as funcionalidades da aplicação e sim realizar a coleta de dados sobre o uso do *framework*. Em seguida, foram criados os *scripts* de teste e executada uma suíte de teste sobre o APP fornecido. Como resultado obteve-se um relatório da execução dos teste e os dados de desempenho da aplicação sobre o dispositivo.

Por fim, na Fase 3 foi conduzida uma entrevista que seguiu como base o protocolo descrito na Seção 6.1. Nesta entrevista, os participantes poderiam tirar dúvidas, interromper para dar sugestões de melhorias e responder as perguntas que eram realizadas pelo autor. Primeiramente, foram apresentados dois relatórios, um gerado manualmente, o qual apresentava o uso dos recursos da rede (*download* e *upload*) e outro, que apresentava um relatório gerado pelo *framework* de consumo de memória. Em seguida, foram realizadas as 6 (seis) perguntas, sendo cada uma delas um CQ definido para este estudo de caso. A entrevista teve duração aproximada de 1 hora. Vale lembrar que antes de iniciar a Fase 3, foi entregue aos participantes um termo de consentimento, o termo é apresentado no Apêndice A.1.

6.3 Resultados e Discussão

Para obter as informações necessárias para responder as RQ, foi conduzida a Fase 3, descrita na Seção 6.2. Com isso, foi possível extrair informações por meio da entrevista com os participantes do estudo de caso. Com essa entrevista foram coletados dados para extrair informações dos resultados apresentados a fim de responder os critérios de qualidade estabelecidos. Como resultado para RQ1 obteve-se uma pontuação por meio dos critérios de qualidade, uma "Parcial" e outra "Total", alcançando 1.5 pontos. para esta questão. Para a RQ2 foi alcançada a pontuação de apenas 1 ponto, sendo a única a receber uma nota "nada" e outra "Total". A RQ3 conseguiu pontuar 1.5 recebendo duas notas, uma "Parcial" e outra "Total". Os resultados são apresentados na Tabela 15.

Tabela 15 – Resultado dos Critérios de Qualidade.

RQ	CQ	Nota	Total
RQ1	CQ1	T	
	CQ2	P	1.5
RQ2	CQ3	T	
	CQ4	N	1
RQ3	CQ5	P	
	CQ6	T	1.5

Fonte: O autor.

Como o estudo de caso foi um estudo exploratório, um dos principais objetivos das questões de pesquisa estão relacionadas a encontrar possíveis melhorias e verificar o uso do *framework*.

As seções seguintes descrevem detalhadamente o resultado da execução da Fase 3. Para apresentar os resultados, as (CQ) foram agrupados dentro das seções, que apresentam o resultado de cada uma das RQ estabelecidas.

6.3.1 RQ1. O *framework* oferece suporte para a realização de teste de desempenho em APPs?

O objetivo desta RQ é coletar evidências para identificar se o *framework* esta fornecendo suporte adequado para realizar testes de despenho em APPs. Para ajudar a responder esta questão, dois critérios são levadas em consideração. Verificar se o resultado gerado pelo *framework* é relevante para a organização e se há necessidade do *framework* coletar mais métricas de desempenho durante os testes.

CQ1. **As métricas de desempenho coletadas durante o teste de desempenho são úteis para a organização?**

Os participantes consideraram que as métricas coletadas são totalmente úteis para a organização. Um dos argumentos mostra que os dados gerados auxiliam na tomada de decisão da evolução do APP. Ou seja, ajuda na identificação de quais funcionalidades devem ser otimizadas. Além disso foi proposta uma melhoria para o *framework*, para que seja possível guardar o histórico de desempenho de cada teste. Quando for executar novamente o mesmo caso de teste o *framework*, então, apresenta uma comparação dos dados de desempenho com os dados das execuções anteriores.

CQ2. **As métricas de desempenho coletadas e os relatórios gerado pelo *framework* são suficientes para dar suporte a teste de desempenho em APPs?**

Para este critério foi considerado que o *framework* oferece suporte parcial na realização de testes de desempenho nos APP. Os participantes sentiram falta principalmente da métrica de consumo da CPU. Porem, ressaltaram que para organizações que não possuem nenhuma métrica de desempenho de suas aplicações, as métricas oferecidas até o momento conseguem auxiliar a evolução da aplicação.

6.3.2 RQ2. Qual a complexidade na utilização deste *framework* para realizar teste de desempenho em APPs?

O objetivo desta questão é analisar a complexidade na realização dos testes de desempenho em APPs. Para responder esta questão, foi levando em consideração desde a estrutura do TestNG para fazer os testes até as configurações necessárias no *framework* para realizar os teste de desempenho nos APPs.

CQ3. **Qual o nível de complexidade para criar os casos de teste de desempenho e executá-los de forma automatizada?**

Neste quesito, os participantes consideraram um nível baixo de complexidade para criar os casos de teste e executá-los de forma automatizada. Possivelmente tiveram esta opinião, por já estarem familiarizados com a linguagem de programação *Java*, o qual é a linguagem predominante dos participantes. Além disso, possuíam um conhecimento prévio da estrutura de teste do *TestNG*, pois possui uma semelhança muito grande com a estrutura de teste do *Junit*, o qual já tinham conhecimento.

CQ4. Qual o nível de complexidade em realizar as configurações necessárias para realizar o teste de desempenho?

Os participantes consideraram um nível alto de complexidade para realizar as configurações. Durante a busca de resposta para esta pergunta, os participantes tiveram muitas dúvidas sobre a classe onde se faz as configurações. Ou seja, dificuldades para operacionalizar o *Appium* e o *OpenSTF*, para que funcionem corretamente.

6.3.3 RQ3. Os resultados gerados pelo *framework* ajudam na compreensão do desempenho do APP em relação aos casos de testes?

O objetivo desta questão é analisar se os resultados gerados pelo *frameowrk* ajudam a compreender o desempenho da aplicação que está sendo testada. Dois pontos são levados em consideração: se os relatórios gerados possuem uma boa legibilidade e, se os dados apresentados de fato ajudam a compreender o desempenho do APP.

CQ5. As métricas apresentadas nos relatórios são suficientes para analisar o desempenho do APPs?

Nesta questão os participantes argumentam novamente a ausência da métrica de uso da CPU. Para esta questão, afirmam que as métricas dos relatórios atendem parcialmente a análise de desempenho do APP.

CQ6. Os dados apresentados nos relatórios são de fácil entendimento e compreensão?

Os participantes consideraram fácil a compreensão dos resultados apresentadas pelo relatório. Também foram feitas algumas sugestões para os relatórios futuro. Primeiro, recomendam a unificação dos dois relatórios apresentados em apenas um documento. Segundo, sugerem aumentar a escala de observação do tempo do relatório do consumo de memória. E, por último, diminuir a escala de consumo de rede de Megabyte (MB) para Kilobyte (KB). O Apêndice A.2 apresenta os relatórios gerados pelo *framework*.

6.4 Limitações do Projeto

Apesar do *framework* realizar a coleta de dados de desempenho da CPU e de bateria do dispositivo, não foi possível realizar a coleta dos mesmo. Em alguns casos de testes foi observado que a informação com o *status* de uso da CPU demorava mais de

10 segundos para ser capturado, criando assim grandes intervalos de tempo entre cada ponto de coleta da informação. Com isto, foi decidido desabilitar essa funcionalidade no estudo de caso. Com relação ao consumo de bateria, esta função também foi desativada durante a execução do estudo. Assim, o dispositivo permaneceu conectado via cabo USB no serviço do *OpenSTF*. Ou seja, o dispositivo permanecia sendo carregado enquanto os testes eram executados, dificultando portanto a análise de consumo de bateria enquanto o teste era executado.

Uma outra limitação foi a execução de teste em dispositivos emulados. Essa limitação deve-se ao fato de que toda a arquitetura do projeto, apresentada na Seção 5.3, foi executada em uma máquina virtual. Contudo, não é possível criar um dispositivo virtual Android dentro de uma máquina virtual[3].

6.5 Ameaças à Validade do Estudo

Ao longo do planejamento e execução do estudo, se analisou e discutiu as ameaças à validade deste estudo, bem como as estratégias utilizadas para mitigá-las. Para a listagem das possíveis ameaças, foram utilizados alguns tópicos e recomendações levantadas no trabalho de Wohlin et al. (2012), outas ameaças percebidas pelo autor. Essas ameaças seguiram o padrão proposto e foram divididas em quatro (4) categorias, sendo: validade de construção, validade interna, validade externa e validade de conclusão.

Validade de Construção: Essa categoria está relacionada com as questões de projeto do desenho experimental e fatores sociais. É possível que os participantes tenham levantado supostas hipóteses sobre o objetivo das perguntas realizadas durante a entrevista, podendo se comportar de forma diferente. Assim, é plausível que estes fatores possam influenciar positivamente ou negativamente no estudo. Com o objetivo de mitigar esta ameaça, o pesquisador buscou aprofundar a discussão referente às dúvidas, mas sem de fato revelar os objetivos específicos e propósito final da avaliação.

Validade Externa: Essa categoria agrupa o conjunto de ameaças referentes à capacidade de replicação do estudo de caso. Sendo assim, é possível supor que a equipe/organização escolhida para executar as atividades planejadas na avaliação do *framework* proposto não seja representativa para toda população. Para atenuar esta ameaça, optou-se por uma organização que se inserisse no contexto de empresas desenvolvedoras de APPs. Também foi selecionada uma equipe que estivesse envolvida continuamente com o desenvolvimento de APPs.

Validade Interna: Essa categoria diz respeito às influências que podem afetar as variáveis independentes. Neste contexto, é possível que os participantes passem por momentos de frustração ou tédio durante a condução da avaliação. Por conta disso, é possível que esta situação afete o desempenho dos participantes. Com o objetivo de

[3] ExecuteScript : <https://forums.virtualbox.org/viewtopic.php?f=6&t=69076>

mitigar esta ameaça, foi adotada a estratégia de proporcionar flexibilidade em relação ao tempo disponível para o andamento do estudo de caso. Isso resultou na divisão da execução em 3 etapas bem definidas: (i) treinamento (ii) criação dos casos de teste e (iii) entrevista. O treinamento teve uma duração média de 30 minutos, enquanto que a entrevista perdurou por cerca de 1 hora. Por outro lado, a etapa voltada a criação dos casos de teste, considerada a mais mais onerosa de todas, ocorreu no espaço de um dia.

Validade de Conclusão: Essa categoria está relacionada aos métodos de análise e interpretação dos resultados obtidos. Em outras palavras, diz respeito a forma que conclusões inferidas são baseadas em dados observados no estudo de caso. Para mitigar o potencial problema de se chegar a resultados não significativos para a análise, foi criada uma máquina virtual contendo todo ferramental necessário para apoiar os participantes durante os testes de desempenho nos APPs. Neste ambiente controlado, todas as métricas foram colhidas e armazenadas para posterior observação.

7 CONSIDERAÇÕES FINAIS

Este estudo abordou o projeto do TCC, expondo seu planejamento metodológico, a fundamentação teórica, um MLM, proposta de um *framework* para realização de teste de desempenho em APPs e por fim uma avaliação empírica por meio de um estudo de caso na indústria.

Levando em consideração todos os problemas na aplicação de teste em APPs descritos na Seção 3.3 e o notório crescimento dos dispositivos móveis apresentados no estudo de Bayindir e Winther (2018), é possível inferir que um *framework* para realizar testes de desempenho em APPs é necessário para avaliar o desempenho dos mesmos. Os resultados do MLM executado indicam que a maioria das ferramentas e *frameworks* que oferecem suporte para medição de desempenho de um APP são em sua grande maioria pagas, aumentando a importância da proposta deste trabalho, pois trata-se de uma solução *open source*.

Como principais resultados obtidos, destaca-se a investigação conduzida para compreender o estado da arte e da prática do tema central desta pesquisa, bem como a construção de uma proposta de *framework* para realizar teste de desempenho em APPs.

A solução desenvolvida permite coletar métricas relacionadas ao desempenho de um APP em um dispositivo móvel, com o intuído de ajudar pesquisadores e engenheiros de teste a entenderem melhor o desempenho das aplicações.

Um estudo de caso também foi executado para verificar a viabilidade de uso do *framework* no mundo real. Como resultado do estudo de caso é possível inferir que o *framework* proposto pode ser usado para realizar os testes almejados. Entretanto, ainda são necessários ajustes para que o *framework* contemple algumas funcionalidades que foram sugeridas pelos participantes do estudo de caso.

7.1 Trabalhos Futuros

Apesar do *framework* estar pronto e funcional, ainda falta aplicar algumas melhorias como:

- Otimizar a coleta de dados de CPU;
- Facilitar a configuração necessária para o funcionamento do *framework*;
- Unificar a geração automática dos relatórios;
- Transformar o *framework* em uma “lib” para que possa ser instanciado facilmente em qualquer projeto;
- Criação do repositório e *wiki* do projeto;
- Permitir conexão com dispositivos remotamente para coleta de dados de bateria.

7.2 Publicações

Um ponto que merece destaque foi o esforço dos pesquisadores envolvidos neste estudo para a realização de um artigo científico, o qual foi submetido ao Simpósio Brasileiro de Engenharia de Software (SBES'2019) abordando o SMS conduzida. Contudo, o estudo não foi aprovado, porém o mesmo foi refatorado e enviado para outro evento, o Simpósio Brasileiro de Qualidade de Software (SBQS'2019)[1], o qual foi aprovado e apresentado no evento que aconteceu em Fortaleza/CE entre os dias 28 de outubro e 1º de novembro de 2019.

Além disso, este estudo possui ainda dois estudos que pretende-se publicá-los em um *Journal*. Sendo eles a busca na literatura cinza e a avaliação do estudo de caso da proposta do *framework* para realizar testes de desempenho em APP.

[1] SBQS'2019: <http://sbqs.sbc.org.br/>

REFERÊNCIAS

AL-AHMAD, A. S.; ALJUNID, S. A.; SANI, A. S. A. Mobile cloud computing testing review. **International Conference on Advanced Computer Science Applications and Technologies**, p. 176–180, 2014. Citado na página 56.

AMALFITANO, D. et al. **Testing Android Mobile Applications: Challenges, Strategies, and Approaches**. [S.l.]: Elsevier Inc., 2013. v. 89. 1–52 p. ISSN 00652458. ISBN 9780124080942. Citado 6 vezes nas páginas 21, 33, 40, 48, 50 e 51.

BASILI, V.; CALDIERA, C.; ROMBACH, H. Goal question metric paradigm. **Encyclopedia of software engineering**, v. 1, p. 528–532, 1994. Citado 3 vezes nas páginas 22, 23 e 40.

BAYINDIR, N.; WINTHER, E. The device trends to watch in 2019. 2018. Citado 2 vezes nas páginas 21 e 81.

BIOLCHINI, J. et al. Systematic review in software engineering. **System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES**, v. 679, n. 05, p. 45, 2005. Citado na página 39.

BRAUN, S.; ELBERZHAGER, F.; HOLL, K. Automation support for mobile app quality assurance–a tool landscape. **Procedia Computer Science**, Elsevier, v. 110, p. 117–124, 2017. Citado na página 56.

BRERETON, P. et al. Using a protocol template for case study planning. In: **Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering**. Swindon, UK: BCS Learning & Development Ltd., 2008. (EASE'08), p. 41–48. Citado na página 71.

CHEN, G.; KOTZ, D. A survey of context-aware mobile computing research. **Dartmouth Computer Science Technical Report TR2000-381**, 2000. Citado na página 33.

COHEN, J. A coefficient of agreement for nominal scales. **Educational and Psychological Measurement**, v. 20, n. 1, p. 37–46, 1960. Citado na página 47.

DEKA, B. et al. ZIPT. In: **In 30th Annual ACM Symposium on User Interface Software and Technology**. [S.l.]: ACM Press, 2017. p. 727–736. ISBN 9781450349819. Citado na página 48.

DELAMARO, J. C. E.; JINO, M. **Introdução ao teste de software**. [S.l.]: Elsevier Brasil, 2004. Citado na página 29.

GAROUSI, V.; FELDERER, M.; MäNTYLä, M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. **Information and Software Technology**, v. 106, p. 101 – 121, 2019. ISSN 0950-5849. Citado na página 39.

GOEL, U. et al. Mitate: Mobile internet testbed for application traffic experimentation. In: SPRINGER. **International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services**. [S.l.], 2013. p. 224–236. ISBN 978-3-319-11569-6. Citado 6 vezes nas páginas 21, 48, 49, 50, 51 e 53.

HYMAN, R. Quasi-experimentation: Design and analysis issues for field settings (book). **Journal of Personality Assessment**, Taylor & Francis, v. 46, n. 1, p. 96–97, 1982. Citado na página 54.

JOORABCHI, A. M. M. E.; KRUCHTEN, P. Real challenges in mobile app development. In: **2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement**. [S.l.: s.n.], 2013. p. 15–24. ISSN 1949-3770. Citado na página 22.

JOORABCHI, M. E.; MESBAH, A.; KRUCHTEN, P. Real challenges in mobile app development. In: **2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement**. [S.l.: s.n.], 2013. p. 15–24. ISSN 1949-3770. Citado na página 21.

KIM, H.; CHOI, B.; WONG, W. E. Performance testing of mobile applications at the unit test level. In: IEEE. **In 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement**. [S.l.], 2009. p. 171–180. Citado 8 vezes nas páginas 33, 40, 48, 49, 50, 51, 52 e 53.

KIM, H.; CHOI, B.; YOON, S. Performance testing based on test-driven development for mobile applications. In: **In 3rd International Conference on Ubiquitous Information Management and Communication**. [S.l.]: ACM, 2009. p. 612–617. ISBN 978-1-60558-405-8. Citado 6 vezes nas páginas 40, 48, 49, 52, 53 e 55.

KIM, H. K. Mobile applications software testing methodology. **Communications in Computer and Information Science**, v. 342 CCIS, p. 158–166, 2012. ISSN 18650929. Citado na página 48.

KÖCHE, J. C. **Fundamentos de metodologia científica: Teoria da ciência e iniciação à pesquisa, 24ª**. [S.l.]: Edição atualizada. Petrópolis: Editora Vozes, 2007. Citado na página 25.

KOCHHAR, P. S. et al. Understanding the test automation culture of app developers. In: **IEEE 8th International Conference on Software Testing, Verification and Validation**. [S.l.: s.n.], 2015. p. 1–10. ISSN 2159-4848. Citado na página 21.

LANDIS, J. R.; KOCH, G. G. The measurement of observer agreement for categorical data. **Biometrics**, Wiley, International Biometric Society, v. 33, n. 1, p. 159–174, 1977. Citado 2 vezes nas páginas 47 e 48.

LINARES-VáSQUEZ, M. et al. Enabling mutation testing for android apps. In: **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: ACM, 2017. (ESEC/FSE 2017), p. 233–244. ISBN 978-1-4503-5105-8. Citado na página 36.

Linares-Vásquez, M.; Moran, K.; Poshyvanyk, D. Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing. In: **2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [S.l.: s.n.], 2017. p. 399–410. Citado 2 vezes nas páginas 35 e 36.

LIU, Y. H. Z.; CAI, L. Software quality testing model for mobile application. In: AWAN, I. et al. (Ed.). **Mobile Web Information Systems**. Cham: Springer International Publishing, 2014. p. 192–204. ISBN 978-3-319-10359-4. Citado na página 22.

LIU, Z.; HU, Y.; CAI, L. Software quality testing model for mobile application. In: **Lecture Notes in Computer Science**. [S.l.: s.n.], 2014. v. 8640 LNCS, p. 192–204. Citado na página 48.

LUO, Q. et al. An empirical analysis of flaky tests. In: **Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, NY, USA: ACM, 2014. (FSE 2014), p. 643–653. ISBN 978-1-4503-3056-5. Citado na página 36.

Malini, A. et al. Mobile application testing on smart devices using mtaas framework in cloud. In: **International Conference on Computing and Communication Technologies**. [S.l.]: IEEE, 2014. p. 1–5. Citado 7 vezes nas páginas 21, 37, 48, 49, 51, 53 e 55.

MEIER, J. et al. **Performance Testing Guidance for Web Applications: Patterns & Practices**. Redmond, WA, USA: Microsoft Press, 2007. ISBN 9780735625709. Citado 2 vezes nas páginas 31 e 32.

MOLYNEAUX, I. **The Art of Application Performance Testing: Help for Programmers and Quality Assurance**. 1st. ed. Manchester, UK: O'Reilly Media, Inc., 2009. Citado 2 vezes nas páginas 31 e 34.

MUCCINI, H.; FRANCESCO, A. D.; ESPOSITO, P. Software testing of mobile applications: Challenges and future research directions. In: **Proceedings of the 7th International Workshop on Automation of Software Test**. Piscataway, NJ, USA: IEEE Press, 2012. (AST '12), p. 29–35. ISBN 978-1-4673-1822-8. Citado 2 vezes nas páginas 21 e 33.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3rd. ed. Hoboken, New jersey, USA: Wiley Publishing, 2011. Citado 3 vezes nas páginas 30, 33 e 34.

NAKAGAWA, E. et al. **Revisão Sistemática da Literatura em Engenharia de Software: Teoria e Prática**. São Paulo, SP: Elsevier Editora Ltda., 2017. Citado 2 vezes nas páginas 39 e 57.

NANDAKUMAR, V.; EKAMBARAM, V.; SHARMA, V. Appstrument-a unified app instrumentation and automated playback framework for testing mobile applications. In: **International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services**. Tokyo, JP: Springer International Publishing, 2013. p. 474–486. Citado 7 vezes nas páginas 21, 38, 40, 48, 49, 51 e 53.

Patton, R. **Software Testing**. Indianapolis, Indiana, USA: SAMS, 2000. Citado na página 29.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: **12th International Conference on Evaluation and Assessment in Software Engineering**. Italy: BCS Learning & Development Ltd., 2008. p. 68–77. Citado 3 vezes nas páginas 39, 40 e 44.

Prathibhan, C. M. et al. An automated testing framework for testing android mobile applications in the cloud. In: **IEEE International Conference on Advanced**

Communications, Control and Computing Technologies. USA: IEEE, 2014. p. 1216–1219. Citado 5 vezes nas páginas 38, 48, 49, 51 e 53.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição**. Novo Hamburgo, Rio Grande do Sul, BR: Editora Feevale, 2013. Citado na página 25.

Rajan, V. S. S.; Malini, A.; Sundarakantham, K. Performance evaluation of online mobile application using test my app. In: IEEE. **2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies**. USA, 2014. p. 1148–1152. Citado 4 vezes nas páginas 37, 48, 51 e 53.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. Nova York, USA: Malaysia; Pearson Education Limited,, 2016. Citado na página 45.

SAHINOGLU, M.; INCKI, K.; AKTAS, M. S. Mobile application verification: A systematic mapping study. In: GERVASI, O. et al. (Ed.). **Computational Science and Its Applications – ICCSA 2015**. Cham: Springer International Publishing, 2015. p. 147–163. Citado na página 22.

SANTOS, A.; CORREIA, I. Mobile testing in software industry using agile: Challenges and opportunities. In: **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. Graz, AUT: IEEE, 2015. p. 1–2. Citado na página 21.

SANTOS VALéRIA L .L. DANTAS, R. M. S. R. M. C. A. I. S. Testes de Aplicações Móveis: uma análise das pesquisas científicas via revisão sistemática. **Lbd.Dcc.Ufmg.Br**, XI Simpósio Brasileiro de Qualidade de Software, Fortaleza, Ceará, BR, p. 203–217, 2012. Citado na página 56.

SOUZA, F. C. et al. Automating search strings for secondary studies. In: _____. **Information Technology - New Generations**. Las Vegas, Nevada, USA: Springer International Publishing, 2018. cap. 558, p. 839–848. Citado 2 vezes nas páginas 45 e 55.

Wasserman, A. I. Software engineering issues for mobile application development. In: **Proceedings of the FSE/SDP workshop on Future of software engineering research**. Santa Fe, New Mexico, USA: ACM, 2010. p. 397–400. Citado na página 21.

WILLOCX, M.; VOSSAERT, J.; NAESSENS, V. Comparing performance parameters of mobile app development strategies. In: IEEE. **2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems**. Austin, Texas, USA: ACM & IEEE, 2016. p. 38–47. Citado 6 vezes nas páginas 40, 48, 50, 52, 53 e 55.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: **In 8th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014**. New York City: ACM, 2014. p. 321–330. Citado na página 54.

WOHLIN, C. et al. **Experimentation in software engineering**. Berlin, Heidelberg, Dordrecht & New York City: Springer Science & Business Media, 2012. Citado na página 78.

WOODSIDE, M.; FRANKS, G.; PETRIU, D. C. The future of software performance engineering. In: **2007 Future of Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 171–187. ISBN 0-7695-2829-5. Citado na página 31.

ZEIN, S.; SALLEH, N.; GRUNDY, J. A systematic mapping study of mobile application testing techniques. **Journal of Systems and Software**, Elsevier Inc., v. 117, p. 334–356, 2016. Citado 4 vezes nas páginas 21, 40, 50 e 56.

Apêndices

APÊNDICE A – ESTUDO DE CASO

A.1 Termo de Consentimento

Termo de Consentimento Livre e Esclarecido

Você está sendo convidado(a) a participar, como voluntário(a), de um estudo de caso para avaliar o uso de um *framework* que oferece suporte para realizar teste de desempenho em aplicações móveis. Este estudo é parte integrante da pesquisa chamada "Proposta de um *framework* para realização de teste de desempenho em aplicações móveis", realizada pelo aluno Marcus Norberto e coordenada pelo Prof. Dr. Maicon Bernardino da Silveira e Prof. Dr. Fábio Paulo Basso.

Por meio deste documento, a qualquer momento, você poderá solicitar esclarecimentos adicionais sobre o estudo ou sobre a pesquisa. Também poderá retirar seu consentimento ou interromper a participação a qualquer momento, sem sofrer qualquer tipo de penalidade ou prejuízo. Ao participar deste estudo você não terá nenhum custo e nem receberá qualquer vantagem financeira. Seus dados pessoais serão mantidos em sigilo. Os resultados deste estudo serão armazenados pelo pesquisador responsável e poderão ser divulgados em publicações científicas. Não será divulgado o nome das aplicações utilizadas e o nome da empresa participante deste estudo.

Para participar deste estudo, você precisará: ouvir as explicações, ler o material fornecido, executar as atividades solicitadas e reportar sugestões de melhorias e críticas. Esperamos que os resultados deste estudo contribuam para a evolução do *framework*.

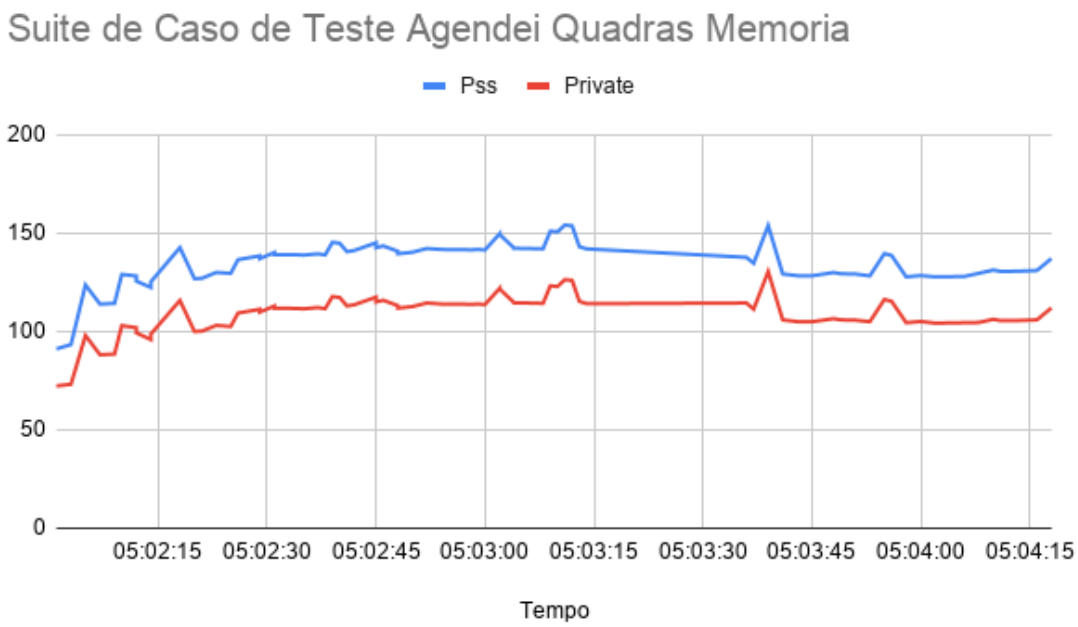
Após esses esclarecimentos, solicitamos o seu consentimento de forma livre para que participe deste estudo. Para tanto, preencha os itens a seguir.

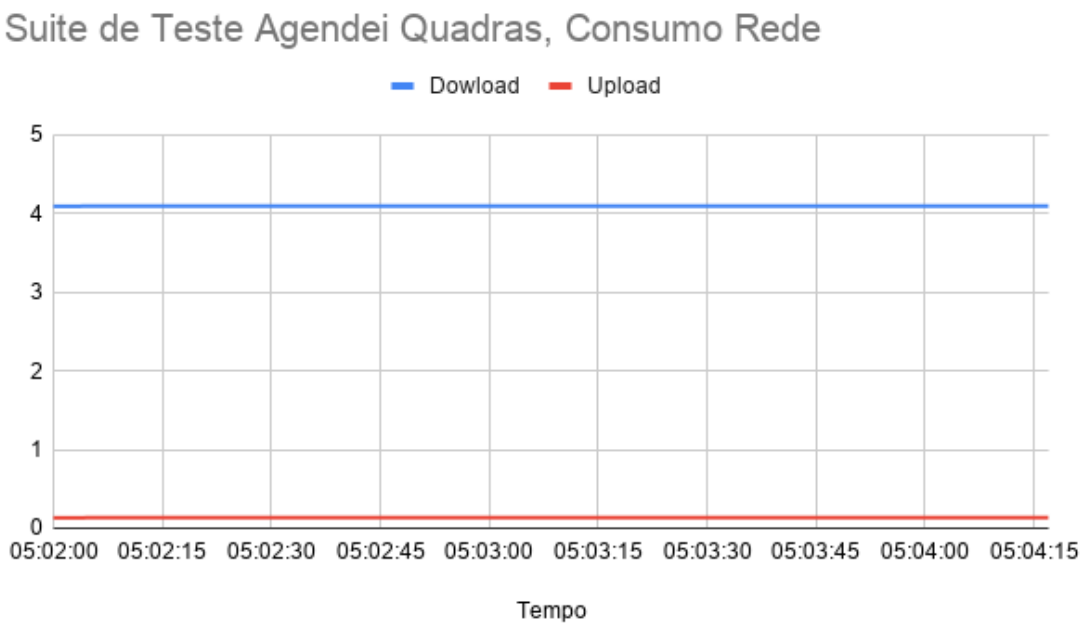
Eu, _____, tendo em vista as informações acima apresentadas, de forma livre e esclarecida, aceito participar deste estudo.

Alegrete, ____ de novembro de 2019.

_____
Assinatura do Participante

A.2 Relatórios




Suite de Teste Agendei Quadras, Consumo Rede
Dowload
Upload
5
4
3
2
1
0
05:02:00
05:02:15
05:02:30
05:02:45
05:03:00
05:03:15
05:03:30
05:03:45
05:04:00
05:04:15
Tempo


Suite de teste 1
Nov 13, 2019 05:01:31
Status
Dashboard
Search
realizaTutorial
Nov 13, 2019 05:01:56
Pass
realizaLogin
Nov 13, 2019 05:02:03
Pass
selecionaQuadra
Nov 13, 2019 05:02:09
Pass
reservaQuadra
Nov 13, 2019 05:02:25
Pass
editaPerfil
Nov 13, 2019 05:02:46
Pass
realizaCadastro
Nov 13, 2019 05:03:34
Pass
realizaCadastro
Nov 13, 2019 05:03:34
Nov 13, 2019 05:04:18
0h 0m 43s+729ms
Status
Timestamp
Details
5:03:34 AM
dispositivo : 420077960157947b
5:03:34 AM
Iniciou o teste
5:03:38 AM
Cadastra novo usuario
5:03:58 AM
Inserindo dados do local
5:04:08 AM
Inserindo senha
5:04:14 AM
Confirmando criação de usuario
5:04:18 AM
Consumo de Memória Principal TOTAL : 75MB

ÍNDICE