Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ferramentas de Geração de Testes existentes na literatura e na indústria: Uma meta-análise Multivocal Systematic Literature Review

Gabriel Matheus da Rocha de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Prof.a Edna Dias Canedo

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ferramentas de Geração de Testes existentes na literatura e na indústria: Uma meta-análise Multivocal Systematic Literature Review

Gabriel Matheus da Rocha de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof.a Edna Dias Canedo (Orientadora)
CIC/UnB

Prof. Dr. Fernando Albuquerque
CIC/UNB

Dr. Aleteia Patricia Favacho de Araujo
CIC/UNB

Prof. Dr. João Luiz Azevedo de Carvalho
Coordenador do Curso de Engenharia da Computação

Brasília, 11 de dezembro de 2023

Dedicatória

Dedico este trabalho primeiramente a meus pais e família que sempre me proporcionaram as melhores condições e incentivos para meu crescimento pessoal e acadêmico de forma a construir meu caminho de acordo com meus princípios e sonhos. Além disso, dedico aos meus amigos que encontrei durante a vida, que me proporcionaram momentos de descontração e apoio emocional. Da mesma forma, dedico o trabalho aos meus professores e orientadores que me acompanharam durante o ensino fundamental, médio e superior, que me proporcionaram inúmeros ensinamentos tanto acadêmicos quanto em foco, disciplina, determinação e evolução pessoal.

Agradecimentos

Agradeço especialmente aos meus familiares e amigos próximos pelos incentivos e apoio emocional durante diferentes etapas da vida. Aos professores e colegas do curso, agradeço pelos ensinamentos recebidos assim como pela união e companheirismo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Contexto: Ferramentas de teste de software são essenciais durante os processos de desenvolvimento e manutenção de aplicações digitais de qualidade em um cenário onde as mesmas se tornam cada vez mais complexas. O mercado disponibiliza centenas de ferramentas de geração de teste com diferentes abordagens e funcionalidades, é importante que desenvolvedores e projetistas possuam conhecimento de quais ferramentas são mais adequadas a seus trabalhos de forma a evitar gastos desnecessários e frustrações facilmente evitáveis. **Objetivo** Este trabalho tem como objetivo identificar as principais ferramentas disponíveis no mercado de forma a evidenciar suas vantagens e desvantagens apresentando assim um guia referencial que auxiliará desenvolvedores a escolher a ferramenta mais adequada para cada situação de teste. **Método** Neste trabalho foi realizado uma revisão multivocal da literatura para identificar trabalhos relacionados à ferramentas de geração de testes, identificando também as ferramentas mais citadas na literatura branca e cinza. Além disso, desenvolveu-se e conduziu-se um questionário com profissionais que atuam em diferentes áreas do desenvolvimento de software, a fim de averiguar a familiaridade deles em relação as ferramentas de teste identificadas na literatura branca e cinza, bem como as vantagens e desafios que possuem mais peso ao decidir qual ferramenta adotar. **Resultados** Os resultados mostraram que grande parte das ferramentas encontradas a partir da literatura branca, tais como Monkey e Dynodroid, não são muito bem conhecidas e utilizadas pelos participantes e que os mesmos possuíam mais familiaridade com ferramentas mais mencionadas na literatura cinza, como por exemplo, Postman e Selenium. Além disso, obteve-se uma lista de características mais desejadas e evitadas pelos participantes ao necessitarem escolher uma nova ferramenta para um projeto. **Conclusão** Os resultados obtidos foram compilados e utilizados para elaborar um Guia referencial onde os profissionais podem identificar as ferramentas mais recomendadas de acordo com um cenário específico, e em seguida utilizar uma tabela comparativa de forma a avaliar a ferramenta mais adequada para o projeto dentre as recomendadas. O Guia pode ser utilizado por qualquer profissional da área de testes.

Palavras-chave: Teste de Software, Ferramentas de Teste de Software, Ferramentas de

Geração de Teste, Comparação de Ferramentas de Teste, Revisão Multivocal da Literatura

Abstract

Context: Software testing tools are essential during the processes of developing and maintaining quality digital applications in a scenario where they become increasingly complex. The market offers hundreds of test generation tools with different approaches and functionalities, it is important that developers and designers have knowledge of which tools are best suited to their work in order to avoid unnecessary expenses and easily avoidable frustrations. **Objective** This work aims to identify the main tools available on the market in order to highlight their advantages and disadvantages, thus presenting a reference guide that will help developers choose the most appropriate tool for each test situation. **Method** In this work, a multivocal literature review was carried out to identify works related to test generation tools, also identifying the most cited tools in white and gray literature. Furthermore, a survey was developed and conducted with professionals working in different areas of software development, in order to ascertain their familiarity with the testing tools identified in the white and gray literature, as well as the advantages and challenges that have more weight when deciding which tool to adopt. **Results** The results showed that most of the tools found in the white literature, such as Monkey and Dynodroid, are not very well known and used by the participants and that they were more familiar with tools more mentioned in the gray literature, such as Postman and Selenium. In addition, a list of characteristics most desired and avoided by participants when needing to choose a new tool for a project was obtained. **Conclusion** The results obtained were compiled and used to prepare a reference guide where professionals can identify the most recommended tools according to a specific scenario, and then use a comparative table in order to evaluate the most appropriate tool for the project among those recommended. The Guide can be used by any testing professional.

Keywords: Software Testing, Software Testing Tools, Test Generation Tools, Testing Tools Comparison, Multivocal Literature Review

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

GL Gray literature - Literatura cinza.

GLM Gray Literature Mapping.

GLR Gray Literature Review.

MBT Model Based Teste.

MLM Multivocal Literature Mapping.

MLR Multivocal Literature Review.

SLM Systematic Literature Mapping.

SLR Systematic Literature Review.

Capítulo 1

Introdução

Software está entre os produtos mais amplamente usados na história da humanidade e ao mesmo tempo possui uma das maiores taxas de falha devido primariamente a sua pobre qualidade [9]. Softwares operam a maior parte dos sistemas manufaturados, mantém registros de virtualmente todos os cidadãos e operam a maioria dos automóveis, eletrodomésticos, sistemas médicos, operações governamentais, etc. Todas dependendo de um nível alto de qualidade para garantir seu funcionamento adequado [9]. Segundo Van et al. [10] o desafio da pesquisa em qualidade de software é fornecer ferramentas e tecnologia que permitirão à indústria de software implantar produtos e serviços de software que sejam seguros, confiáveis e utilizáveis dentro de uma estrutura econômica que permita às empresas competir de forma eficaz.

O processo de teste de software desempenha papel importante em alcançar e avaliar a qualidade de um produto de software aumentando a qualidade do mesmo ao repetir o processo de testar, encontrar defeitos e concertá-los durante todo o ciclo de desenvolvimento do software, assegurando o correto funcionamento do produto antes de seu lançamento [11]. O teste de software é um processo de verificação da análise e melhoria da qualidade do software [11].

O conceito de teste pode ser definido como a verificação dinâmica de que um programa fornece comportamentos esperados em um conjunto finito de casos de testes, adequadamente selecionados de um domínio de execução geralmente infinito [6]. O teste de software dessa forma consiste em um processo de aplicação de alguns critérios de teste de propósito geral bem definidos a uma estrutura ou modelo de software [12]. O seu escopo geralmente inclui a análise do código em execução em diferentes cenários e condições assim como a verificação dos aspectos: O código em questão realiza o que deveria fazer e o que precisa fazer? [13]. Devido a essa natureza, é importante que o processo de teste de software seja difundido durante todo o ciclo de vida de desenvolvimento e manutenção do mesmo para avaliar a qualidade da aplicação e reduzir o risco de falhas durante sua execução [6].

A realização do teste manipula grande quantidade de informações em um trabalho intensivo o que sem suporte pode ser árduo e propenso a erros [11]. Com o objetivo de oferecer alternativas mais eficazes e seguras, diferentes ferramentas de teste podem ser adotadas no projeto dentre uma grande variedade que oferece diferentes abordagens com vantagens e desvantagens de execução [11].

1.1 Problema de pesquisa

De acordo com Li e Yang [14] uma nova ferramenta adentrou a indústria em 2017 para integrar o vasto catálogo de ferramentas já existente em engenharia de software. O autor apresentou o DroidBot como uma ferramenta automatizada de geração de entradas para Android que é leve e compatível com a maioria dos aplicativos baseados no sistema Android. A ferramenta se diferencia das demais devido a sua técnica de operação que permite usuários integrarem suas próprias estratégias e algoritmos através da geração de entradas de testes guiadas pela interface do usuário com base em um modelo de transição de estado em tempo real.

O Droidbot é uma nova adição as ferramentas automatizadas de geração de entradas para Android mas que ao afirmar não precisar de instrumentar sistemas ou dispositivos é capaz de atuar em qualquer cenário como testes de compatibilidade e análise de malwares, cenários que as demais ferramentas não conseguem. O DroidBot é apenas uma das muitas ferramentas de geração de testes disponíveis mas pode ser a ferramenta ideal necessária para resolver um problema que desenvolvedores de Android a muito tempo tinham dificuldades para resolver mas que não tinham o conhecimento necessário para isso.

Desde 25 anos atrás até os dias atuais, o teste de software representa cerca de 50% do tempo total e mais de 50% do total gasto financeiramente em um projeto de desenvolvimento de software [15]. Uma pesquisa industrial de 2014 com 1.543 executivos de 25 países indica que testes e garantia de qualidade de sistemas intensivos de software representam aproximadamente 26% dos orçamentos de TI enquanto um estudo de 2013 da Universidade de Cambridge afirma que o custo global de localização e remoção de bugs de software aumentou para $312 bilhões anualmente [16].

A decisão de realizar um teste de software aborda uma análise de custo e benefício que se feita de forma equivocada alocará recursos inadequadamente o que pode gerar grande prejuízo. Para que o projeto de software evite riscos e perdas é necessário que os envolvidos possuam conhecimento e entendimento acerca de testes de software e sobre a melhor forma de realiza-los. O entendimento de cada situação e quais são as melhores ferramentas que podem ser usadas a seu dispor garante que o desenvolvedor responsável pelo teste

seja capaz de garantir a qualidade e bom funcionamento gerado pelo teste mas também assegurar que os recursos alocados para atividade foram devidamente manipulados.

1.2 Justificativa

Li e Yang [14] apresentaram uma ferramenta específica para sistemas Androids. Analogamente, porém de forma mais ampla, a indústria e a literatura de engenharia de software disponibilizam inúmeras ferramentas para serem utilizadas em diferentes cenários e sistemas. O conhecimento de como obter tais ferramentas, como utilizá-las e quais são as mais eficientes e indicadas para cada projeto é essencial para desenvolvedores que desejam mais flexibilidade e eficiência durante o trabalho, além de entregar produtos de alta qualidade e facilidade de manutenção.

Nestes tempos econômicos difíceis, os gerentes de desenvolvimento de software se esforçam para fazer mais e melhores testes com mais rapidez, a maioria reconhece que as ferramentas de teste automatizadas facilitam testes mais produtivos e de qualidade, mas a aquisição dessas ferramentas costuma ser complicada [17]. A automação de teste de software é um domínio orientado a ferramentas, reivindicado como a principal área de oportunidades de melhoria nas atividades de teste e exigindo investimentos em tempo, custo e esforço [18]. Ao iniciar ou pesquisar ferramentas de teste automatizado de software, é importante criar uma lista de requisitos a serem analisados ao escolher uma ferramenta para avaliação. Se não houver um direcionamento claro, é possível perder tempo baixando, instalando e avaliando ferramentas que atendem apenas a alguns requisitos ou podem não atender a nenhum deles [19].

Diante deste contexto, este trabalho irá identificar e apresentar os principais pontos relevantes em relação às ferramentas de geração de teste do mercado de forma a auxiliar gerentes de projetos e seus desenvolvedores a escolherem e adotarem as ferramentas mais adequadas para seus trabalhos. Dessa forma, evita-se altos investimentos iniciais desnecessários na seleção, configuração de automação e treinamento de desenvolvedores à ferramentas incapazes de realizar o trabalho desejado.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral desse trabalho é investigar as ferramentas utilizadas na academia e na indústria para realizar a geração de testes. Para isso, escolheu-se adotar o método de rea-

lizar uma revisão multivocal de literatura (MLR) com o intuito de identificar e sintetizar informações e dados relevantes em relação às ferramentas de geração de testes.

1.3.2 Objetivo Específico

Os seguintes objetivos específicos foram definidos de forma a direcionar os esforços para atingir o objetivo geral deste trabalho.

1. Realizar uma revisão multivocal de literatura para identificar as ferramentas de geração de testes existentes na literatura e na indústria;
2. Investigar as ferramentas de geração de testes utilizadas na indústria pelos profissionais da área, identificando os seus pontos fortes e fracos;
3. Identificar as principais dificuldades e desafios enfrentados pelos profissionais, por meio de um survey, ao executar testes de software e relacionar possíveis soluções com ferramentas de geração de testes disponíveis no mercado;
4. Analisar se há conformidade dos profissionais no que diz respeito ao uso das ferramentas identificadas;
5. Propor um guia referencial prático para apoiar os profissionais na seleção das ferramentas de geração de testes a serem utilizadas.

1.4 Resultados Esperados

Como resultado desse trabalho, será disponibilizado um guia contendo as ferramentas de geração de testes utilizadas na literatura e na indústria, para auxiliar desenvolvedores de software na seleção, avaliação e utilização de diferentes ferramentas de geração de testes durante as fases dos projetos de desenvolvimento de software.

1.5 Metodologia de Pesquisa

Uma revisão sistemática da literatura - Systematic Literature Review (SLR) - é "um meio de avaliar e interpretar todas as pesquisas disponíveis relevantes para uma questão de pesquisa específica, área de tópico ou fenômeno de interesse" (Kitchenham, 2004) [20]. Este trabalho adota a metodologia da revisão multivocal da literatura - Multivocal Literature Review (MLR) sendo esta um tipo de revisão sistemática da literatura que considera tanto a literatura acadêmica e a cinza em seu estudo. Para a execução desta abordagem utilizou-se o processo de MLR, conforme diretrizes definidas por Mendoza et

al. [1], com algumas adaptações e a adoção das estratégias de pesquisa Snowballing e Snowballing reverso. Na pesquisa Snowballing identifica-se artigos que citam um artigo incluído anteriormente no conjunto de estudo enquanto a pesquisa de Snowballing reverso identifica novos artigos a partir das referências de um artigo incluído anteriormente no conjunto [21].

As etapas de pesquisa adotadas neste trabalho são apresentadas na Figura 1.1 em que EO representa a quantidade de estudos iniciais, E1 a quantidade de estudos após o resultado do primeiro filtro, E2 o resultado após o segundo e E3 a quantidade final de estudos selecionados.

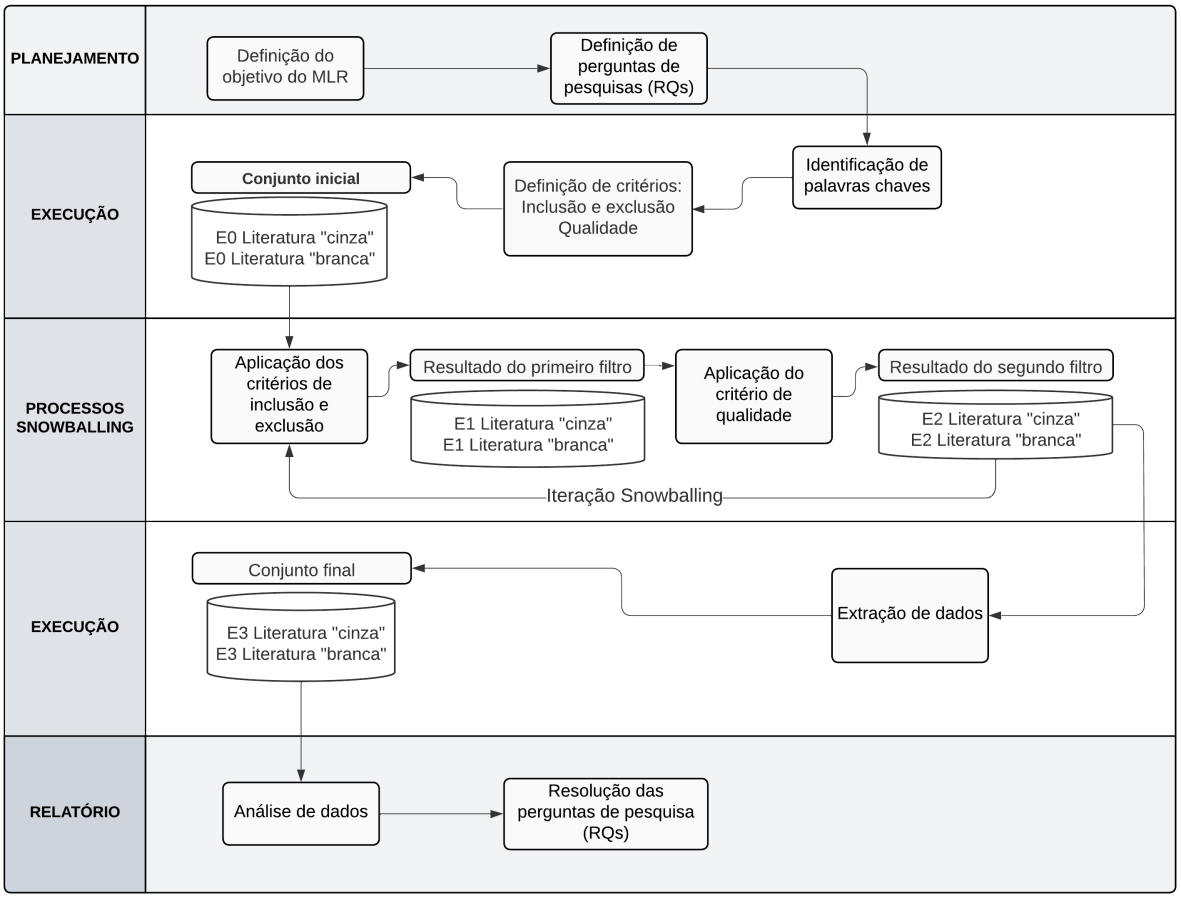

Figura 1.1: Metodologia de pesquisa adotada [1]

Afim de comparar os resultados obtidos na pesquisa, será utilizado um procedimento de survey para coleta de informações acerca da utilização de ferramentas de geração de teste por parte dos desenvolvedores. Os resultados obtidos pelos dois métodos são analisados de forma a verificar concordância e disparidade sobre as ferramentas mais utilizadas além das vantagens e desafios da utilização das mesmas. As etapas adotadas no processo de metodologia deste trabalho são:

1. Definição do protocolo a ser utilizado para conduzir a MLR
2. Definição das perguntas de pesquisa
3. Identificação de palavras-chave
4. Definição de critérios de inclusão, exclusão e qualidade
5. Realização dos processos de Snowballing e Snowballing reverso
   (a) Aplicação dos critérios de inclusão e exclusão
   (b) Aplicação dos critérios de qualidade
6. Extração de dados da pesquisa
7. Aplicação de um questionário com base nos resultados da pesquisa de forma a avaliar as perspectivas dos desenvolvedores sobre o assunto;
8. Análise dos resultados do questionário;
9. Resposta das perguntas de pesquisa
10. Proposição de um guia referencial sobre a utilização de ferramentas de geração de testes

A inclusão de literatura acadêmica e cinza neste método permite alcançar o objetivo proposto apresentando diferentes pontos de vista e experiências que se complementam preenchendo as lacunas de conhecimento geradas por cada literatura analisada separadamente.

1.6 Estrutura do Trabalho Final de Curso

O Capítulo 2 deste trabalho aborda o referencial teórico necessário para o completo entendimento do documento. Dessa forma, disponibiliza um breve histórico e explanação de conceitos relacionados a testes de software e de ferramentas de geração de teste além de uma perspectiva geral de como o assunto é encontrado na literatura e na indústria.

O Capítulo 3 detalha a metodologia MLR adotada no trabalho explicando os principais conceitos que a definem e a de termos necessários para sua compreensão. Além disso, detalha cada etapa dos procedimentos executados e o processo de obtenção e análise dos resultados.

O Capítulo 4 apresenta os resultados encontrados a partir dos processos de Snowballing e de Snowballing inverso assim como a síntese dos materiais encontrados e a análise acerca dos dados extraídos.

Capítulo 2

Referencial Teórico

Este capítulo apresenta definições e conceitos necessários para o completo entendimento do assunto abordado no trabalho além do referencial teórico utilizado como base para a pesquisa.

2.1 Breve histórico

Durante o curso da história existiram diversas definições e avanços no campo de teste de software. A Figura 2.1 ilustra a evolução descrita por meio de uma linha do tempo com os principais tópicos mencionados.

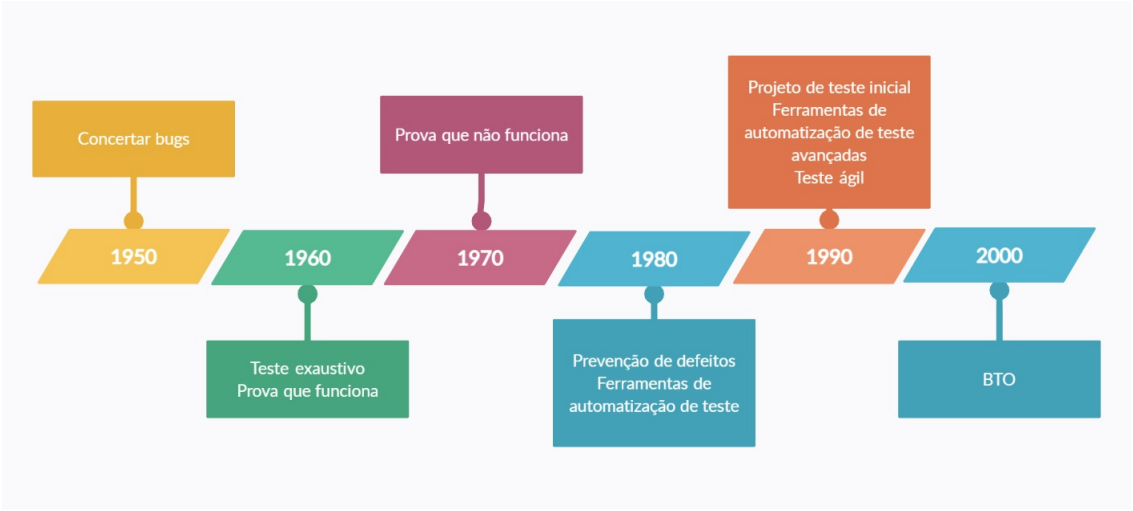

Figura 2.1: linha do tempo da evolução do teste de software [2]

A Seção 2.1 discute brevemente os pontos mais importantes da evolução desse campo, segundo Lewis e William [2]:

- Nos anos de 1950 testes de software eram definidos como: “o que desenvolvedores fazem para encontrar bugs em seus programas".

- No inicio da década de 1960 a ideia de teste intensivo considera os possíveis caminhos (paths) percorridos em um código ou a total variação possível de entradas para o programa. A ideia de testar completamente um software foi considerada teoricamente impossível devido a existência de diversos caminhos de entrada que possuem um domínio muito grande.

- Entre as décadas de 1960 e 1970 a definição sofreu uma revisão e se modificou para “o que é feito para mostrar a corretude de um programa"ou “o processo de estabelecer confiança de que um programa faz o que é suposto que faça".

- No início da década de 1970 foi proposto uma nova técnica de ciência da computação: “prova de corretude". O conceito era teoricamente promissor mas se demonstrou um método ineficiente de teste de software por ser insuficiente e consumir muito tempo. Em testes simples era fácil demonstrar que o software “funcionava"mas diversos defeitos permaneciam sem serem descobertos em implementações reais.

- No final da década de 1970 afirmou-se que o teste é o processo de executar um programa com a intenção de encontrar um erro e não de provar que o mesmo funciona. A existência das duas definições de testes (Prova de que funciona versus Prova de que não funciona) apresenta o “paradoxo do teste"com dois objetivos contraditórios: a) Gerar confiança de que o produto funciona bem e b) Descobrir erros no produto antes de avançar ou entrega-lo ao cliente. Atualmente o segundo objetivo é amplamente mais aceito entre os desenvolvedores uma vez que desenvolvedores que aceitem o primeiro podem inconscientemente ignorar defeitos para tentar provar que o software funciona corretamente.

- Na década de 1980 a definição de teste foi expandida para incluir prevenção de falhas. Promoveu a importância de testar não apenas o software mas seus requerimentos, design, código e os próprios testes além de sugerir que a adoção de uma metodologia de teste seja obrigatória e revisada durante todo o ciclo de desenvolvimento do produto.

- Nos meados de 1980 ferramentas de automação de testes surgiram de forma a automatizar o trabalho manual com o objetivo de realizar mais testes do que uma pessoa é capaz e de forma mais eficiente e confiável aumentando consequentemente a qualidade do produto alvo. Essas ferramentas eram primitivas e não possuíam recursos de linguagem script.

- No início da década de 1990 o teste de software foi redefinido como "planejamento, design, construção, manutenção e execução de testes e ambientes de testes"assumindo que um bom teste é um processo gerenciado durante todo o ciclo de vida. Nesse mesmo período surgiram ferramentas de teste mais avançadas de capture/replay que oferecem diversos recursos de geração de relatórios e linguagem script.

- Nos meados de 1990 softwares foram sendo desenvolvidos sem um modelo padrão de teste devido a popularidade da internet tornando-os muito mais difícil de testar. Durante esse período, a técnica de “teste ágil"(agile teste) surge para abordar esse problema se utilizando de testes exploratórios, testes rápidos e testes baseados em riscos.

Desde os primeiros dias dos primeiros programas de computador na década de 1960, a medida que os programas ficavam cada vez maiores, a necessidade de eliminar seus defeitos de forma sistemática recebia cada vez mais atenção. Com a comunidade de pesquisa e profissionais da área se aprofundando no conceito de teste de software, na década de 1970 um novo campo de pesquisa foi criado: teoria dos testes. A teoria enfatiza [15]:

- Detecção de defeitos por meio de testes baseados em execução

- Projeção de casos de testes de diferentes fontes (especificação de requisitos, código-fonte e domínios de entrada e saída de programas)

- Seleção de um subconjunto de casos de teste do conjunto de todos os casos de teste possíveis

- Eficácia da estratégia na seleção de casos de teste

- Utilização de oráculos de teste durante o teste (mecanismo que determina se um determinado teste teve sucesso ou falha)

- Priorização da execução dos casos de teste selecionados

- Análise de adequação de casos de teste.

A teoria dos testes herda a limitação fundamental do teste que segundo Dijkstra [22]: O teste pode apenas revelar a presença de erros, nunca sua ausência. Apesar disso, ao considerar a teoria dos testes, testadores e desenvolvedores de software são capazes de projetar casos de testes mais eficazes com um custo menor além de garantir uma perspectiva maior de detectar os defeitos de um programa de computador assegurando sua qualidade [15].

2.2 Conceitos de teste de software

Ao estudar testes de software é necessário previamente conhecer e entender as diferenças entre 3 conceitos essenciais dessa atividade: Defeito, erro e falha [23].

- Defeito: Ato inconsistente causado por pessoas físicas por exemplo, pelo mau uso de uma tecnologia. Pode ocasionar erros no produto.
- Erro: Diferença entre resultado obtido e esperado sendo a manifestação concreta de um defeito em um artefato de software. Uma falha pode ser resultada por diversos erros da mesma forma que erros podem não gerar falhas.
- Falha: Comportamentos inesperados do software que afetam diretamente o usuário final da aplicação podendo inviabilizar sua utilização.

Nem todas as entradas de um programa irão "acionar"um certo erro ocasionando uma falha e muitas vezes é complicado relacionar uma falha ao erro que a deu origem [23]. A análise desse problema leva ao modelo de Erro/Falha que afirma que quatro condições são necessárias para a observação de uma falha: Acessibilidade, Infecção, Propagação e Revelabilidade [24].

1. Acessibilidade: O teste deve alcançar o local ou locais no programa que contém a falha
2. Infecção: Após a localização ser confirmada o estado do programa deve estar incorreto
3. Propagação: O estado infectado deve se propagar pelo restante da execução de forma que a saída final do programa seja incorreta
4. Revelabilidade: O testador deve observar parte da porção incorreta do estado final do programa, caso observe apenas as partes corretas a falha não é revelada.

O processo de teste de software pode ser dividido em diferentes etapas: Planejamento, desenvolvimento dos casos de teste, execução dos casos de teste e avaliação dos resultados encontrados [25].

O planejamento de teste deve atender com precisão às necessidades da organização e também do cliente [26]. O Plano de Teste geralmente descreve estratégia de teste, escopo, objetivos, ambiente de teste, entregáveis do teste, riscos e mitigação envolvidos, cronograma, níveis de teste a serem aplicados, técnicas, métodos e ferramentas a serem utilizadas [26].

A essência de teste de software é determinar um conjunto de casos de teste para o item que será testado. Um caso de teste tem uma identidade própria e é associado a um comportamento do programa possuindo um conjunto de entradas e de saídas esperadas [25]. Um caso de teste completo possui um identificador, uma breve descrição de seu propósito, uma descrição das condições necessárias para a execução, as entradas para os testes de caso, as saídas esperadas, uma descrição das condições esperadas após a execução dos testes e um histórico de execução [25].

2.2.1 Níveis de teste

O teste de software geralmente é realizado em diferentes níveis ao longo dos processos de desenvolvimento e manutenção de um produto de software. Os níveis podem ser distinguidos com base no objeto do teste (alvo) ou no propósito (objetivo) [6]. O alvo do teste pode ser um único módulo, um grupo de módulos relacionados (propósito, uso, comportamento ou estrutura), ou um sistema inteiro. Dessa forma os três estágios de teste são classificados como: unidade, integração e sistema [6].

- O teste de unidade analisa elementos de software que podem ser testados separadamente verificando o funcionamento dos mesmos de forma isolada. Normalmente, o teste ocorre com acesso ao código que está sendo testado e com o suporte de ferramentas de depuração (processo de identificar e corrigir um erro descoberto). Segundo Myers et al.[27] é possível citar três grandes vantagens ao adotar o teste de unidade sendo elas:

    1. O teste auxilia o desenvolvedor como uma forma de gerenciar os elementos combinados do teste, uma vez que a atenção é focada inicialmente em unidades menores do programa.
    2. O teste facilita a tarefa de depuração uma vez que quando um erro é encontrado, sabe-se em qual unidade o mesmo se encontra.
    3. O teste introduz o paralelismo no processo de teste do software, apresentando a oportunidade de testar várias unidades simultaneamente.

- O teste de integração consiste em verificar a interações entre componentes de software. Estratégias modernas e sistemáticas de integração são tipicamente orientadas pela arquitetura, envolvendo a integração incremental dos componentes ou subsistemas de software com base em encadeamentos funcionais identificados.

- O teste de sistema objetiva testar o comportamento do sistema como um todo. Esse tipo de teste é mais adequado para avaliar os requisitos não funcionais do sistema

como segurança, velocidade e confiabilidade além de interfaces externas para outros aplicativos, utilitários, dispositivos de hardware ou ambientes operacionais.

O teste pode ser destinado a verificar diferentes propriedades. Os casos de teste podem ser projetados para verificar se as especificações funcionais foram implementadas corretamente, esse tipo de teste é comumente referido na literatura como teste funcional ou de conformidade [6]. Testes funcionais garantem que os recursos e funcionalidades do software estejam se comportando como esperado sem qualquer falha. Da mesma forma testes podem avaliar propriedades não funcionais como desempenho, confiabilidade, usabilidade e muitos outros, analisando aspectos que não são abordados nos testes funcionais. Diferentes níveis de teste abordam diferentes finalidades. A Tabela 2.1 apresenta os testes relacionados aos objetivos mais citados na literatura [6].

2.2.2 Técnicas de teste

Técnicas de teste de software são amplamente categorizados em dois tipos: estáticos e dinâmicos além de estratégias de caixa-preta, branca e cinza [28],[27], que serão abordadas em seguida.

No teste manual, os testadores executam manualmente os casos de teste. Qualquer produto deve ser testado manualmente antes de realizar o teste de automação [29]. É um processo lento e trabalhoso realizado pelo analista, desenvolvedor e equipe de teste [30]. Esse tipo de teste é preferível quando o alvo é um projeto de curto prazo e quando escrever um script torna-se demorado e custoso [29].

O teste dinâmico ou automatizado é mais rápido que o manual, neste tipo de teste testadores precisam escrever scripts de teste utilizando ferramentas automatizadas [31]. O teste automatizado permite ao testador a capacidade de criar cenários de teste repetíveis e reutilizáveis que podem ser executados sempre que necessário [31]. A Tabela 2.2 apresenta uma comparação entre testes manuais e automatizados seguindo a literatura [7]:

Duas das técnicas de teste de software mais conhecidas são: Teste funcional (Caixa preta) e Teste estrutural (Caixa branca). Segundo Shaukat et al.[28] e Myers et al.[27] as mesmas se comportam da seguinte forma:

- Teste de caixa preta (Black box): Teste que não se preocupa com os detalhes internos ou a estrutura de um produto de software. Dessa forma, se preocupa com a funcionalidade do produto, ou seja, se está funcionando corretamente ou não [28]. Nesta abordagem, os dados de teste são derivados apenas das especificações [27].

- Teste de caixa branca (White box): Estratégia de teste usada para testar o software se preocupando com a estrutura interna ou detalhes do produto de software.

Tipo de teste	Descrição
Teste de aceitação/Qualificação	Verifica se o sistema se comporta de acordo com os requisitos do cliente determinando se o mesmo satisfaz ou não o critério de aceitação.
Teste de instalação	Verificação do software após a sua instalação no ambiente destino .
Testes Alpha e Beta	Uso experimental do software por potenciais usuários (Alpha) ou um grupo maior de representantes (Beta). O teste é realizado antes do lançamento do produto e os usuários relatam os problemas encontrados durante a experimentação.
Teste de confiabilidade	Identifica e corrige falhas melhorando a confiabilidade do produto.
Teste de regressão	Teste que indica que o comportamento do software não é alterado de forma indesejada pelas alterações incrementais que sofre.
Teste de performance	Avalia características de desempenho como capacidade e tempo de resposta de forma a verificar se o software atende aos requisitos de desempenho.
Teste de segurança	Verifica a confidencialidade, integridade e disponibilidade dos dados do sistema de forma a verificar se o software está protegido contra ataques e ameaças externas.
Teste de stress	Determina os limites comportamentais e a defesa do sistema exercitando o mesmo em sua capacidade máxima e além dela.
Teste Back-to-back	Duas os mais versões de um programa são executadas simultaneamente com as mesmas entradas. As saídas são comparadas e analisadas em caso de discrepância.
Teste de recuperação	Verifica a recuperação do software após uma falha de sistema.
Teste de interface	Verifica se as interfaces do sistema realizam a troca correta de dados e informações de controle entre os componentes
Teste de configuração	Verifica o comportamento do sistema sob diferentes configurações especificadas
Teste de usabilidade	Avalia a facilidade com que usuários finais interagem e aprendem a utilizar o software

Tabela 2.1: Tipos de teste de acordo com suas funcionalidades [6]

Teste manual	Teste automatizado
Menos confiável uma vez que devido a possibilidade de erro humano não é sempre preciso	Mais confiável uma vez que executa o mesmo teste N número de vezes o que reduz o erro humano
O custo do teste depende dos recursos humanos utilizados	O custo do teste depende do custo das ferramentas de teste utilizadas
Devido a interferência humano o teste é mais lento consumindo mais tempo	Devido aos scripts de teste, o teste é mais rápido consumindo menos tempo
Utilizado quando dois ou três casos de teste precisam ser executados	Utilizado quando os casos de teste precisam ser executados repetidamente
Não garante que o usuário conseguirá utilizar a aplicação com facilidade	Garante que o usuário conseguirá utilizar a aplicação com facilidade
Também pode ser executado em paralelo mas aumenta o custo do teste	Pode ser executado em paralelo
Não é necessário ter conhecimento de programação para execução dos testes	Exige conhecimento de programação para a execução dos testes

Tabela 2.2: Comparação entre teste manual e automatizado [7]

Este teste garante que os resultados esperados correspondam aos resultados reais e permite descobrir os erros ocultos encontrados pelo software [28]. Essa estratégia deriva dados de teste a partir da análise da lógica do programa (Muitas vezes negligenciando a especificação) [27].

- Teste de caixa cinza (Grey box): Estratégia resultante da combinação de testes de caixa branca e preta sendo útil quando há poucos detalhes sobre a informação interna disponível. [28]. Nesta estratégia o testador depende da definição da interface e da especificação funcional em vez do código-fonte [32].

A seguir é apresentado um breve resumo das técnicas de teste atualmente conhecidas segundo Kaur et al.[33]

1. Teste aleatório: Os testes são gerados puramente ao acaso o que fornece uma abordagem relativamente simples para automação de teste. Essa é uma técnica de caixa-preta e os valores aleatórios podem ser gerados manualmente, bem como por um gerador de números pseudo-aleatórios.

    - Aleatório puro: Gera-se casos de teste aleatoriamente até que sua quantidade pareça ser suficiente.
    - Guiado pelo número de casos: Gera-se casos de teste aleatoriamente até que um determinado número específico de quantidade de casos seja atingido.

- Adivinhação de erro: Os casos de teste são gerados pelo conhecimento do desenvolvedor sobre erros típicos que ocorrem durante a programação. Os casos são gerados até que todos pareçam ter sido abordados

2. Teste Funcional: Enfatiza o comportamento externo da entidade de software sob teste, ou seja, o software em teste é visto como uma "caixa preta". Nesta técnica a seleção de casos de teste é baseada no requisito ou especificação de design do software em teste.

    - Particionamento por Equivalência: Utilizado para reduzir o número total de casos de teste para um conjunto finito de casos de teste. Dessa forma, divide o domínio de entrada de um programa em diferentes conjuntos de estados válidos ou inválidos para condições de entrada (classes de equivalência).
    - Análise de Valor de Contorno: Utilizada para identificar erros nos limites em vez de encontrar no centro do domínio de entrada. É semelhante à técnica de Particionamento por Equivalência mas utiliza o domínio de saída para criar os casos de teste.

3. Teste de fluxo de controle: Especifica uma sequência de atividades executadas por humanos e/ou aplicativos de software, geralmente representados por meio de notações gráficas. Cada sequência de ações constitui um fluxo de trabalho. O critério básico do fluxo de controle é a Cobertura de Declaração (SC), que exige que os dados de teste executem todas as instruções do programa pelo menos uma vez.

    - Cobertura de Decisão (DC): Todo desvio possível deve ser executado pelo menos uma vez. O critério DC trata uma decisão como um único nó na estrutura do programa.
    - Cobertura de Condição (CC): Cada condição em cada decisão tomou ambos os valores T (True - Verdadeiro) e F (False - Falso) pelo menos uma vez.
    - Critério D/CC: Cada decisão no programa teve todos os resultados possíveis pelo menos uma vez e cada condição em cada decisão teve todos os resultados possíveis em pelo menos uma vez.

4. Teste de Fluxo de Dados: É um tipo de teste estrutural que requer os detalhes da estrutura do programa. Possui ênfase nas variáveis que são definidas e utilizadas em diferentes pontos do programa, Utilizando o gráfico do programa para mapear os valores variáveis das variáveis dentro do programa.

5. Teste de mutação: Fornece um critério de teste chamado pontuação de adequação de mutação que pode ser usada para medir a capacidade de detectar falhas. As falhas

representam os erros cometidos por um programador, de modo que são introduzidos deliberadamente um conjunto de programas defeituosos chamados mutantes. Cada programa mutante é desenvolvido pela aplicação de um operador mutante ao programa original. Muitos operadores de mutação devem produzir mutantes equivalentes. O programa resultante é equivalente ao original e obtém a mesma saída que o original [34].

- Mutação forte: O número de mutantes gerados pode ser extremamente grande sendo muito custoso para ser usado diretamente na prática.
- Mutação fraca: Examina-se uma pequena porcentagem de mutantes selecionados aleatoriamente de cada tipo ignorando o restante. A técnica objetiva reduzir o custo de aplicação do teste.
- Mutação Seletiva: Examina-se apenas alguns tipos específicos de mutantes e ignora-se os outros.

6. Teste de regressão: Aplicado quando são feitas alterações em um software existente sendo realizado entre duas versões diferentes de software de forma a fornecer confiança de que as alterações recém-introduzidas não comprometam o comportamento da parte não alterada existente do software. Os métodos de teste de regressão podem usar um proxy para controlar o tamanho e o tempo de execução de um conjunto de testes para eliminar os testes que cobrem de forma redundante os requisitos do mesmo.

   - Técnicas de minimização: Utilizadas para selecionar conjuntos mínimos de casos de teste do conjunto de testes que fornecem cobertura de partes modificadas ou afetadas do programa.
   - Técnicas de fluxo de dados: Utilizadas para selecionar casos de teste que exercitam interações de dados que foram afetadas por modificações.
   - Técnicas seguras: Garantem a revelação de falhas no programa quando um determinado conjunto de condições de segurança pode ser satisfeito.
   - Técnicas ad hoc/aleatórias: Seleciona aleatoriamente um número predeterminado de casos de teste do conjunto de testes.
   - Técnica de Retestar Tudo: Reutiliza todos os casos de teste selecionando efetivamente todos os casos de teste no conjunto de testes.

2.3 Ferramentas de teste de software

Uma ferramenta de teste pode ser utilizada para testes de aplicativo da web, aplicativo de desktop, teste de aplicativo móvel ou combinação de dois aplicativos. Também pode envolver qualquer funcionalidade de teste, como teste de unidade, teste de regressão, teste de integração, etc. [35].

Há uma grande variedade de ferramentas de teste automatizadas disponíveis no mercado, sejam de código aberto ou comerciais. Usuários que conhecem as diferenças entre as opções são capazes de determinar a ferramenta de teste correta para cada ambiente [31]. Dessa forma, escolhendo ferramentas de software que realizam apenas um tipo específico de teste e limitadas a um tipo específico de linguagem ou optando por ferramentas que oferecem suporte a uma ampla gama de aplicativos, com melhores recursos e funcionalidades mas que podem exigir custos adicionais [31]. Além disso, o tamanho do projeto, o custo orçado para teste e a plataforma onde o projeto será usado também devem refletir nos critérios de seleção de uma ferramenta de teste [35].

2.3.1 Categorias de ferramentas

Para um melhor entendimento das características e funcionamento das ferramentas de teste é necessário compreender o significado de alguns termos novos sendo eles [28]:

- Ferramentas de código aberto (Open Source tools): Ferramentas que podem ser baixadas e instaladas facilmente de forma gratuita no computador. Estas ferramentas permitem realizar testes de aplicativos da Web e de desktop (por exemplo: Selenium).
- Ferramentas comerciais: Ferramentas de origem comercial que exigem pagamentos para serem utilizadas (por exemplo: Quick Test Professional)
- Ferramentas de avaliação comercial (Commercial trial): Ferramentas que estão disponíveis gratuitamente para uso durante a avaliação e, após a sua conclusão, exige pagamentos do usuário para que o mesmo continue podendo usá-las.

As ferramentas de testes podem ser classificadas de acordo com sua funcionalidade [6]. A Tabela 2.3 apresenta algumas dessas ferramentas além de uma breve descrição de cada funcionalidade. As ferramentas da mesma forma podem ser comparadas com base nos seguintes fatores segundo Singh et al. [36].

1. Recursos de gravação: Eficácia com que a ferramenta é capaz de registrar os casos de teste

2. Velocidade de execução: Pode ser determinada em termos do tempo médio de execução do teste do número de transações

3. Geração de Scripts: Como casos de teste podem ser exportados para códigos equivalentes em uma ampla variedade de linguagens. Esse fator avalia a capacidade da ferramenta de exportar código com um suporte a um grande conjunto de linguagens.

4. Teste orientado a dados: Possibilidade de importar dados por diversas fontes externas de arquivos, como por exemplo, de um arquivo excel.

5. Facilidade de Aprendizagem: Facilidade com que o usuário consegue explorar os recursos e utilizar toda a funcionalidade da ferramenta.

6. Relatórios/saída de teste: Capacidade de apresentar resultados em um formato adequado de fácil leitura e compreensão para o testador.

Ferramentas disponíveis	Funcionalidade
"Arreios"de teste	Fornecem um ambiente controlado no qual os testes podem ser iniciados e as saídas do teste podem ser registradas
Geradores de teste	Fornecem assistência ao gerar casos de teste. Pode ser aleatória, baseada em caminho, baseada em modelo ou uma mistura dos mesmos.
Ferramentas de captura/reprodução	Executam novamente ou reproduzem automaticamente testes executados que registraram entradas e saídas anteriormente.
Oracle/ferramentas de verificação de assertivas	Ajudam a decidir se um resultado de teste é bem-sucedido ou não.
Analisadores de cobertura e instrumentadores	Avaliam quais e quantas entidades do gráfico de fluxo do programa foram exercidas entre todas aquelas exigidas pelo critério de cobertura de teste selecionado.
Rastreadores	Registram o histórico dos caminhos de execução de um programa.
Ferramentas de teste de regressão	Suportam a reexecução de um conjunto de testes após a modificação de uma seção do software.
Ferramentas de avaliação de confiabilidade	Suportam a análise de resultados de teste e visualização gráfica para avaliar medidas relacionadas à confiabilidade de acordo com modelos selecionados.

Tabela 2.3: Classificação das ferramentas [6]

2.3.2 Ferramentas de geração de testes

Para realizar testes de software é necessário projetar casos de teste eficientes focados em locais onde os bugs podem estar escondidos, o que demanda muito tempo e esforço ao ser executada manualmente [37]. Com o objetivo de melhorar a eficiência dos casos de teste ,e por consequência do teste como um todo, é possível utilizar uma ferramenta de geração automática de casos de teste para agilizar e facilitar o processo [37].

Segundo Hartman [38] "Indiscutivelmente, toda atividade de teste de software é baseada em modelo, já que qualquer caso de teste deve ser projetado usando algum modelo mental da aplicação em teste". O teste baseado em modelo - Model Based Teste (MBT) - é uma técnica que gera testes de software a partir de descrições explícitas do comportamento de um aplicativo. Criar e manter um modelo de um aplicativo facilita a geração e atualização de testes para o mesmo [39]. Um modelo é uma descrição do comportamento de um sistema e como é mais simples do que os sistema que descreve, o mesmo possibilita prever o comportamento do sistema além de facilitar seu entendimento. Um modelo de estado finito consiste em um conjunto de estados, um conjunto de eventos de entrada e as relações entre eles. Dado um estado atual e um evento de entrada, é possível determinar o próximo estado atual do modelo [39].

Um gerador de teste baseado em modelo pode ser definido como um processo automatizado que aceita duas entradas principais: a) Um modelo formal do software em teste e b) Um conjunto de diretivas de geração de testes que orientam a ferramenta em sua geração [38]. A saída de um gerador de teste é um conjunto de casos de teste que inclui uma sequência de estímulos para o sistema em teste e as respostas esperadas para os respectivos estímulos, conforme previsto pelo modelo [38]. É importante notar que as diretivas de geração de teste não são fornecidas explicitamente para muitas das ferramentas no mercado, mas são "conectadas"à estrutura das mesmas.

Exemplos de ferramentas de geração de testes baseado em modelo:

- AETG: É usado em uma variedade de aplicações para testes de unidade, sistema interoperabilidade, gera planos de teste de alto nível e casos de teste detalhados. AETG usa novos algoritmos combinatórios para gerar conjuntos de teste que cobrem todas as combinações válidas de parâmetros n-way. O tamanho de um conjunto de teste AETG cresce logaritmicamente no número de parâmetros de teste permitindo que os testadores definam modelos de teste com dezenas de parâmetros que podem ser usados como por exemplo como parâmetros de configuração do sistema, entradas do usuário e outros eventos externos [40].

- DGL: Linguagem baseada no conceito de gramáticas livres de contexto que pode ser usada para gerar dados de teste de vários tipos podendo ser utilizada para gerar

testes tanto para software quanto para hardware. Os testes podem ser gerados de forma sistemática, aleatória ou uma combinação de ambos [41].

O teste baseado em caminho é uma abordagem de caixa branca considerada teoricamente mais rigorosa e eficaz na detecção de erros do que outros critérios comuns de teste de software [42]. Essa abordagem exercita um conjunto básico de caminhos de execução, ou seja, um conjunto finito de caminhos de execução linearmente independentes, que podem ser usados para construir qualquer caminho de execução arbitrário através de um programa. Dessa forma, testar um conjunto básico de caminhos de execução por meio de um programa garante o teste de todos os caminhos de execução possíveis por meio do mesmo [42].

O teste baseado em caminho usa a complexidade ciclomática (mensura a complexidade de um determinado módulo (uma classe, um método, uma função etc), a partir da contagem do número de caminhos independentes que ele pode executar até o seu fim.) e a análise matemática dos gráficos de fluxo de controle para orientar o processo de teste de software. O gráfico de fluxo de controle apresenta um modelo básico de todos os caminhos de execução através de um programa, os nós representam declarações ou expressões computacionais e as arestas representam a transferência de controle entre os nós. Cada caminho de execução possível do ponto de entrada ao ponto de saída em uma unidade de programa tem um caminho correspondente em seu grafo de fluxo [42]. A Figura 2.2 representa um exemplo de gráfico de controle de fluxo ilustrando caminhos e decisões em um caso fictício de login em um aplicativo:

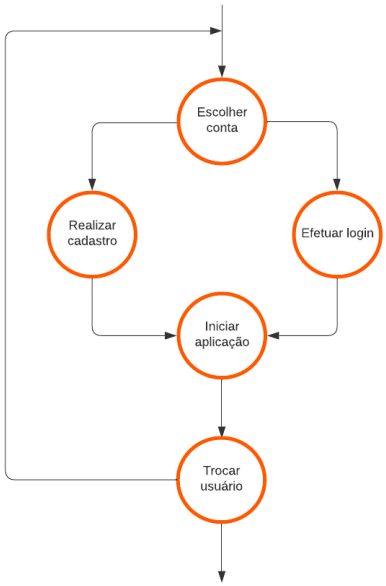

Figura 2.2: Exemplo de gráfico de controle de fluxo[3]

2.4 Trabalhos correlatos

O estudo deste trabalho foi inspirado em diferentes literaturas que comparam diversas ferramentas de geração de testes disponíveis na literatura. A ideia de tais documentos gerou uma base para a ideia de realizar uma pesquisa envolvendo tanto literatura branca quanto cinza de forma a analisar e comparar inúmeras ferramentas de geração de testes em diferentes cenários e pontos de vista indicando a melhor situação para a utilização de cada uma.

Wang et al. [43] apresentaram os resultados obtidos em um experimento que comparou três ferramentas conhecidas de geração de dados de teste de unidade acessíveis ao público, JCrasher, TestGen4j e JUB. Os autores criaram dois conjuntos de testes em Java sendo que um continha valores aleatórios e o outro valores para satisfazer a cobertura de borda e aplicando os nessas ferramentas compararam os resultados as avaliando com base em suas pontuações de mutação. Os resultados mostraram que as ferramentas automáticas de geração de dados de teste geraram testes com quase as mesmas pontuações de mutação que os testes aleatórios.

Inúmeras ferramentas de geração de teste foram desenvolvidas para oferecer suporte a testes de interface do usuário em aplicativos móveis, especialmente para aplicativos Android. Nesse cenário, Wang et al. [44] estudaram as ferramentas de geração de testes de última geração ou práticas existentes em 68 aplicativos industriais amplamente utilizados os comparando diretamente em relação à cobertura de código e capacidade de detecção de

falhas. O resultado indicou que a ferramenta do Google "Monkey"atinge a maior cobertura de método em 22 dos 41 aplicativos cujos dados de cobertura de método puderam ser obtidos.

Shafique et al. [45] conduziram uma revisão sistemática cujo objetivo foi determinar o estado da arte atual do suporte proeminente de ferramentas MBT, se concentrando em ferramentas que dependem de modelos baseados em estado. Os resultados buscam ajudar empresas de software interessadas em selecionar a ferramenta MBT mais adequada para suas necessidades e organizações dispostas a investir na criação de suporte à ferramenta MBT.

Na literatura, é possível encontrar ferramentas de pesquisa para gerar automaticamente casos de teste para APIs RESTful mas nenhuma comparação direta dessas ferramentas está disponível para orientar os desenvolvedores na decisão de qual ferramenta se adapta melhor ao seu projeto de API REST. Corradini et al. [46] apresentaram os resultados de uma comparação empírica das abordagens automatizadas de geração de casos de teste de caixa preta: RestTestGen, RESTler, bBOXRT e RESTest em termos de robustez e cobertura de teste. Ao utilizar a ferramenta para gerar casos de teste para 14 serviços REST, chegaram a conclusão de que a ferramenta RESTler parece ser a mais sólida sendo a única capaz de testar com sucesso todos os estudos de caso sem apresentar falhas.

2.5 Síntese do capítulo

Este capítulo iniciou apresentando um histórico do teste de software e pela explicação de conceitos básicos importantes para o entendimento do funcionamento sobre testes. Em seguida, apresentou informações acerca das diferentes abordagens e características do processo de teste de software. Por fim, apresentou as diferentes classificações e categorizações presente entre as ferramentas de teste com um foco nas ferramentas voltadas a geração de casos de teste.

Capítulo 3

Revisão Multivocal de Literatura

Este trabalho realiza uma Revisão Multivocal da Literatura para identificar as Ferramentas de geração de testes. Com mais de 300 ferramentas de teste automatizadas no mercado é necessário que o profissional expanda a área de experiência a medida que essas ferramentas crescem e amadurecem, sabendo identificar a ferramenta certa para o teste certo, instalar a ferramenta e garantir que a mesma esteja funcionando corretamente [47]. O estudo leva em consideração artigos científicos publicados a partir de 2015 e outras fontes de informação consideradas como literatura cinzenta que abordam ferramentas de geração de testes.

Este Capítulo oferece uma visão geral da metodologia de pesquisa utilizada e em seguida uma visão geral da abordagem sistemática utilizada para reunir a literatura considerada relevante para o projeto. Para a revisão multivocal de literatura e posterior discussão acerca do tema, utilizou-se bibliotecas digitais: ACM Digital Library, Science Direct, Springer Link, IEEE Xplore Digital Library e Google Scholar. O estudo objetiva identificar as vantagens e desafios da utilização de ferramentas de geração de testes e, com base nos resultados encontrados, propor um guia referencial para auxiliar os profissionais de software diante dos obstáculos de utilizar tais ferramentas em seus projetos.

3.1 Conceitos da revisão multivocal de literatura

3.1.1 Literatura cinza

A definição mais aceita do conceito de literatura cinza é a acordada na Terceira Conferência Internacional realizada em Luxemburgo, novembro de 1997: “Aquilo que é produzido em todos os níveis do governo, acadêmicos, empresas e industrias em formatos eletrônicos e impressos não controlados por editores comerciais"[48],[49]. Dessa forma, a produção da literatura cinza é de uma fonte para a qual a publicação não é a atividade principal

como relatórios anuais, noticias, apresentações, vídeos, e etc, com credibilidade e controle de saída moderado. Adams [4], apresentou um modelo que classifica os diferentes tipos de fonte de Gray literature - Literatura cinza (GL). O modelo apresenta duas dimensões: especialização e controle de saída, mostrando uma gradação entre os extremos "desconhecido"e "conhecido"formando os tons de cinza. Esse modelo de tons de cinza mostra o aspectos da literatura "branca", "cinza"e "preta", a literatura branca indica que a experiência e o controle de saída são totalmente conhecidos, a cinza possui controle e credibilidade moderados enquanto a preta é a menos segura consistindo em ideias, conceitos e pensamentos. A Figura 3.1 representa o modelo visualmente.

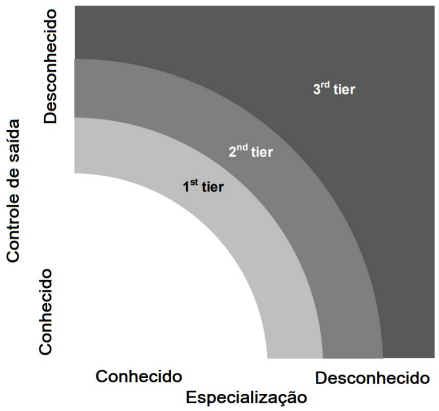

Figura 3.1: Modelo dos níveis de literatura cinza [4]

3.1.2 Estudos secundários

Um estudo secundário é definido como um estudo de estudos, ou seja, uma revisão de estudos individuais ou primários sendo bem comum em engenharia de software [50]. Estudos secundários geralmente não geram novos dados como um estudo primário mas analisam um conjunto de estudos primários agregando os resultados encontrados para fornecer evidências concretas acerca de um certo fenômeno em especifico. De acordo com Gorousi et al. [50], os estudos secundários em engenharia de software podem ser categorizados em seis tipos sendo eles: Mapeamentos Sistemáticos da Literatura - Systematic Literature Mapping (SLM), Revisão Sistemática da Literatura - Systematic Literature Review (SLR), Mapeamento da Literatura Cinza - Gray Literature Mapping (GLM), Revisão da Literatura Cinza - Gray Literature Review (GLR), Mapeamento Multivocal da Literatura - Multivocal Literature Mapping (MLM) e Revisão Multivocal da Literatura - Multivocal Literature Review (MLR). Neste trabalho realizaremos um MLR.

3.1.3 Benefícios de incluir literatura cinza

A literatura cinza pode aumentar a abrangência sobre o estudo fornecendo dados não publicados na literatura publicada comercialmente, reduzir o viés da publicação e promover uma imagem equilibrada das evidências disponíveis [51]. O viés da publicação consiste na propensão de publicar apenas estudos que possuírem achados e resultados positivos, como a literatura cinza pode obter e registrar resultados neutros e negativos sua inclusão fornece uma compreensão mais equilibrada e precisa do tamanho dos efeitos discutidos reduzindo a chance de haver distorção nos resultados da metanálise e da revisão sistemática [51]. Apesar das vantagens, foi relatado que apenas 31% das meta-análises publicadas incluem literatura cinzenta [52]. Essa omissão pode ocorrer porque a natureza da literatura cinzenta torna sua exclusão mais conveniente; é difícil de recuperar, frequentemente incompleto e sua qualidade pode ser difícil de avaliar [53].

3.2 Planejamento

A revisão multivocal da literatura (MLR) é uma forma de revisão sistemática (estudo que analisa e sintetiza informações de estudos existentes gerando novos resultados e conclusões) que produz dados qualitativos e quantitativos permitindo uma visão mais ampla do estado atual do assunto pesquisado a partir da consideração da literatura cinza [54]. Assim como as revisões sistemáticas da literatura, MLRs são usadas para enfrentar o problema do viés do pesquisador procurando encontrar o máximo possível de pesquisas relevantes usando métodos explícitos para verificar antecedentes e revisar a literatura de materiais e pesquisas anteriores [54][55]. Para que a revisão da leitura seja eficiente a mesma deve garantir que o campo abordado seja corretamente compreendido fornecendo assim uma boa base para futuras pesquisas.

3.2.1 Perguntas de pesquisa

Ao realizar um processo de revisão de literatura inicialmente identifica-se a necessidade de se realizar tal revisão [56]. No caso deste trabalho essa necessidade é a formulação de um documento guia sobre ferramentas de geração de teste.

Este trabalho objetiva identificar e sintetizar informações relevantes em relação às ferramentas de geração de testes de forma a gerar um guia prático aos profissionais da área interessados. Para que seja possível que os leitores entendam mais facilmente e corretamente interpretem o estudo, Easterbrook et al. [57] indicaram a formulação de perguntas de pesquisa de forma a direcionar a pesquisa. Dessa forma, foram definidas 3 perguntas de pesquisa (Research Question - RQ):

- **RQ1**: Quais as ferramentas de geração de testes utilizadas na literatura?
- **RQ2**: As ferramentas de geração de testes identificadas na literatura são usadas na indústria?
- **RQ3**: Quais as vantagens e desafios do uso das ferramentas de geração de teste?

As perguntas de pesquisa guiarão a revisão multivocal da literatura conduzida nesse trabalho. De forma a responder a **RQ1**, será criado uma tabela e um gráfico indicando as ferramentas de geração de testes mais repetidamente mencionadas nos artigos selecionados. Para a **RQ2** e **RQ3** será realizado um survey com os profissionais da área de software visando coletar as informações necessárias para que em conjunto com os resultados da pesquisa realizada, seja possível responder as questões satisfatoriamente. O survey será descrito na Seção 3.4.

3.2.2 Critério de inclusão

Para que um estudo seja incluído, ele deve estar escrito em Português ou Inglês e abordar sucintamente a temática de ferramentas de geração de testes. O processo de inclusão dos estudos foi realizado em três etapas em que um artigo apenas é qualificado a seguir para a seguinte caso o mesmo cumpra o critério de inclusão da atual. As etapas são:

1. Identificação de trabalhos a partir dos processos de Snowballing e Snowballing reverso: O artigo avaliado deve ter sido publicado a partir do ano de 2012.
2. Leitura do abstract, título e palavras chaves: A leitura das sessões citadas permite uma rápida compreensão do assunto abordado e concede uma ideia geral do escopo do artigo permitindo assim verificar se o estudo em questão aborda os temas desejados para a pesquisa e o rápido descarte do mesmo caso a resposta seja negativa.
3. Leitura do trabalho na íntegra: O artigo avaliado deve demonstrar que sua inclusão ao conjunto da pesquisa traz certa novidade e contribui de alguma forma para a resolução de no mínimo uma pergunta de pesquisa. Dessa forma, adotou-se que o estudo deve abordar ao menos um dos tópicos:
    - Ferramentas de geração de teste disponíveis;
    - Vantagens e Desafios ao utilizar ferramentas de geração de testes;
    - Relação entre desenvolvedor e ferramentas de geração de testes.

A Tabela 3.1 apresenta o procedimento da aplicação dos critérios de inclusão:

Etapa	Critérios de inclusão
Identificação de trabalhos a partir do processo de Snowballing e Snowballing inverso	Documentos publicados a partir de 2012
Leitura dos resumos e trechos que foram citados ou que citam	Aderência às temáticas propostas
Leitura do trabalho na íntegra	Abordar ao menos um dos tópicos mencionados

Tabela 3.1: Critérios de inclusão adotados no Snowballing

3.2.3 Critério de exclusão

Além de materiais que não se qualificam nos critérios de inclusão como documentos publicados antes de 2012, foram excluídos literatura inacessível, ou seja não disponível para a comunidade, resultados que a pesquisa do *Google* considera semelhantes, anúncios de ferramentas de fornecedores.

3.2.4 Avaliação de qualidade

Segundo Garousi et al. [8], a avaliação de qualidade dos estudos consiste em determinar até que ponto um estudo é valido e livre de viés. A literatura formal geralmente segue um processo controlado de revisão e publicação, como a literatura cinza é mais variada e menos controlada a qualidade dessa literatura é mais diversa e possivelmente complicada de avaliar sendo necessário abordar um modelo para avaliação de sua qualidade. Para avaliar a qualidade de estudos da literatura cinza Garousi et al. [8] sugere aplicar e adaptar certos critérios de avaliação sendo eles: autoridade do produtor, metodologia, objetividade, data, novidade e tipos de qualidade, os quais foram adaptados para esse estudo, através das seguintes perguntas:

1. **Autoridade do produtor**: A organização ou indivíduo que publicou o estudo possui credibilidade? ou seja, possuiu uma boa reputação, publicou outros trabalhos na área e/ou possui experiência na área? Por exemplo: Software Engineering Institute (SEI).

2. **Metodologia**: O estudo tem um objetivo claramente declarado? O estudo é apoiada por fontes e referências confiáveis? O estudo responde alguma das perguntas de pesquisa estabelecidas?

3. **Objetividade**: As afirmações do estudo são as mais objetivas possíveis ou são de opiniões subjetivas? ou seja, as informações providas pelo estudo são apresentadas e apoiados por dados concretos?

4. **Data de publicação**: O estudo possui uma data de publicação claramente indicada?

5. **Novidade**: O estudo adiciona algo novo e único a pesquisa? retifica ou ratifica algum outro estudo da literatura?

6. **Tipos de qualidade**:

    - 1º nível: Controle de saída alta/Alta credibilidade: Livros, revistas, teses, relatórios governamentais
    - 2º nível: Controle de saída moderado/ Credibilidade moderada: relatórios anuais, artigos de notícias, apresentações, vídeos, sites de perguntas e respostas, artigos wiki
    - 3º nível: Controle de saída baixo/ Baixo credibilidade: blogs, e-mails, tweets Garousi et al. [8].

Cada estudo de literatura cinza é avaliado pelas perguntas mencionadas e recebe um ponto para cada resposta "sim"podendo obter um total de 10 pontos de acordo com os critérios autoridade do produtor, metodologia, objetividade, data e novidade. Adicionalmente, cada fonte recebe 0, 0,5 ou 1 ponto caso seja nível 3, 2 ou 1 respectivamente. De forma a estabelecer uma avaliação de qualidade sistemática para todas as fontes de literatura cinza, foi definido que a pontuação de qualidade de 6 é o “limiar”. Qualquer fonte acima disso é incluída no estudo enquanto as demais com pontuação abaixo dela são excluídas. A Tabela 3.2 exemplifica uma análise de qualidade considerando três literaturas cinzas fictícias sendo elas: GL1: Postagem em um blog, GL2: Revista e GL3: Vídeo do Youtube.

Ao analisar a Checklist verifica-se que GL1 foi excluído enquanto GL2 e GL3 foram incluídos no processo.

Critério	Questões	GL1	GL2	GL3
Autoridade do produtor	A organização ou indivíduo que publicou o estudo possui credibilidade? ou seja, possuiu uma boa reputação e/ou possui experiência na área?	0	1	0
	A organização ou indivíduo publicou outros estudos nessa mesma área de conhecimento?	1	1	1
Metodologia	O estudo tem um objetivo claramente declarado?	0	1	1
	O estudo é apoiado por fontes e referências confiáveis?	1	1	0
	O estudo responde alguma das perguntas de pesquisa estabelecidas?	0	0	1
Objetividade	As afirmações do estudo são as mais objetivas possíveis ou são de opiniões subjetivas?	0	1	0
	As conclusões são apoiadas pelos dados?	0	1	0
Data de lançamento	O estudo possui uma data de lançamento claramente indicada?	1	1	1
Novidade	O estudo adiciona algo novo e único a pesquisa?	0	0	1
	O estudo retifica ou ratifica algum outro estudo da literatura?	0	1	1
Tipo de qualidade		0	1	0.5
Soma total		3	9	6.5

Tabela 3.2: Checklist de análise de qualidade da literatura cinza [8]

3.3 Execução da pesquisa

Após a definição do protocolo a ser seguido, deu-se inicio a execução da revisão multivocal e aquisição de materiais de pesquisa. O processo de identificação e seleção de novos estudos a serem incluídos neste trabalho foi realizada por meio das abordagens de snowballing forward e inverse snowballing em um conjunto inicial de artigos selecionados. O conjunto inicial neste caso consiste em apenas um único documento sendo esse o artigo de Li e Yang [14]. A abordagem de bola de neve seguida foi abordada seguindo o estudo de Wohlin et al. [5]:

O processo de Snowballing reverso consiste em usar a lista de referências para identificar novos artigos a serem incluídos. O primeiro passo dessa abordagem é percorrer a lista de referências e excluir artigos que não atendam a critérios estabelecidos, e em seguida, remover os papéis da lista que já foram examinados nesta ou em uma iteração anterior. As duas primeiras etapas na bola de neve para trás foram utilizadas para extrair o máximo de informações possíveis do documento que estava sendo examinado não havendo transferência para novos artigos até que não havia mais informações disponíveis no estudo que está sendo examinado. Por outro lado, Snowballing para frente refere-se à identificação de novos artigos com base naqueles artigos que citam o artigo que está sendo examinado contudo, possui uma abordagem para examinar os papéis semelhante à abordagem utilizada pela bola de neve para trás [5].

Ao identificar um estudo que ao ser submetido aos critérios de inclusão se tornou qualificado a adentrar a pesquisa, o mesmo foi lido integralmente. Primeiramente, leu-se o resumo e as partes mais relevantes do artigo após breve navegação pelo documento para poder tomar uma decisão sobre inclusão ou exclusão de maneira eficiente. Após o snowballing e o snowballing reverso, os novos artigos identificados na iteração são colocados em uma pilha para ir para a próxima iteração, como uma iteração foi realizada por vez, foi possível obter rastreabilidade entre os documentos [5]. A Figura 3.2 ilustra as abordagens de Snowballing adotadas.

Segundo Garousi et al.[8], a condição de parada do processo de busca de literatura cinza não é tão clara quanto a da pesquisa formal que segue uma cadeia de pesquisa. Os autores sugerem três diferentes critérios de parada sendo elas:

1. Saturação teórica: Quando nenhum novo conceito emerge dos resultados da pesquisa
2. Esforço limitado: Incluir apenas os principais N resultados do mecanismo de pesquisa
3. Esgotamento de evidências: Extrair todas as evidências

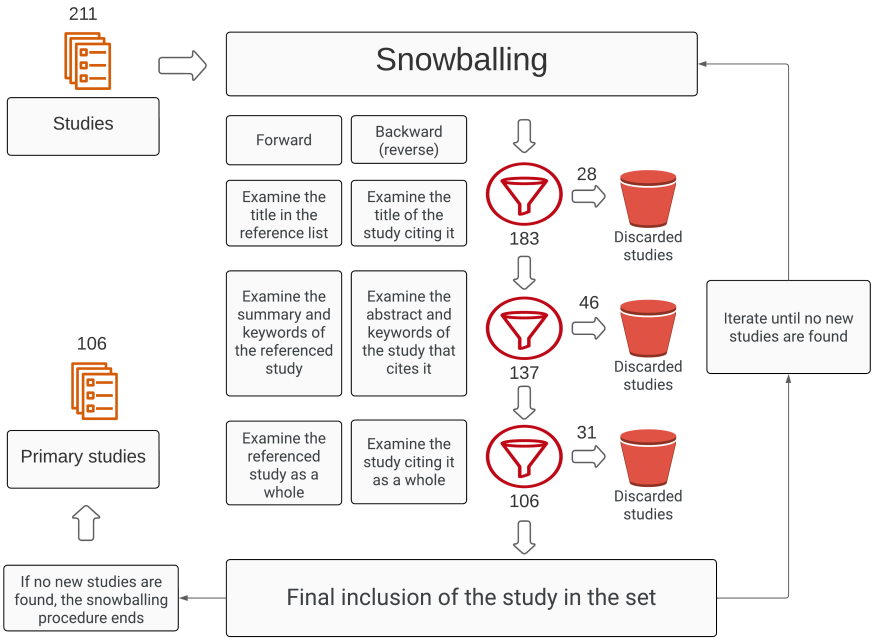

Figura 3.2: Etapas do procedimento de snowballing para frente e para trás [5]

Este trabalho adota a abordagem 1, ou seja, o processo de busca foi encerrado quando ao pesquisar por novos estudos não obteve-se nenhum conteúdo novo e relevante acerca de ferramentas de testes para adicionar a este trabalho que já não tenha sido incluído anteriormente.

3.4 Elaboração do Survey

O survey realizado teve como objetivo coletar informações de profissionais da área de software em relação a quais ferramentas de geração de testes são utilizadas por eles e quais são as principais vantagens e desafios de utilizá-las. Desenvolvedores de software com perfis variados, tanto em idade, quanto em experiência profissional foram convidados a participar a pesquisa com o único requisito necessário sendo que o mesmo deve ser desenvolvedor ou um projetista de software.

A divulgação da pesquisa ocorreu primariamente por meio da rede social LinkedIn uma vez que permite um contato mais direcionado ao público alvo da pesquisa além de certa divulgação para grupos profissionais em outras plataformas como o WhatsApp e o Microsoft Teams objetivando alcançar o maior alcance possível.

O desenvolvimento do questionário foi dividido em quatro etapas, a primeira etapa diz respeito ao perfil do participante, a segunda etapa (pergunta Q07) foi definida para

responder a **RQ2** e as etapas 3 (perguntas Q08 a Q26) e 4 (perguntas Q27 a Q40) para apoiar na resposta da **RQ3**:

1. As perguntas Q01 a Q06, vide Tabela 3.3, são relativas a compreensão do perfil do participante.

2. A pergunta Q07 é relativamente extensa e busca identificar a familiaridade dos entrevistados em relação a diferentes ferramentas disponíveis. A Tabela 3.4 apresenta as 30 ferramentas que serão apresentadas aos entrevistados em duas colunas, a primeira indicando as ferramentas encontradas a partir de pesquisas em artigos e literatura branca enquanto a segunda as ferramentas encontradas a partir do mecanismo de busca do Google. As ferramentas foram escolhidas de forma a gerar variedade uma vez que grande parte das ferramentas mais citadas nos estudos encontrados eram exclusivamente para utilização em sistema Android e iOS. A Resposta coletada segue a escala de Likert [58]:

   (a) Utilizo com frequência

   (b) Utilizo ocasionalmente

   (c) Conheço mas não utilizo

   (d) Não conheço

3. As perguntas Q08 a Q40 foram elaboradas e inspiradas a partir de artigos publicados na rede social Linkedin que avaliam diferentes ferramentas de testes expondo seus pontos positivos e negativos [59],[60],[61],[62],[63],[64]. As perguntas Q08 a Q26, conforme apresentado na Tabela 3.5, são relativas aos pontos positivos e vantagens ao se utilizar ferramentas de geração de teste. Os entrevistados foram apresentados aos 19 tópicos e deviam indicar o impacto desses tópicos em sua decisão de adotar ou não uma ferramenta de geração de teste com base neles. A Resposta coletada para essas questões são da escala Likert [58]:

   (a) Essencial: A ferramenta obrigatoriamente precisa possuir tal recurso/funcionalidade para ser vantajoso utilizá-la

   (b) Importante: A presença de determinado recurso/funcionalidade é um fator considerável ao se escolher uma ferramenta

   (c) Útil: O recurso/funcionalidade traz vantagens desejáveis ao se utilizar uma ferramenta

   (d) Irrelevante: O recurso/funcionalidade não altera minha posição diante a decisão de utilizar ou não uma ferramenta

Exemplo: Ao adotar uma nova ferramenta, o entrevistado 1 acha essencial que a mesma possua recursos de gravação e reprodução mas irrelevante se possui uma comunidade ativa ou não. É recomendado que a opção Essencial seja selecionada no máximo 3 vezes. O resultado coletado pelas perguntas será utilizado para responder a **RQ3** com o auxílio do resultado das perguntas Q27 a Q40.

4. As perguntas Q27 a Q40, conforme apresentado na Tabela 3.6, são relativas aos pontos negativos e desafios ao se utilizar ferramentas de geração de teste e possuem funcionamento semelhante as questões relacionadas as vantagens. Neste caso, os entrevistados são apresentados a 14 tópicos e devem indicar o impacto desses desafios em sua escolha de adotar uma ferramenta. A Resposta coletada segue a escala de Likert [58]:

   (a) Inaceitável: A dificuldade impossibilita a utilização de uma ferramenta.

   (b) Desafiador: A dificuldade torna a utilização de uma ferramenta consideravelmente mais difícil/desagradável.

   (c) Superável: A dificuldade complica a utilização de uma ferramenta mas não representa uma barreira relevante.

   (d) Irrelevante: A dificuldade não representa um desafio ao se utilizar uma ferramenta

   Exemplo: Ao adotar uma nova ferramenta, o entrevistado 1 acha um grande desafio lidar com baixo desempenho mas considera que a curva de aprendizado é facilmente superável. É recomendado que a opção Inaceitável seja selecionada no máximo 3 vezes.

ID	Tópico	Explicação
Q08	Versatilidade	Suporte a uma ampla gama de tecnologias, incluindo web, mobile, desktop e aplicativos híbridos [59].
Q09	Teste multiplataforma	Capacidade de realizar testes automatizados em vários sistemas operacionais, como Windows, macOS, iOS e Android [59].
Q10	Reconhecimento de objetos e scripts	Fornecimento de recursos robustos de reconhecimento de objetos facilitando a identificação e interação com vários elementos dentro de um aplicativo [59].

Q11	Gravação e reprodução	Recursos de gravação e reprodução, permitindo que os testadores registrem suas interações com um aplicativo e gerem scripts de teste automaticamente [59].
Q12	Bibliotecas de teste e integrações	Possuir conjunto abrangente de bibliotecas de teste e integrações incorporadas com ferramentas e estruturas de desenvolvimento populares [59].
Q13	Visualização de teste	Fornecimento de logs visuais e resultados de execução de teste, facilitando a análise e depuração de falhas de teste [59].
Q14	Base semelhante a uma estrutura já conhecida	Possuir recursos compartilhados entre ferramentas de forma que facilite os esforços de automação de testes [62].
Q15	Código aberto	Ser gratuito [62].
Q16	Comunidade	Possuir uma comunidade grande e ativa de usuários e desenvolvedores, o que significa que os usuários podem facilmente encontrar ajuda e suporte quando necessário [62].
Q17	API	Permitir fácil integração com outras ferramentas que podem usar a poderosa API para automatizar muitos comandos [60].
Q18	Linguagens de programação	Oferecer suporte a uma ampla variedade de linguagens de programação como Java, Ruby, Python, C#, JavaScript, etc., tornando-o acessível a um grande número de desenvolvedores [62].
Q19	Recompilação do aplicativo	Não requer a recompilação do aplicativo em teste e permite interagir com o aplicativo da mesma forma que um usuário faria. Isso fornece resultados bastante realistas em relação ao feedback do usuário [62].
Q20	Testes baseados em nuvem	Suporte a testes baseados em nuvem [62].
Q21	Recursos	Oferecer uma ampla gama de recursos de teste como teste de gestos, rotações de tela, notificações de dispositivos... [62]

Q22	Rastreabilidade	Oferecer excelente rastreabilidade nas execuções de teste, integrando-se facilmente com outras soluções de gerenciamento de teste [63].
Q23	Utilização e reutilização	Ser fácil configurar um script de teste automatizado mesmo sem um histórico de codificação [63].
Q24	Suporte do fornecedor	Possuir fornecedor que fornece bastante suporte e atualizações frequentes para manter tudo atualizado [63].
Q25	Capacidade de migrar scripts	Facilidade de migrar scripts para outras ferramentas de teste, reduzindo o medo de bloqueio do fornecedor [63].
Q26	Acessibilidade	Oferecer recursos de acessibilidade que tornam mais fácil para os testadores testar certas configurações ARIA e garantir que os aplicativos atendam a uma variedade de deficiências diferentes [63].

Tabela 3.5: Perguntas direcionadas as vantagens ao utilizar ferramentas de testes

ID	Tópico	Explicação
Q27	Custo	Ser uma ferramenta comercial e o custo de licenciamento poder ser uma barreira para os testadores [59].
Q28	Curva de aprendizado	Possuir curva de aprendizado relativamente íngreme de forma que dominar todos os seus recursos possa exigir tempo e esforço significativos [59].
Q29	Sobrecarga de manutenção	Requer manutenção regular para evitar que os testes que dependem de seus recursos sejam interrompidos [59].
Q30	Inconsistência	Possibilidade de não registrar cada ação ou resposta do objeto corretamente, aumentando a possibilidade de falhas de teste quando o aplicativo pode estar funcionando conforme o esperado [62].

Q31	Baixo desempenho	Ser lenta quando se trata de verificação de objetos e possuir velocidades de execução não ideais para a execução regular do pipeline [61].
Q32	Suporte limitado para outros tipos de aplicativos	Ser projetado principalmente para testar aplicativos de um certo tipo ex. móveis e poder não ser tão adequado para testar outros tipos de aplicativos, como aplicativos da Web ou de desktop [62].
Q33	Pobre suporte a emuladores	Não funcionar corretamente com emuladores de dispositivos conectados a uma máquina Windows ou Mac [62].
Q34	Teste de nuvem limitadas	Opções de teste de nuvem limitadas [59].
Q35	Ilegibilidade dos casos de teste	Casos de teste não legíveis para pessoas sem o conhecimento técnico necessário [61].
Q36	Geração de relatórios dependente de terceiros	Não possui recursos de geração de relatórios integrados e requer ferramentas adicionais para gerar relatórios [62].
Q37	Grande pegada	Possuir pegada de VM para seus executores de automação bastante considerável, não sendo ideal para conteinerização generalizada ou uso de nuvem [63].
Q38	Bloqueio do fornecedor	Possuir muitos recursos e bibliotecas integrados que não podem ser migrados para outras ferramentas [64].
Q39	Bloqueio de versão	Possuir licenças e funcionalidades geralmente vinculadas a números de versão, o que faz com que muitas empresas usem versões desatualizadas do aplicativo, o que cria um risco de manutenção [64].
Q40	Longos ciclos de desenvolvimento	Recursos geralmente são entregues apenas após largos intervalos de tempo podendo ficar atrás de novas tecnologias do setor [64].

Tabela 3.6: Perguntas direcionadas aos desafios de utilizar ferramentas de testes

ID	Pergunta	Escala de resposta
Q01	Qual é o seu nome?	Em aberto
Q02	Qual é a sua idade?	Menos de 21 anos; entre 21 a 25 anos; entre 25 e 30 anos; entre 30 e 40 anos; entre 40 e 50 anos; acima de 50 anos
Q03	Qual é o seu nível de escolaridade?	Graduando; graduado; especialização; mestrado e doutorado.
Q04	Há quanto tempo você trabalha com desenvolvimento de software?	Menos de um ano; entre 1 e 3 anos; entre 4 e 10 anos; entre 11 e 20 anos; acima de 20 anos
Q05	Em qual etapa de desenvolvimento de software você trabalha profissionalmente?	Em aberto
Q06	Você trabalha com quais linguagens de programação?	Em aberto

Tabela 3.3: Perguntas direcionadas ao perfil do participante.

ID	Literatura branca	Literatura cinza
Q08	Randoop [65]	Katalon
	Evosuite [66]	Selenium
	MuJava [67]	Appium
	JMeter [68]	TestComplete
	SoapUI [69]	Cypress
	SOAtest [70]	Ranorex Studio
	Monkey [14]	LambdaTest
	DynoDroid [71]	Robot Framework
	AndroidRipper [14]	Cucumber
	SwiftHand [72]	Playwright
	Korat [73]	Puppeteer
	LoadRunner [68]	Postman
	C++test [73]	Eggplant
	JTest [73]	Apache JMeter
	PEX [73]	ACCELQ

Tabela 3.4: Lista de ferramentas apresentadas na QO7.

3.5 Resultados

3.5.1 Análise de dados

Devido ao fato do MLR ser um processo de pesquisa sistemático é necessário que exista um link de rastreabilidade entre os dados extraídos e as fontes primárias para que seja possível reproduzir a pesquisa e chegar aos mesmos resultados ao seguir exatamente cada etapa do processo. Além disso, durante esse processo garantiu-se que foram extraídos tantos

dados qualitativos/quantitativos necessários para abordar suficientemente cada pergunta de pesquisa estabelecida anteriormente, dados esses que serão melhor estudados na fase de síntese. Para uma análise compreensiva dos dados e a garantia de que os pontos descritos foram devidamente respeitados, o processo de análise foi dividida nos seguintes passos que foram adotados, desde a coleta dos dados pelas técnicas de snowbolling, até a elaboração do guia referencial.

1. Criação de uma tabela que identifica quantos documentos foram encontrados em cada etapa de Snowbolling além de quais documentos mencionam cada ferramenta.

2. Criação de tabelas e gráficos que ilustrem as ferramentas de geração de testes mais mencionados na literatura branca e cinza com base nos resultados obtidos.

3. Coleta e análise dos dados da survey identificando padrões ainda que em um nível de análise complexo em que as respostas não possuem escala definida.

4. Revisão e comparação dos resultados da pesquisa com os do survey com o foco de responder cada pergunta de pesquisa específica. As informações serão extraídas e registradas com o máximo de dados quantitativos/qualitativos necessários para atender adequadamente cada pergunta de pesquisa.

5. Criação do guia para auxiliar desenvolvedores de software na seleção, avaliação e utilização de diferentes ferramentas de geração de testes disponíveis na literatura e na industria.

3.6 Síntese do Capítulo

Este capítulo iniciou pela explicação das ferramentas utilizadas e metodologia a ser seguida para a realização do trabalho além dos conceitos necessários para o entendimento da revisão multivocal de literatura. Em seguida, apresentou cada etapa da metodologia detalhadamente e por fim, apresentou como os dados foram analisados, desde a coleta até a elaboração do guia referencial.

Capítulo 4

Resultados

Neste capítulo serão apresentados o desenvolvimento da pesquisa com suas respectivas discussões, de modo que: a Seção 4.1 indica os resultados obtidos na literatura branca e cinza; a Seção 4.2 elucida uma análise dos resultados, isto é, uma comparação dos resultados obtidos no questionário com os da Seção 4.1; e a Seção 4.3 apresenta a proposição do guia referencial prático, para para auxiliar desenvolvedores de software na seleção, avaliação e utilização de diferentes ferramentas de geração de testes.

4.1 Ferramentas de geração de testes utilizadas na literatura (RQ.1)

4.1.1 Ferramentas encontradas na literatura branca

A Tabela 4.1 apresenta os estudos encontrados durante os processos de Snowballing em suas três iterações diferentes. A quantidade de iterações foi definida durante a pesquisa uma vez que percebeu-se a enorme repetição de estudos já adicionados ou excluídos e que o conteúdo encontrado se distanciava cada vez mais da temática desejada uma vez que se especializava em questões de privacidade e detecção de malwares em aplicativos móveis não mencionando ferramentas de testes.

Estudo	1ª Iteração		2ª Iteração		3ª Iteração	
	Frente	Trás	Frente	Trás	Frente	Trás
Droidbot: A lightweight test input generator for android	21	11	59	48	-	72

Tabela 4.1: Resultados do processo Snowballing para trás

Os aplicativos Android são programas orientados a eventos, portanto, suas entradas são normalmente na forma de eventos de UI (cliques, deslizar e entradas de texto) ou eventos do sistema (recebimento de chamadas e mensagens de texto) [74]. A interface gráfica do usuário (GUI) é o tipo mais importante de UI para a maioria dos aplicativos móveis, onde os aplicativos apresentam conteúdo e ferramentas acionáveis na tela [75]. A Tabela 4.2 indica algumas das ferramentas mais encontradas nos estudos resultantes do processo Snowballing.

Ferramenta	Tipo	Estratégia	Artigos
Monkey	Caixa-preta	Aleatória	[76] [77] [71] [72] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [75] [44] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119] [120] [121] [122] [123] [124] [125] [126] [127] [128] [129] [130] [131] [132] [133] [134] [135] [136] [137] [138] [139] [140] [141] [142] [143] [144] [145] [146] [147] [148] [149] [150] [151] [152] [153] [154] [155] [156]
Dynodroid	Caixa-preta	Aleatória	[76] [71] [72] [76] [82] [84] [86] [88] [89] [90] [91] [93] [96] [99] [75] [44] [101] [103] [104] [105] [107] [108] [109] [157] [114] [115] [120] [124] [131] [133] [134] [135] [137] [138] [139] [142] [144] [145] [146] [147] [148] [149] [150] [152] [153] [154] [155]
SwiftHand	Caixa-preta	Modelo	[76] [72] [86] [90] [91] [99] [44] [104] [108] [157] [112] [114] [137] [158] [144] [146] [148] [150] [152] [153] [155]
Puma	Caixa-preta	Modelo	[76] [78] [86] [90] [99] [75] [103] [105] [108] [109] [112] [114] [131] [132] [134] [146] [152] [153] [154] [155]
Evodroid	Caixa-cinza	Sistemática	[76] [72] [84] [86] [90] [91] [99] [101] [108] [157] [112] [114] [123] [124] [127] [134] [137] [144] [145] [146] [147] [148] [150] [155]
AndroidRipper	Caixa-preta	Modelo	[77] [80] [82] [85] [88] [90] [94] [97] [99] [102] [108] [109] [114] [134] [159] [142]

MobiGUITAR	Caixa-preta	Modelo	[76] [82] [86] [89] [92] [108] [157] [112] [114] [131] [142] [148]
Randoop	Caixa-cinza	Aleatória	[160] [161] [162] [163] [164] [73] [165]
Evosuite	Caixa-preta	Sistemática	[160] [161] [162] [166] [128] [163] [148] [164] [155] [165]
A3E	Caixa-cinza	Modelo	[90] [91] [93] [97] [99] [44] [103] [109] [157] [112] [114] [115] [120] [131] [135] [138] [159] [142] [146] [147] [150] [152] [153] [155]
Orbit	Caixa-cinza	Modelo	[90] [93] [99] [103] [108] [114] [131] [146] [147] [150]
JMeter	-	Sistemática	[167] [157] [168]
Robotium	Caixa-preta	Sistemática	[89] [169] [170] [97] [102] [114] [138] [159] [149] [152] [153]
PEX	Caixa-cinza	Sistemática	[171] [162] [172] [173] [163] [73]
Humanoid	Caixa-preta	Aprendizagem profunda	[98] [75] [101] [103] [105] [109] [110] [122] [125] [129] [132] [133] [134] [135] [143] [147] [152] [153] [174]
Ape	Caixa-preta	Modelo	[98] [99] [110] [115] [116] [117] [118] [120] [121] [122] [123] [124] [125] [128] [129] [131] [133] [137] [158] [138] [140] [174]
TimeMachine	Caixa-preta	Estado	[98] [99] [115] [122] [124] [125] [129] [130] [152] [153] [174]
Sapienz	Caixa-cinza	Sistemática	[75] [44] [101] [103] [104] [105] [109] [157] [110] [111] [112] [114] [115] [120] [121] [122] [123] [124] [125] [127] [130] [131] [134] [135] [137] [138] [139] [143] [144] [145] [146] [147] [148] [149] [150] [152] [153] [154] [155]
Stoat	Caixa-preta	Modelo	[75] [44] [71] [103] [104] [108] [109] [157] [110] [112] [115] [116] [117] [118] [120] [122] [123] [124] [125] [127] [130] [131] [135] [138] [139] [159] [142] [143] [144] [146] [147] [148] [149] [150] [152] [153] [154] [155]
C++Test	Caixa-branca	Sistemática	[163] [145] [73]
JTest	Caixa-cinza	Sistemática	[164] [73]
MuJava	Caixa-branca	Sistemática	[175] [176] [177]
Soap UI	Caixa-preta	Sistemática	[178]

Korat	Caixa-cinza	Sistemática	[73] [176]

Tabela 4.2: Ferramentas encontradas

Monkey: Faz parte do kit de ferramentas para desenvolvedores Android e, portanto, não requer nenhum esforço adicional de instalação. Implementa a estratégia aleatória mais básica, pois considera o aplicativo em teste uma caixa preta e só pode gerar eventos de UI [76]. Envia tipos aleatórios de eventos de entrada para locais aleatórios na tela sem considerar sua estrutura GUI [75]. Monkey é amplamente utilizado na indústria para testes de estresse porque é fácil de usar e compatível com qualquer versão do Android. É uma base popular para avaliar novas técnicas de teste [98]

Dynodroid: Pode gerar entradas de UI e de sistema e permite combinar entradas de humanos e máquinas [71]. Pode gerar eventos do sistema e verificar quais são relevantes para o aplicativo. É capaz de selecionar eventos que foram selecionados com menos frequência (estratégia de frequência) e levar em consideração o contexto (BiasedRandom estratégia), ou seja, eventos que são relevantes em mais contextos serão selecionados com mais frequência [75].

SwiftHand: Gera apenas eventos de UI de toque e rolagem e não pode gerar eventos do sistema [75]. Utiliza aprendizado de máquina para aprender um modelo do aplicativo durante o teste, usa o modelo aprendido para gerar entradas do usuário que visitam estados inexplorados do aplicativo e usa a execução do aplicativo nas entradas geradas para refinar o modelo [72]. SwiftHand cria um modelo dinâmico de máquina de estado finito do aplicativo e o refina para minimizar as reinicializações do aplicativo enquanto o explora [146].

Puma: Framework que pode ser facilmente estendido para implementar qualquer análise dinâmica em aplicativos Android usando a estratégia básica de exploração aleatória também implementada pelo Monkey [76]. Análises escritas no PUMA podem personalizar a exploração do aplicativo escrevendo manipuladores de eventos compactos que separam a lógica de análise da lógica de exploração [78].

Evodroid: Utiliza algoritmos evolutivos para gerar informações relevantes. Na estrutura de algoritmos evolutivos, o EvoDroid representa indivíduos como sequências de entradas de teste e implementa a função de aptidão para maximizar a cobertura [76]. Ele usa o conhecimento específico da plataforma Android para analisar estaticamente o aplicativo

e inferir um modelo de seu comportamento. O modelo inferido permite a busca evolutiva para determinar como os indivíduos devem ser cruzados para transmitir sua composição genética às gerações futuras [84].

AndroidRipper: Utiliza um ripador acionado pela interface do usuário para percorrer sistematicamente a interface do usuário do aplicativo [108]. O AndroidRipper analisa dinamicamente a GUI da aplicação com o objetivo de obter sequências de eventos acionáveis através dos widgets da GUI. Cada sequência fornece um caso de teste executável. Durante sua operação, o AndroidRipper mantém um modelo de máquina de estado da GUI (Árvore GUI). O modelo Árvore GUI contém o conjunto de estados da GUI e transições de estado encontrados durante o processo de extração [77].

MobiGUITAR: Ferramenta que gera dinamicamente um modelo de aplicativo durante a exploração, com base no estado de tempo de execução dos widgets GUI [108]. É baseado em três etapas principais: (1) Ripping que atravessa dinamicamente a GUI de um aplicativo e cria seu modelo de máquina de estado, (2) Geração que usa o modelo e critérios de adequação de teste para obter testes que são sequências de eventos, e (3) Execução que reproduz os testes [82].

Randoop: Gerador de casos de teste aleatórios, ferramenta que combina chamadas de método aleatoriamente para cobrir grandes porções de código [160]. Randoop depende de uma estratégia de testes aleatórios baseada em feedback, coletando informações da execução dos testes à medida que são gerados para evitar testes redundantes e ilegais, para gerar testes de regressão que capturam como o sistema se comporta como está [165]. RANDOOP é uma ferramenta de geração de testes para programas orientados a objetos. Portanto, incorpora apenas técnicas fracas de geração de dados para tipos primitivos [73].

Evosuite: Utiliza um algoritmo evolutivo para desenvolver um conjunto de testes unitários que satisfaçam um determinado conjunto de objetivos de teste [165].

APE: ferramenta de teste de GUI baseada em modelo, utiliza informações de tempo de execução para evoluir dinamicamente seu critério de abstração por meio de uma árvore de decisão, que pode equilibrar efetivamente o tamanho e a precisão do modelo [98]. Utiliza internamente o Monkey para ocasionalmente emitir eventos aleatórios da interface do usuário e eventos do sistema para evitar travar nos estados locais [98].

Robotium: Framework de automação de testes que permite aos desenvolvedores es-

crever testes de UI de caixa preta para aplicativos Android. Permite que desenvolvedores escrevam testes de aceitação de função, sistema e usuário abrangendo diversas atividades do Android [89].

Pex: Ferramenta de execução simbólica para linguagens .NET que pode ser usado para gerar casos de teste para um serviço web [172]. Pex é um gerador de entrada de teste, mas persiste os dados gerados como código executável na forma de testes de unidade tradicionais (sem parâmetros) [179]. A ferramenta é capaz de explorar todos os caminhos viáveis do método em teste: PEX começa com uma entrada aleatória simples para um determinado método em teste. Ao executar o método, coleta informações de tempo de execução, por exemplo, valores simbólicos para todas as variáveis e restrições de caminho. A cada declaração de condição, coleta informações sobre os critérios de ramificação. PEX reexecuta o método com valores de entrada que satisfazem todas as condições do caminho em um processo conhecido como execução simbólica concreta (concólica) [73]. Pex só pode explorar implementações determinísticas de serviços de thread único, cujo código compilado pode ser instrumentado [179]

Humanoid: É a primeira ferramenta de teste baseada em aprendizagem profunda. O núcleo é um modelo de rede neural profunda que prevê quais elementos da interface do usuário na página da GUI atual têm maior probabilidade de interação dos usuários e como interagir com eles [98].

TimeMachine: Desenvolve uma população de estados que podem ser capturados na descoberta e retomados quando necessário para encontrar erros profundos [98]. Sua singularidade é a capacidade de capturar instantâneos e retomar o estado específico do aplicativo para testes adicionais por meio da máquina virtual subjacente baseada em Android [98]. No início de uma sessão de teste, o TimeMachine tira um instantâneo do estado inicial. Durante a execução do teste, o TimeMachine tira um instantâneo de cada estado interessante e o adiciona ao corpus de estado, retorna ao estado interessante e executa o próximo teste [99]. Para cada transição de um estado para outro, o TimeMachine também registra a sequência de eventos mais curta [99].

Orbit: Analisa estaticamente o aplicativo para extrair eventos de UI relevantes, depois usa essas informações para construir um modelo do aplicativo e, em seguida, usa o modelo para testar o aplicativo [146]. No entanto, ele usa uma exploração simples e profunda que reinicia o aplicativo de seu estado inicial para voltar aos estados anteriores [103].

A3E: Explora aplicativos com duas estratégias: Exploração em profundidade (Depth First), que analisa sistematicamente os aplicativos enquanto são executados nos dispositivos reais e sem acesso ao código-fonte, e exploração direcionada (Targeted Exploration), que prioriza a exploração de atividades que começam na atividade inicial em uma transição de atividade estática gráfico [103]. A3E representa cada atividade como um estado individual sem considerar que a atividade pode existir em diferentes estados o que leva a à falta de alguns comportamentos da aplicação, uma vez que nem todos os estados das atividades estão sendo explorados [103].

Stoat: Combina análise estática e dinâmica para construir um modelo do aplicativo e então usa o modelo gerado para orientar a exploração do aplicativo durante o teste. Stoat injeta, além de eventos de UI, eventos de sistema, de modo a melhorar a eficácia da geração de entradas de teste [146].

Sapienz: Utiliza abordagem baseada em pesquisa multiobjetivo para explorar e otimizar automaticamente sequências de teste, minimizando a duração e, ao mesmo tempo, maximizando a cobertura e a detecção de falhas [146]. Sapienz explora os componentes do aplicativo usando GUIs específicas e sequências complexas de eventos de entrada com um padrão predefinido [103]. Este padrão pré-definido é denominado gene motivo que captura a experiência dos testadores. Assim, produz uma maior cobertura de código ao concatenar os eventos atômicos [103].

C++Test: Ferramenta comercial da Parasoft de melhoria de qualidade de software para C/C++ que vem como um IDE independente ou um plugin Eclipse [73]. É melhor usado para gerar testes de regressão, que detectam mudanças no comportamento dos sistemas em teste ao longo do tempo [73]. Possui lista de recursos semelhantes ao JTest, porém, com implementação de técnicas de geração de dados menos sofisticados [73].

JTest: Produto de teste abrangente da Parasoft para Java, oferece suporte às equipes de desenvolvimento na construção de novos ou na melhoria da qualidade de aplicativos Java legados. Jtest facilita análise estática, análise de tempo de execução, automação de processos de revisão de código e testes unitários [73]. Jtest suporta geração automática de testes JUnit e geração automática de testes de regressão. Para testes de regressão, a execução inicial do teste determina os valores de retorno esperados, que são registrados e usados em execuções posteriores [73].

Mujava: Sistema de mutação para classes Java que gera mutantes automaticamente

para testes de mutação tradicionais e testes de mutação em nível de classe [67]. Depois de criar mutantes, o muJava permite que o testador entre e execute testes e avalie a cobertura de mutação dos testes. No muJava, os testes para a classe em teste são codificados em classes separadas que fazem chamadas para métodos na classe em teste [67].

SoapUI: Ferramenta de teste de API gratuita, de código aberto e multiplataforma que permite criar e executar testes automatizados funcionais, de regressão, de conformidade e de carga de maneira fácil e rápida [69]. SoapUI também permite realizar testes não funcionais, como testes de desempenho e segurança [69].

Korat: Ferramenta de geração de casos de teste baseada na especificação Design by ContractTM. Ele usa métodos pós-condição como oráculo e usa a pré-condição para gerar dados de entrada de teste complexos [73]. Korat usa um método repOK() e um método finatization() para construir todos os dados de entrada de teste não isomórficos até um determinado limite [73].

LoadRunner: Ferramenta de teste automatizada cuja principal função é identificar gargalos que ajudam a detectar e prevenir problemas de desempenho de software [180]. LoadRunner é o melhor para verificar o desempenho e também verificar a rede [180].

4.1.2 Outras ferramentas encontradas na literatura branca

Ferramentas encontradas durante o processo snowballing que não são muito citadas ou desempenham papel central em determinado estudo, porém, seu mencionamento agrega certo valor ao trabalho.

MonkeyRunner: API fornecida pelo Android SDK que permite ao programador escrever scripts de teste em Python para exercitar aplicativos Android [79].

Q-testing: Ferramenta de teste baseada em aprendizagem por reforço que utiliza uma rede neural treinada para comparar páginas GUI [98]. Utiliza-se recompensas que são usadas e atualizadas iterativamente para orientar os testes e cobrir mais funcionalidades do aplicativo em questão: Se uma página for semelhante a qualquer uma das páginas GUI exploradas anteriormente, o comparador dará uma pequena recompensa. Caso contrário, o comparador dará uma grande recompensa [98].

4.1.3 Ferramentas encontradas na literatura cinza

Selenium: Framework com muitas ferramentas e plug-ins para testes de aplicativos da Web [181]. É uma escolha popular no espaço de automação de testes de código aberto, em parte devido ao seu grande e ativo desenvolvimento e comunidade de usuários [181]. Fornece suporte para vários sistemas operacionais e diferentes linguagens de programação como: Java, Python, C# e JavaScript [182]. Pode ser integrado a vários frameworks para fornecer um framework híbrido, o que torna o teste mais simples [183]. Algumas vantagens do Selenium incluem sua facilidade de uso e flexibilidade permitindo aos usuários depurar e definir pontos de interrupção em casos de teste [35]. Alguns desafios incluem falta de suporte a instruções condicionais e de iteração, testes de banco de dados e nenhuma capacidade de tratamento de erros [35].

Appium: Ferramenta de teste mobile multiplataforma de código aberto que permite aos desenvolvedores escrever testes em várias plataformas, como iOS e Android [35]. Fornece testes de API REST [182]. Appium centra-se no teste da interação do usuário com o conteúdo de aplicativos da web para dispositivos móveis. Os resultados dos testes são utilizados para avaliar a precisão e o envolvimento do usuário com o aplicativo móvel, bem como a facilidade de uso ou disponibilidade dos recursos [35]. Existem bibliotecas clientes em Java, Ruby, Python, PHP, JavaScript e C#, que suportam extensões do Appium para o protocolo WebDriver [182].

Test Complete: Plataforma comercial integrada para testes de aplicativos desktop, móveis e web [181]. Oferece alguns recursos importantes de automação de testes, como testes orientados por palavras-chave e dados, testes entre navegadores, testes de API e integrações de CI [181]. Possui recursos que garantem que os testes sejam repetidos uma vez após a gravação [182]. A criação de frases de efeito utilizadas para testes é visual, simples e não requer nenhuma habilidade de programação [35]. O script requer compreensão das instruções do script, mas permite testes mais dominantes e adaptáveis [35]. Fácil configuração e uso porém precisa de treinamento para usar a ferramenta corretamente [184].

Cucumber: Suporte a diferentes linguagens, como Java.net e Ruby. Os cenários de teste podem ser inseridos em texto simples em inglês. Portanto, não requer informações de código e oferece facilidade de uso [182].

Ranorex Studio: Ferramenta e Framework de teste [185]. É uma das ferramentas comerciais mais populares para construir e executar testes automatizados de web e GUI

[186]. Permite testes ponta a ponta em diferentes dispositivos como desktop, web e dispositivos móveis. Os testes são automatizados no desktop, ele pode então ser executado em dispositivos móveis iOS ou Android nativos ou virtuais ou em simuladores e emuladores [182]. Ranorex é composto, mas não se limita aos seguintes recursos: códigos de teste reutilizáveis, integração com diversas ferramentas, reconhecimento de GUI, gravação e reprodução, detecção de bugs e etc [35]. Ranorex é fácil de usar, barato e pode ser usado por qualquer equipe de testes [35].

Katalon: Framework de testes automatizados que oferece um conjunto abrangente de recursos para implementar soluções completas de testes automatizados [181]. Construído com base nas estruturas de código aberto Selenium e Appium [181]. Não oferece suporte a testes distribuídos suportando apenas testes de automação Web, Mobile e API. fornece exportações para C#, Java, Ruby, Python, Groovy ou Robot Framework [182].

Cypress: Ferramenta de teste ponta a ponta para automação moderna de testes da web baseada em JavaScript. Opera diretamente no navegador utilizando uma abordagem de manipulação de DOM interagindo diretamente com a aplicação que está sendo testada [187]. Possuiu suporte para controle de tráfego de rede que permite testes de cenários extremos sem envolvimento do servidor [188]. Suporta apenas scripts de casos de teste em JavaScript e não fornece suporte para múltiplas guias [187].

Postman: Plataforma que suporta todos os estágios do ciclo de vida de uma API: desenvolvimento, teste, publicação e documentação [189]. Oferece testes automatizados e integração no pipeline de CI/CD (integração contínua/entrega contínua) em os usuários podem criar conjuntos de testes personalizados em JavaScript, parametrizar solicitações, executar os testes e depurar, bem como testes exploratórios onde os usuários podem usar scripts para enviar solicitações assíncronas, encadear solicitações para criar cenários de teste e documentar as descobertas [189].

Apache Jmeter: Software de código aberto que permite testes de carga, testes funcionais e medição de desempenho [189]. Para testes de carga, existe um modo CLI que permite carregar testes de qualquer sistema operacional compatível com Java, também oferece suporte a multithreading e suporte a CI [189].

Robot Framework: Framework de automação de testes baseada em palavras-chave [190]. Os casos de teste são armazenados em arquivos HTML e fazem uso de palavras-chave implementadas em bibliotecas de teste para conduzir o software em teste, enquanto

os conjuntos de testes são criados a partir de arquivos e diretórios [190].

4.2 Resultados do Survey

Com o objetivo de responder a RQ.2 e RQ.3, conduziu-se um survey. O survey ficou disponível online por 4 semanas. No total, obtivemos 40 respostas e o tempo médio de resposta foi de aproximadamente 10 minutos. As respostas obtidas foram esquematizadas em uma planilha virtual para a análise das informações coletadas.

A Tabela 4.3 apresenta os resultados das questões Q02 e Q03 revelando o perfil demográfico dos participantes. A tabela indica que a maior parte dos participantes (62.5%) possui idade entre 21 e 25 anos e nenhum com idade superior a 50. Da mesma forma, a tabela indica o nível de escolaridade dos mesmos em que a maioria (92%) não possui pós-graduação.

Idade (Anos)		Escolaridade	
Intervalo	(%)	Nível	(%)
Menos de 21	10	Graduando	54
Entre 21 e 25	62.5	Graduado	38
Entre 25 e 30	20	Especialização	8
Entre 30 e 40	5	Mestrado	0
Entre 40 e 50	2.5	Doutorado	0
Acima de 50	0		

Tabela 4.3: Perfil dos participantes

Em relação a atuação profissional dos participantes, a Tabela 4.4 apresenta os resultados da questões Q04 referente ao tempo de experiência dos participantes, ou seja, o tempo em que ocuparam cargos relacionados a desenvolvimento de software. É possível observar que a vasta maioria dos participantes, aproximadamente 77% trabalha nessa área a menos de 3 anos.

Experiência (Anos)	(%)
Menos de 1	10
Entre 1 e 3	68
Entre 4 e 10	23
Entre 11 e 20	0
Acima de 20	0

Tabela 4.4: Experiência dos participantes

Ao serem questionados sobre a atuação profissional, quase a totalidade pertence à etapas de desenvolvimento e programação de software sendo que cerca de um pouco mais

de um quarto dos participantes se descreve como desenvolvedor fullstack presente em diversas etapas como análise de requisitos, planejamento, codificação e testes. Ainda no escopo profissional, a questão Q05 busca identificar quais linguagens de programação os participantes utilizam em suas áreas, a linguagem mais mencionada foi Javascript sendo citada por 67% dos participantes, em segundo lugar Python citada por 46% e por fim Typescript por 23% e Php com 15%. A Tabela 4.5 apresenta a quantidade de vezes que determinada linguagem foi mencionada de acordo com a faixa-etária dos participantes.

Linguagem	**Menos de 21 anos**	**Entre 21 e 25 anos**	**Entre 25 e 30 anos**	**Entre 30 e 40 anos**	**Entre 30 e 40 anos**
Python	3	10	5		
Java	1	10		2	1
Javascript	3	19	5	2	
Typescript		8	3		
PHP		4	1	1	
Ruby		2			
C#		1	1		
C	1	2			
C++	1	1			

Tabela 4.5: Linguagens mencionadas pelos participantes

4.2.1 Ferramentas de geração de testes usadas na indústria (RQ.2)

Com o intuito de responder a RQ.2, foi adicionado no survey uma pergunta, Q07, questionando aos participantes sobre a sua familiaridade com determinadas ferramentas de testes. Ferramentas estas identificadas a partir dos estudos selecionados durante os processos de snowballing e na literatura "branca". O gráfico 4.1 ilustra os resultados obtidos comparando as ferramentas usadas e as que mesmo não sendo utilizadas são ao menos conhecidas. A Tabela 4.6 indica a quantidade de participantes que conhecem a ferramenta, os mesmos são separados de acordo com sua faixa-etária.

A ferramenta mais mencionada nos estudos obtidos: Monkey e as demais mais mencionadas em sequência: AndroidRipper, Swifthand e Dynodroid não são utilizadas pelos participantes mas cerca de 17,5%, 10%, 10% e 5% responderam que conhecem as respectivas ferramentas mencionadas. As quatro ferramentas mencionadas são utilizadas exclusivamente para desenvolvimento mobile e a ferramenta Monkey se demonstra como a mais conhecida por uma diferença de menos de 10%.

Ao considerar as demais ferramentas, o JTest é consideravelmente mais utilizada que as demais sendo que dos 30% dos participantes que responderam que utilizam a ferramenta, um terço a utiliza com frequência. A ferramenta JMeter é a segunda mais utilizada com

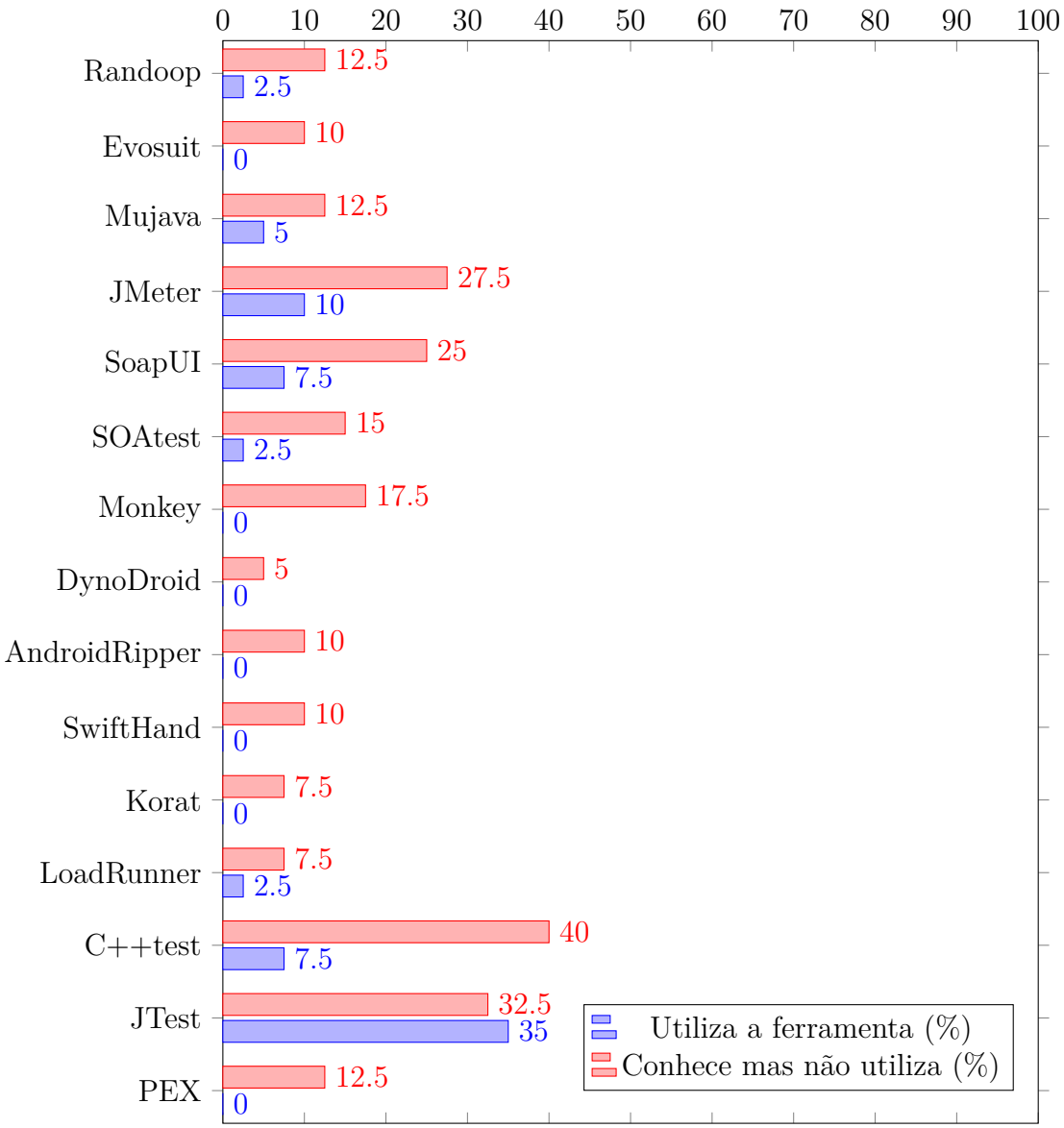

Figura 4.1: Familiaridade com as ferramentas encontradas em literatura branca

um porcentual de cerca de 10% em que metade a usa com frequência e em seguida C++ test com taxa de utilização de 7,7%. Observa-se que as três ferramentas também são bem conhecidas mesmo não sendo utilizada por todos os participantes, no total, cerca de 66,7% conhecem o JTest, 48,7% o c++ teste e 38.5% o JMeter. As demais ferramentas são relativamente desconhecidas com Dynodroid e Korat não sendo familiares para mais dos 90% participantes e Evosuit, AndroidReaper, Swifthand, LoadRunner e PEX por mais de 85%.

A questão Q08 é semelhante a questão Q07 porém com o foco do estudo em ferramentas citadas em fontes não acadêmicas como na literatura cinza ou em pesquisas na internet. O gráfico 4.2 ilustra os resultados obtidos comparando as ferramentas usadas e as que

Linguagem	**Menos de 21 anos**	**Entre 21 e 25 anos**	**Entre 25 e 30 anos**	**Entre 30 e 40 anos**	**Entre 30 e 40 anos**
Randoop	1	2	2		1
Evosuite	1	2			1
Mujava	1	4		1	1
JMeter	1	9	2	2	1
Soap UI	1	8	2	1	1
SOAtest	1	3	1	1	1
Monkey	1	3	1	1	1
Dynodroid		1			1
AndroidRipper		3			1
SwiftHand		2	1		1
Korat		2			1
LoadRunner	1	2			1
C++Test	2	11	4	1	1
JTest	3	18	4	1	1
PEX	1	2		1	1

Tabela 4.6: Ferramentas conhecidas encontradas em literatura branca

mesmo não sendo utilizadas são ao menos conhecidas. A Tabela 4.7 indica a quantidade de participantes que conhecem a ferramenta, os mesmos são separados de acordo com sua faixa-etária.

Dentre as ferramentas citadas apenas duas não são utilizadas por nenhum participante sendo elas Ranorex studio e AccelQ, além destas, apenas três são desconhecidas para mais de 80% dos questionados sendo elas Katalon, LambdaTeste e Eggplant.

As ferramentas mais conhecidas são Postman, Selenium, Cypress e Cucumber sendo familiares para respectivamente 87,5%, 85%, 72,5% e 72,5% dos participantes. Considerando a utilização e a sua frequência, a ferramenta Postman é a mais utilizada sendo que dos 72,5% dos participantes que a utilizam, 62% a utiliza com frequência. Selenium aparece em segunda colocação em que dos 42,5% dos participantes que a utilizam, 29,4% a utiliza com frequência e por fim Cypress em que dos 40% que a utilizam, 50% a utiliza com frequência.

4.2.2 Vantagens e desafios do uso das ferramentas de geração de teste (RQ.3)

A terceira e quarta etapa do survey foram elaboradas para responder a RQ.3. As questões Q09 e Q10 do survey são respectivas a funcionalidades, características e capacidades que podem ser observadas em diferentes ferramentas de testes. Os participantes foram solicitados a avaliar uma lista dessas opções de forma a tentar quantificar o impacto que

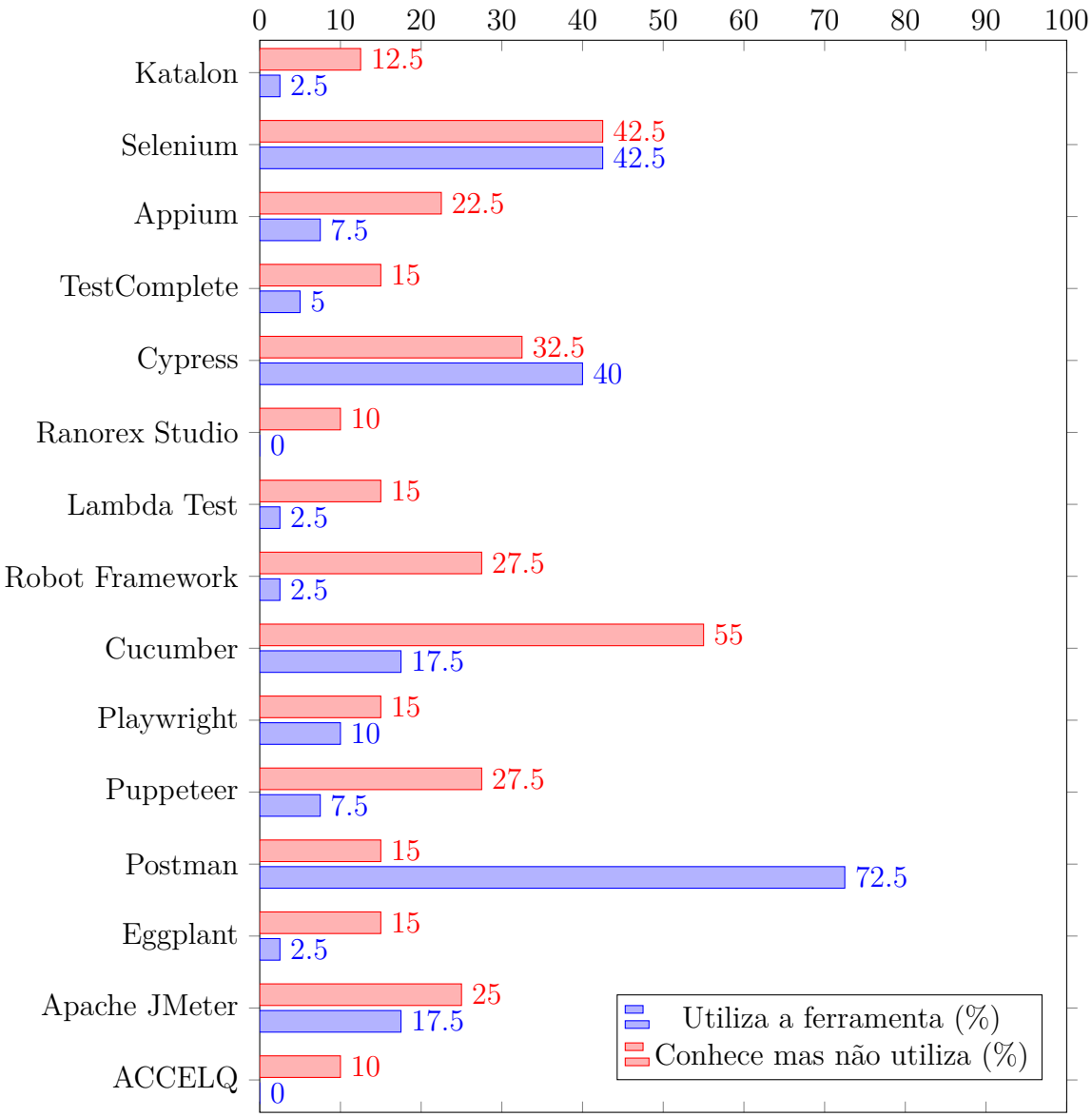

Figura 4.2: Familiaridade com as ferramentas encontradas em literatura cinza

essas características exercem na vida profissional de um desenvolvedor. As funcionalidades listadas não são diretamente relacionadas a nenhuma ferramenta específica de forma que os profissionais pensem nas opções de forma geral.

A questão Q09 lista características que podem influenciar um profissional da área a escolher uma ferramenta em detrimento de outra levando em conta apenas o fato de uma possuir tal funcionalidade e a outra não. Das opções listadas apenas cinco foram consideradas irrelevantes por mais de 10% dos participantes sendo elas: Recursos de gravação e reprodução; Suporte a uma ampla variedade de linguagens de programação; Possuir base semelhante a alguma estrutura reconhecida; Suporte a testes baseados em nuvem e Oferecer recursos de acessibilidade com as duas primeiras opções sendo as mais

Linguagem	**Menos de 21 anos**	**Entre 21 e 25 anos**	**Entre 25 e 30 anos**	**Entre 30 e 40 anos**	**Entre 30 e 40 anos**
Katalon		4		1	1
Selenium	3	22	6	2	1
Appium	1	8	1	1	1
TestComplete	1	4	1	1	1
Cypress	3	18	5	2	1
Ranorex Studio		2		1	1
Lambda Test		3	3		1
Robot Framework	1	6	3	1	1
Cucumber	2	19	5	2	1
Playwright		7	2		1
Puppeteer	1	9	2	1	1
Postman	3	23	7	1	1
Eggplant		5	1		1
Apache Jmeter	1	10	3	2	1
ACCELQ		2		1	1

Tabela 4.7: Ferramentas conhecidas encontradas em literatura cinza

votadas com respectivamente 17.9% e 15,4%.

Esperava-se que a opção da ferramenta ser gratuita fosse ser marcada como essencial por uma quantidade maior de participantes, a característica foi considerada como importante por 47,5% porém essencial por apenas 25% dos participantes. Observou-se também que a maioria dos participantes não se incomoda pela necessidade de recompilar um programa para realizar um teste uma vez que a característica foi marcada como a menos essencial com apenas 5%.

Os participantes demonstraram claramente que valorizam não apenas as funcionalidades da ferramenta em si mas também o tratamento que a mesma recebe pelos seus respectivos fornecedores e demais desenvolvedores que trabalham com as mesmas. Cerca de 82,1% considera uma comunidade grande e ativa de usuários e desenvolvedores importante ou essencial para uma ferramenta e 66,6% demonstram o mesmo em relação a ferramenta possuir suporte e atualizações frequentes.

Em relação a scripts de testes, 22.5% considera essencial e 55% importante ser fácil configurar um script de teste automatizado e 50% essencial ou importante possuir tal facilidade ao migrar scripts para outras ferramentas de teste. A automação de testes desempenha um papel fundamental na redução do alto custo dos testes de GAPs (teste manual de caixa preta de aplicativos baseados em interface gráfica do usuário) [191]. Para

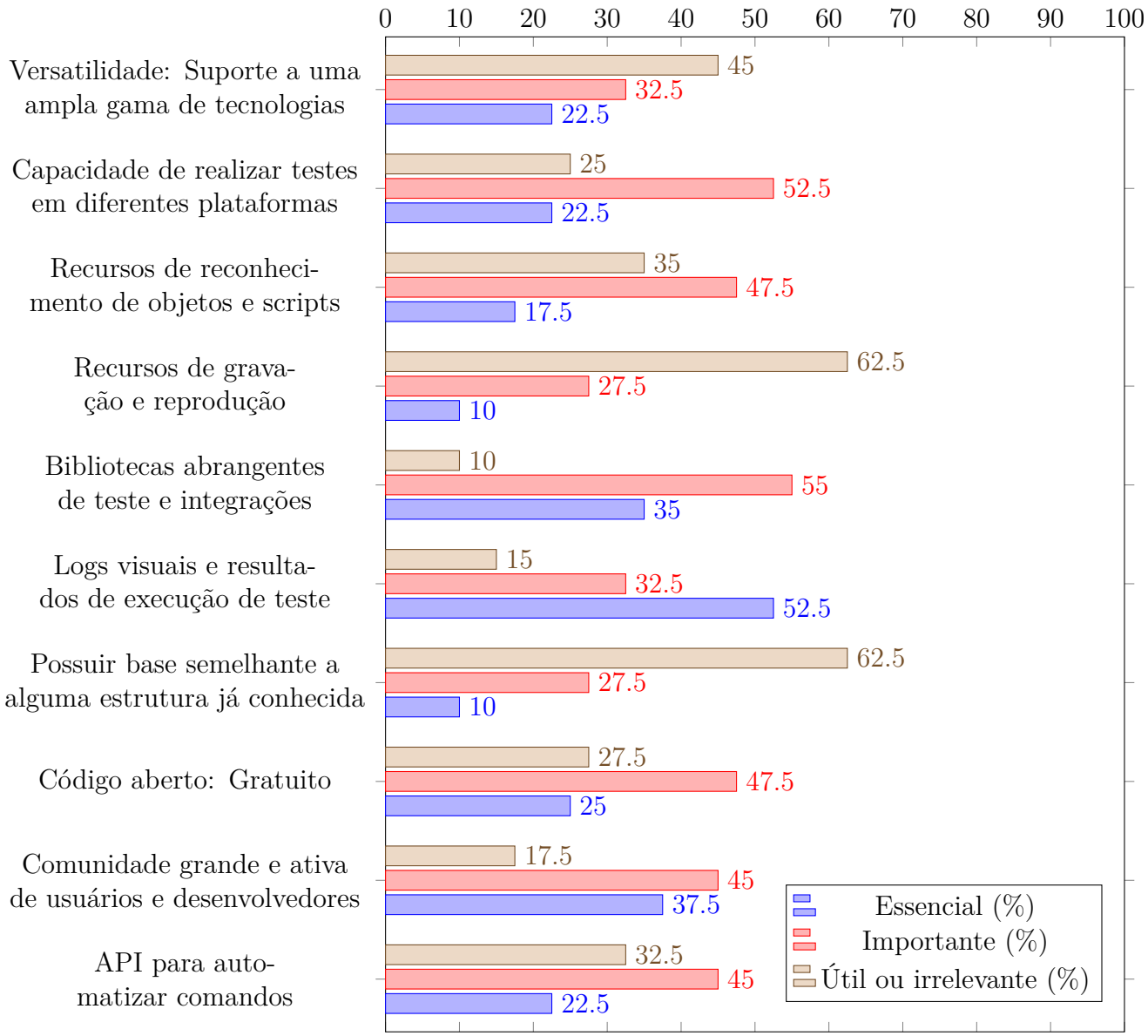

Figura 4.3: Vantagens desejáveis ao selecionar ferramentas Parte 1

automatizar esse processo, engenheiros de teste escrevem programas usando linguagens de script (por exemplo, JavaScript e VBScript), e esses programas (scripts de teste) imitam os usuários executando ações em objetos GUI desses GAPs usando algumas estruturas de teste subjacentes. O esforço extra colocado na escrita de scripts de teste é recompensado quando esses scripts são executados repetidamente para determinar se os GAPs se comportam conforme desejado [191].

A questão Q10 lista características que podem influenciar um profissional da área a escolher uma ferramenta em detrimento de outra, dessa vez levando em conta a presença de certo obstáculo ou desvantagem que pode tornar a ferramenta não desejável. Dentre os resultados obtidos a opção inconsistente de destaca como a mais indesejável de forma que

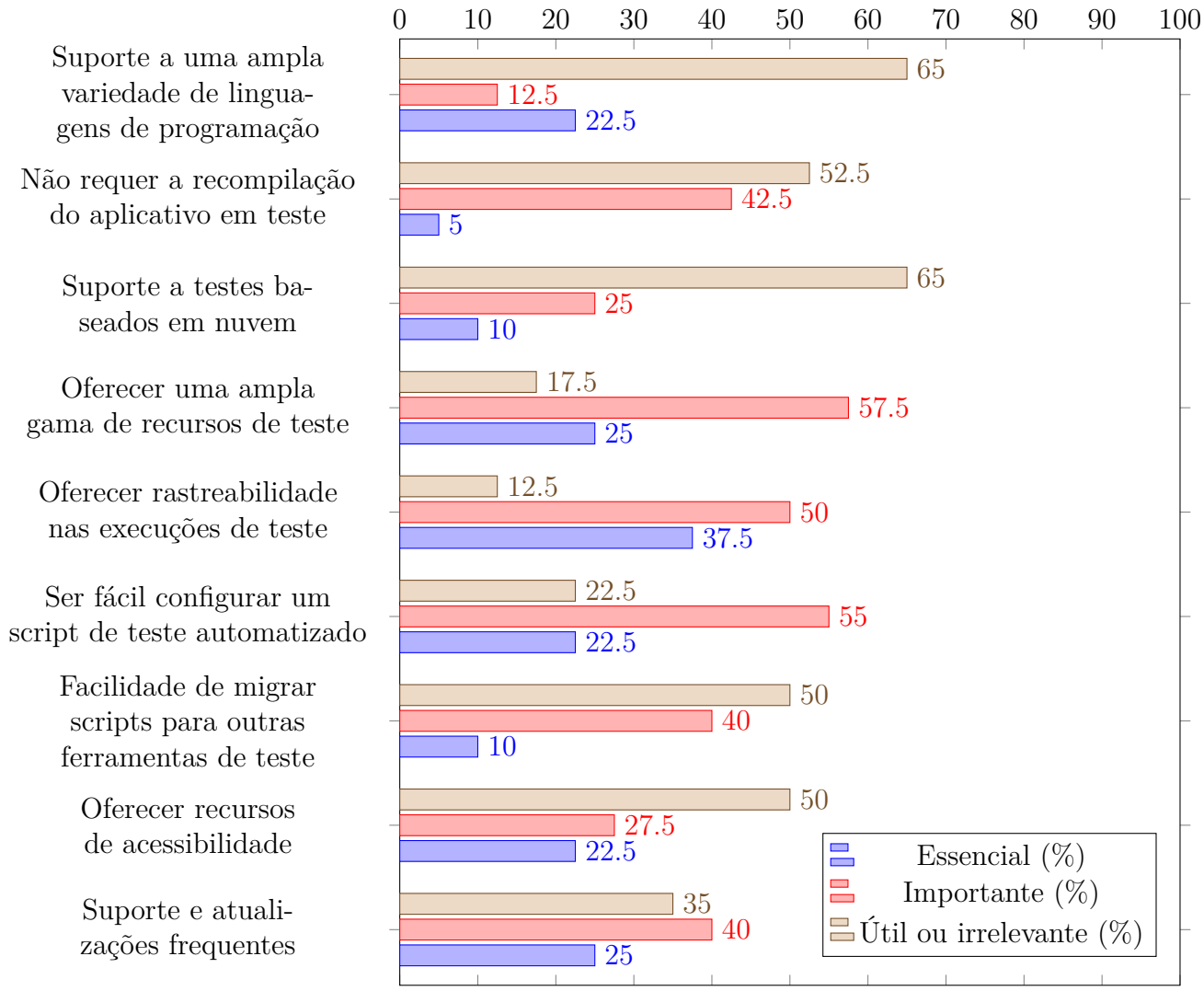

Figura 4.4: Vantagens desejáveis ao selecionar ferramentas Parte 2

80% dos participantes a considera inaceitável e 15% desafiador, nenhuma outra opção é considerada inaceitável por mais de 50% dos usuários com as opções requerer manutenção regular e geração de relatórios dependente de terceiros empatados em segunda posição com 30% as considerando inaceitável e 47.5% e 32.5% desafiador respectivamente.

O obstáculo da ferramenta possuir custo de licenciamento é considerado superável ou irrelevante por 57.5% dos participantes tornando essa a desvantagem menos impactante para os mesmos. O resultado é condizente com os dados obtidos na questão Q09 em que mais participantes consideram a vantagem da ferramenta ser gratuita útil ou irrelevante (27.5%) do que essencial (25%). Dessa forma, é possível inferir que os participantes estão dispostos a pagar por ferramentas que melhor saciam suas necessidades ao invés de se limitarem apenas as que possuem código aberto.

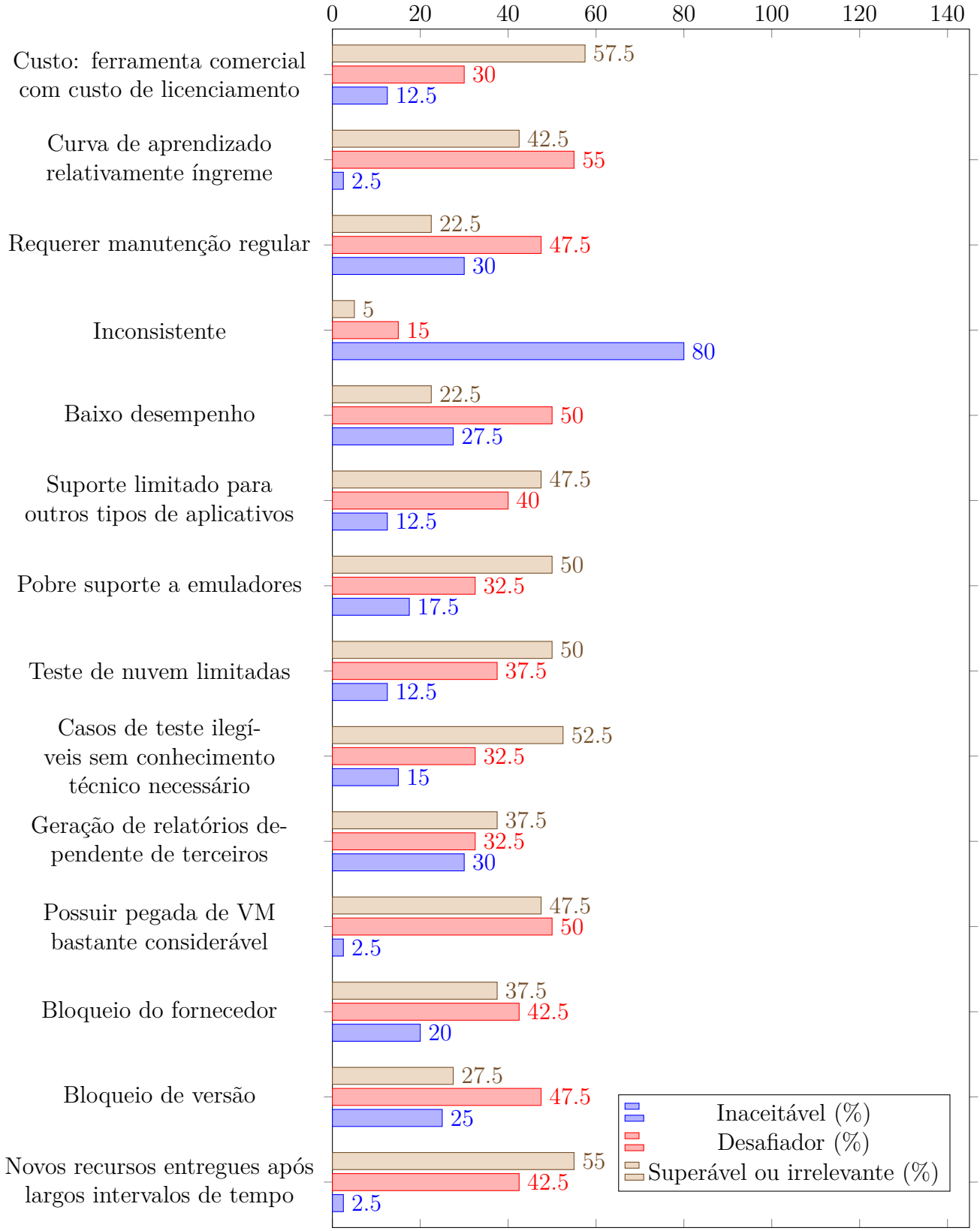

Figura 4.5: Desafios evitáveis ao selecionar ferramentas

4.3 Síntese das questões de pesquisa

RQ1: Quais as ferramentas de geração de testes utilizadas na literatura? Considerando apenas ferramentas de teste em aplicações mobile, as ferramentas Monkey e Dynodroid foram as mais mencionadas na literatura branca com ferramentas como Sapienz, Stoat, SwiftHand e Evodroid também possuindo bastante relevância.

RQ2: As ferramentas de geração de testes identificadas na literatura são usadas na indústria? A maior parte das ferramentas identificadas na literatura não são muito utilizadas e conhecidas sendo SOAP UI e JMeter as ferramentas mais utilizadas mas sendo algumas das menos mencionadas. As duas ferramentas mais mencionadas na literatura branca Monkey e Dynodroid não são utilizadas por nenhum participante sendo que 17.5% conhece Monkey e apenas 5% conhece Dynodroid.

RQ3: Quais as vantagens e desafios do uso das ferramentas de geração de teste? De acordo com os participantes, as seguintes vantagens são as mais desejadas ao utilizar uma ferramenta, sendo consideradas essenciais:

1. Logs visuais e resultados de execução de teste (52.5%): Fornecer visibilidade dos resultados da execução de testes, permitindo fácil depuração e/ou análise de falhas de testes.

2. Comunidade grande e ativa de usuários e desenvolvedores (37.5%): Indica que os usuários podem facilmente encontrar ajuda e suporte quando necessário.

3. Rastreabilidade nas execuções de teste (37,5%): Fornecer rastreabilidade em todo o processo, o que garante controle mais rigoroso do teste.

4. Bibliotecas abrangentes de testes e integrações (35%): Incluir um conjunto abrangente de bibliotecas de teste e integrações integradas com ferramentas e estruturas de desenvolvimento.

Em relação as maiores desafios, de acordo com os participantes os seguintes desafios são considerados inaceitáveis e os que mais impactam em suas escolhas de ferramenta:

1. Inconsistência (80%): Possibilidade de não registrar todas as ações ou respostas do objeto corretamente, aumentando a possibilidade de falhas nos testes.

2. Requerer manutenção regular (30%): Requerer manutenção regular para evitar que testes que dependam de recursos integrados sejam interrompidos.

3. Geração de relatórios dependente de terceiros (30%): Não possuir recursos de relatórios integrados e requerer ferramentas adicionais para gerar relatórios.

4. Baixo desempenho (27.5%): Testes podem ser mais lentos em comparação com testes unitários ou outras estruturas de teste.

4.4 Limitações e Ameaças para Validar o Estudo

Segundo Wohlin et al. [192] há quatro classificações de tipos de ameaça à validade dos resultados em experimentos: Validade de conclusão, interna, construção e externa.

1. Conclusão: Questões que afetam a capacidade de tirar a conclusão correta sobre as relações entre o tratamento e o resultado de um experimento.

2. Interna: Influências que podem afetar a variável independente no que diz respeito à causalidade, sem o conhecimento do pesquisador.

3. Construção: Generalização do resultado do experimento para o conceito ou teoria por trás do experimento.

4. Externa: Condições que limitam a capacidade de generalizar os resultados da experiência para a prática industrial.

Ao avaliar possíveis limitações e ameaças ao validar este estudo, identificamos as seguintes ameaças:

Validade de Conclusão: Não é possível garantir que todas as ferramentas e frameworks relevantes, relacionados à aplicação de testes de software em todas as áreas, puderam ser selecionados para o desenvolvimento do guia referencial prático. Para mitigar essa ameaça, realizou-se as pesquisas em diversas bibliotecas digitais diferentes e em diferentes literaturas branca e cinza, a fim de que a escolha das ferramentas pudesse ser mais abrangente.

Validade Interna: Como o survey foi realizado de forma assíncrona e sem monitorização, os participantes podem ter finalizado o mesmo apressadamente sem prestar a devida atenção, afetando a qualidade das respostas. Para mitigar a possibilidade do acontecimento, redigiu-se um aviso legal no começo do questionário, objetivando informar que a decisão de participar é voluntária e completamente anônima. Dessa forma, espera-se que apenas pessoas interessadas no assunto se sentiram motivadas a participar do trabalho.

Validade de Construção: Durante a realização do processo de snowballing para a obtenção de estudos relacionados a ferramentas de testes, observou-se que a grande maioria das ferramentas obtidas eram prioritariamente ou exclusivamente voltadas a testes

mobile trazendo pouca representatividade aos demais tipos de teste. Para mitigar essa ameaça, utilizou-se uma grande variedade de literatura cinza de forma a suprir a falta de conteúdo acerca dos testes voltados a API, Desktop e Web.

Validade Externa: A divulgação do survey também se manifesta como ameaça uma vez que sua divulgação de forma manual e pela plataforma LinkedIn pode ter sugerido pessoas interconectadas em relação a formação acadêmica e localidade. O tamanho da amostra utilizada foi uma limitação para o estudo uma vez que deve-se ter cautela ao generalizar os resultados obtidos para toda a população brasileira de desenvolvedores levando em conta a seleção de apenas um número limitado de participantes. Além do fato de não haver garantia de que todas as áreas de teste foram devidamente representadas, uma vez que pode haver muitos participantes especializados em testes Web e muito poucos em testes mobile, por exemplo. Para mitigar essas ameaças, a pesquisa foi divulgada em outros grupos de diferentes plataformas para profissionais da área, mas não é possível garantir que a seleção tenha sido totalmente arbitrária.

4.5 Guia Referencial Prático

O objetivo do guia referencial proposto é auxiliar os desenvolvedores de software a escolher a ferramenta de teste mais adequada de acordo com o cenário em que melhor se enquadra. O guia utiliza as ferramentas encontradas durante a revisão multivocal da literatura e os principais resultados obtidos na survey indicando o que os profissionais da área mais procuram ao selecionar uma ferramenta para seus projetos.

O guia é dividido em quatro partes especializadas em uma determinada área de teste (API; Mobile; Desktop; Web), cada área seleciona e organiza as ferramentas de teste mais relevantes as separando por tipos de teste (Functional; Regression; UI; Load) e por preço (Open source; Commercial). Dessa forma é possível rapidamente identificar quais ferramentas são recomendadas para cada cenário especifico. Em seguida, apresenta-se uma tabela comparativa indicando informações úteis acerca das ferramentas da área de forma a auxiliar o usuário a escolher a ferramenta que melhor se enquadra ao seu caso especifico dentre as recomendadas.

De forma simples, o usuário primeiro identifica as ferramentas recomendadas de acordo com a sua área, tipo de teste e preço e em seguida compara as ferramentas encontradas na tabela de forma a encontrar a ferramenta mais adequada para seu projeto atual. O guia é apresentado nas Figuras 4.6, 4.7, 4.8, e 4.9 e complementado pelas Tabelas 4.8, 4.9, 4.10 e 4.11

	Soap UI	Katalon	Postman

Ease of Learning	Basic features are easy to use, more complex scenarios require a deeper understanding of the tool	Entry-level knowledge to Java/Groovy for debugging	Easy for developers and testers to understand and navigate the tool
Community Support	Vibrant community, with numerous forums and resources available for help and guidance	Weak support	Large and active community of users and developers
Record-Playback	Does not support	The Web Recorder Utility function captures actions being performed on the application and converts them into executable code in the back-end	Does not support
Scripting language	Groovy and JavaScript	Groovy	JavaScript
Report generation	Has inbuilt feature to generate reports after the test execution. The report document presents number of test cases, test passed and failed in the report along with the test graphs	Presents analytic results in the form of built-in reports that can be exported in PDF, HTML, Excel, or CSV formats	API documentation includes complete API, path, and operation information, such as authentication methods, parameters, request bodies, response bodies and headers, and examples

API Requests	SOAP/WSDL, REST, GraphQL, JMS	REST: Swagger, OpenAPI 3.0 & WADL, SOAP (SOAP 1.1 & 1.2), GraphQL, Authentication: Basic, Bearer, OAuth 1.0, OAuth 2.0, and NTLM	HTTP, REST, SOAP, GraphQL e WebSockets. Authentication: OAuth 1.2/2.0, AWS Signature, Hawk
	JMeter	TestComplete	LoadRunner
Ease of Learning	GUI is user-friendly. Advanced features and concepts can have a steep learning curve	Relatively steep learning curve	Steep learning curve
Community Support	Has a vibrant community that has developed a wide range of plugins to extend its functionality	Has an active and supportive user community	Strong community and professional support
Record-Playback	Supports record and replay option, the recorded scenario can be edited using JavaScript. The Blazemeter plugin can further be used for editing the recorded script to modify the test case.	Allows record and play back HTTP requests	Support. Captures the application operations based on protocols
Scripting language	Groovy, Beansheel, Javascript, Java	JavaScript	C

Report generation	Provides various built-in listeners and reporting tools that help analyze test results and identify performance bottlenecks	Provides visual logs and test execution results. Offers detailed reports with screenshots and test coverage information, aiding in test analysis	Provides graphical representation of results through LoadRunner Analysis
API Requests	HTTP, HTTPS, FTP, JDBC, SOAP, REST, WebSocket	REST	HTTP, HTTPS

Tabela 4.8: Ferramentas recomendadas para teste API

	Appium	Katalon	Robot	TestComplete
Ease of Learning	Requires a certain level of programming knowledge in order to design the test scripts correctly	Easy for testers with little to no coding experience to create and run automated tests	Basic features are easy to use, more complex scenarios require a deeper understanding of the tool	Relatively steep learning curve
Community Support	Large and active community of users and developers	Weak support	Has a wide range of libraries available but some specific or niche libraries may have limited community support	Has an active and supportive user community

Record-Playback	Has an Appium Inspector tool to record and playback. It records and plays native application behavior by inspecting DOM and generates the test script	It has a very simple and easy-to-use interface. Easily capture elements, edit a recorded test, or re-use it to create more automated test cases	Does not have any built-in recorder utility	Script-free record and replay tool allows for recording multi-touch gestures (swipe, pinch, drag, drop, or scroll) and playing them back
Language support	Java, Ruby, Python, C#, JavaScript, PHP	Java, Groovy	Own domain-specific language (DSL). Python, Java, JavaScript	JavaScript, Python
Report generation	Does not have built-in reporting capabilities and requires additional tools to generate reports	Reports are easy to read while also consisting all the important details needed to know in case of test fails	Generates detailed reports and logs automatically, providing visibility into test execution results	Provides reporting capabilities that allow view and export detailed reports
Platforms	iOS, Android, Tizen, Native application, Web mobile, Hybrid	Android, iOS, Native application, Web mobile, Hybrid	Android, iOS, Native application, Web mobile, Hybrid	Android, iOS, Native application, Web mobile, Hybrid
	Ranorex	JMteter	LoadRunner	
Ease of Learning	User-friendly, can be used by any testing team and organisation	GUI is user-friendly. Advanced features and concepts can have a steep learning curve	Steep learning curve	

Community Support	Community support is not as extensive as some open-source tools	Has a vibrant community that has developed a wide range of plugins to extend its functionality	Strong community and professional support	
Record-Playback	Offers many low-code features including capture-and-replay functionality to record tests	Doesn't have a built-in record-and-playback feature. It provides a proxy server component called the "HTTP(S) Test Script Recorder"that allows to record HTTP requests	LoadRunner's Mobile Protocol can record the interactions of a mobile application and later replay those interactions to simulate multiple virtual users accessing the application simultaneously	
Language support	C#, VB.net,	Java	VuGen (Virtual User Generator) script, it is specific to LoadRunner	
Report generation	Provides detailed reports after test execution, including information about test results, screenshots, and logs	Generates HTML report that provides detailed information about response times, throughput, error rates, and more	Generates detailed reports to analyze the results and identify performance bottlenecks	

Platforms	Android, iOS, Native application, Web mobile, Hybrid	Android (with plugins)	iOS, Android	

Tabela 4.9: Ferramentas recomendadas para teste móvel

	Katalon	Robot Framework	TestComplete
Ease of Learning	Easy for testers with little to no coding experience to create and run automated tests	Basic features are easy to use, more complex scenarios require a deeper understanding of the tool	Relatively steep learning curve
Community Support	Weak support	Has a wide range of libraries available but some specific or niche libraries may have limited community support	Has an active and supportive user community
Record-Playback	Does not provide native record and playback functionality	Does not have any built-in recorder utility	Offers a record and playback feature, allowing testers to record their interactions with an application and generate test scripts automatically
Language support	Java, Groovy	Own domain-specific language (DSL). Python, Java, C#	Python, JavaScript
Report generation	Reports are easy to read while also consisting all the important details needed to know in case of test fails	Generates detailed reports and logs automatically, providing visibility into test execution results	Provides reporting capabilities that allow view and export detailed reports

Platforms	Windows	Windows, Linux, MAC	Windows
	Ranorex Studio	Apache JMeter	
Ease of Learning	User-friendly, can be used by any testing team and organisation	Steep Learning Curve	
Community Support	Community support is not as extensive as some open-source tools	Has a vibrant community that has developed a wide range of plugins to extend its functionality	
Record-Playback	Fully interchangeable and modifiable recording and coding modules. Capture entire workflows for replay or convert actions to user code and modify it	Doesn't have a built-in record-and-playback feature. It provides a proxy server component called the "HTTP(S) Test Script Recorder"that allows to record HTTP requests	
Language support	C#, VB.net	BeanShell	
Report generation	Fully customizable reports with screenshots, text, and videos	Provides various built-in listeners and reporting tools that help to analyze test results and identify performance bottlenecks	
Platforms	Windows	Windows, Linux, MAC	

Tabela 4.10: Ferramentas recomendadas para teste de desktop

	Selenium	Cypress	Katalon	Robot

Ease of Learning	Challenging	Easy to learn	Easy for testers with little to no coding experience to create and run automated tests	Basic features are easy to use, more complex scenarios require a deeper understanding of the tool
Community Support	Large and active community of users and developers	Has a strong and active community support	Weak support	Has a wide range of libraries available but some specific or niche libraries may have limited community support
Record-Playback	Selenium IDE records test runs and exports the recording into TestNG or JUnit test cases	Does not have a traditional record-and-playback feature	Supports. Provides a browser extension that facilitates the recording of interactions with web elements	Does not have any built-in recorder utility. Its necessary to install third-party browser extension
Language support	Java, PHP, C#, Python, Groovy, Ruby, Perl	JavaScript	Java, Groovy	Own domain-specific language (DSL). Python, Java, JavaScript
Report generation	Does not have its own built-in reporting tool	HtmlReport generation using Mocha	Reports are easy to read while also consisting all the important details needed to know in case of test fails	Generates detailed reports and logs automatically, providing visibility into test execution results

Browsers support	Chrome, Firefox, Internet Explorer, Edge, Safari	Chrome, Edge	Chrome, Firefox, Edge, Safari	Chrome, Firefox, Internet Explorer, Edge
	TestComplete	Ranorex Studio	Jmeter	
Ease of Learning	Relatively steep learning curve	User-friendly, can be used by any testing team and organisation	Steep Learning Curve	
Community Support	Has an active and supportive user community	Community support is not as extensive as some open-source tools	Has a vibrant community that has developed a wide range of plugins to extend its functionality	
Record-Playback	Offers a record and playback feature, allowing testers to record their interactions with an application and generate test scripts automatically	Fully interchangeable and modifiable recording and coding modules. Capture entire workflows for replay or convert actions to user code and modify it	Doesn't have a built-in record-and-playback feature. It provides a proxy server component called the "HTTP(S) Test Script Recorder"that allows to record HTTP requests	
Language support	Python, JavaScript	C#, VB.net	BeanShell	

Report generation	Provides reporting capabilities that allow view and export detailed reports	Provides detailed reports after test execution, including information about test results, screenshots, and logs	Provides various built-in listeners and reporting tools that help to analyze test results and identify performance bottlenecks	
Browsers support	Chrome, Firefox, Internet Explorer, Edge, Safari	Chrome, Firefox, Safari, Edge	Primarily operates at the protocol level and is not tied to specific browsers	

Tabela 4.11: Ferramentas recomendadas para teste web

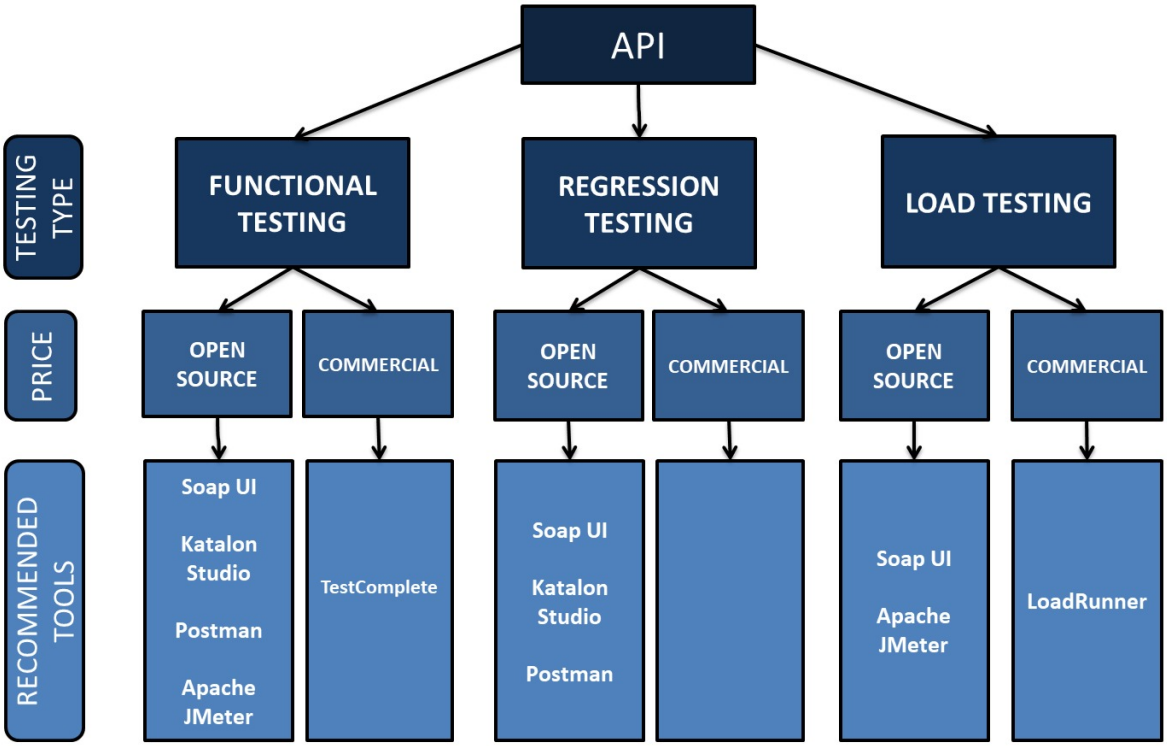

Figura 4.6: Guia prático: API Testing

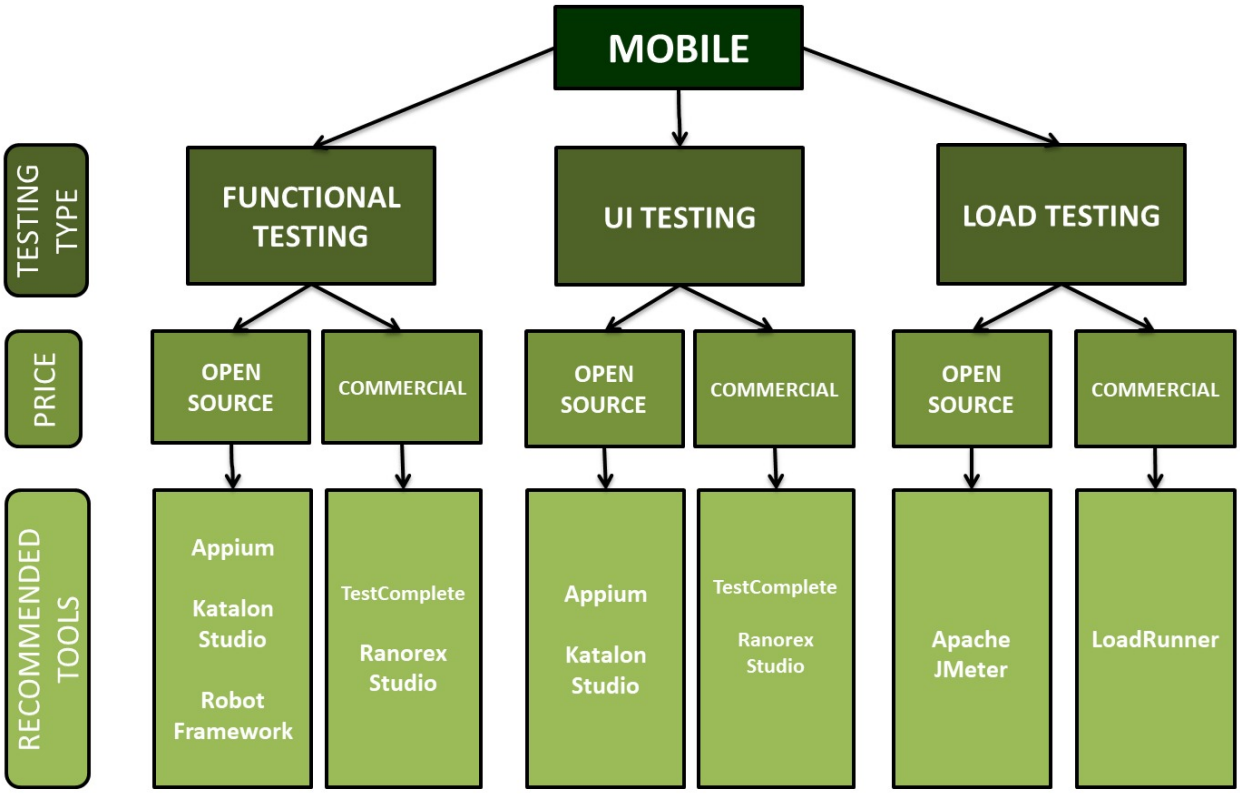

Figura 4.7: Guia prático: Mobile Testing

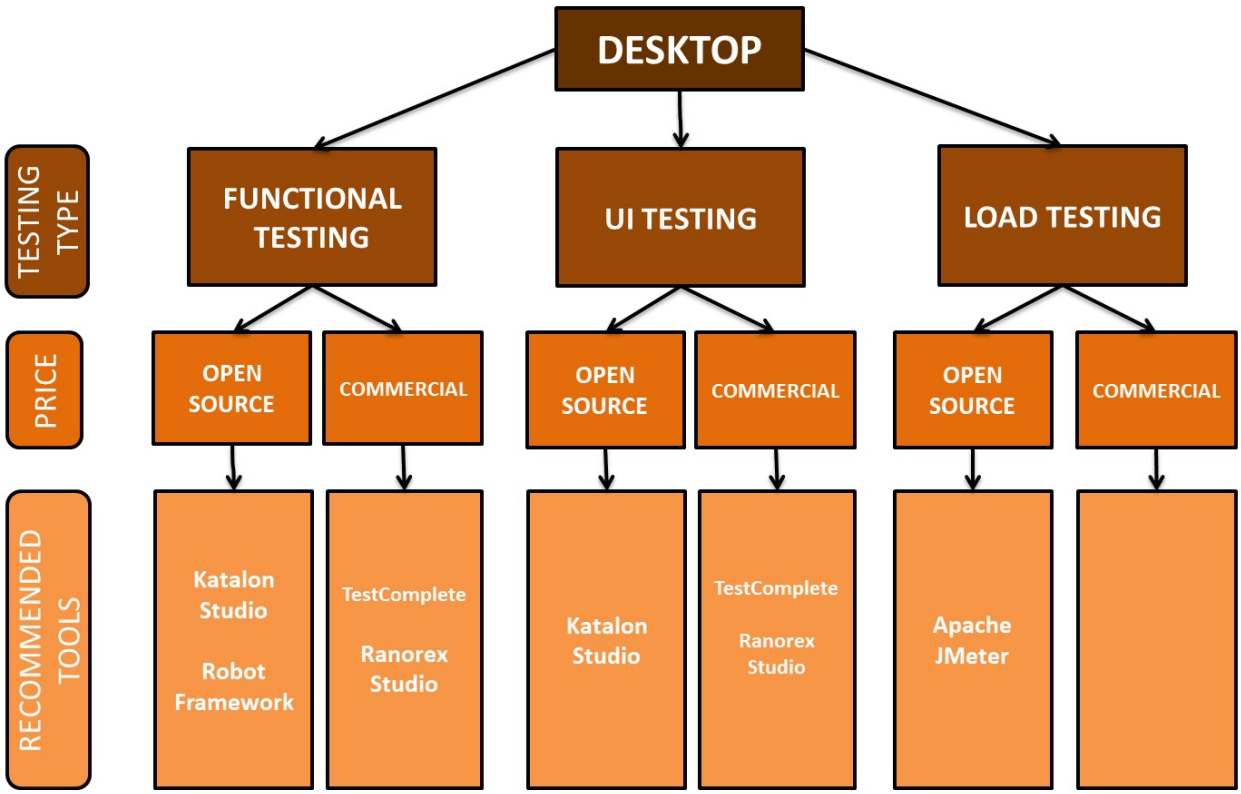

Figura 4.8: Guia prático: Desktop Testing

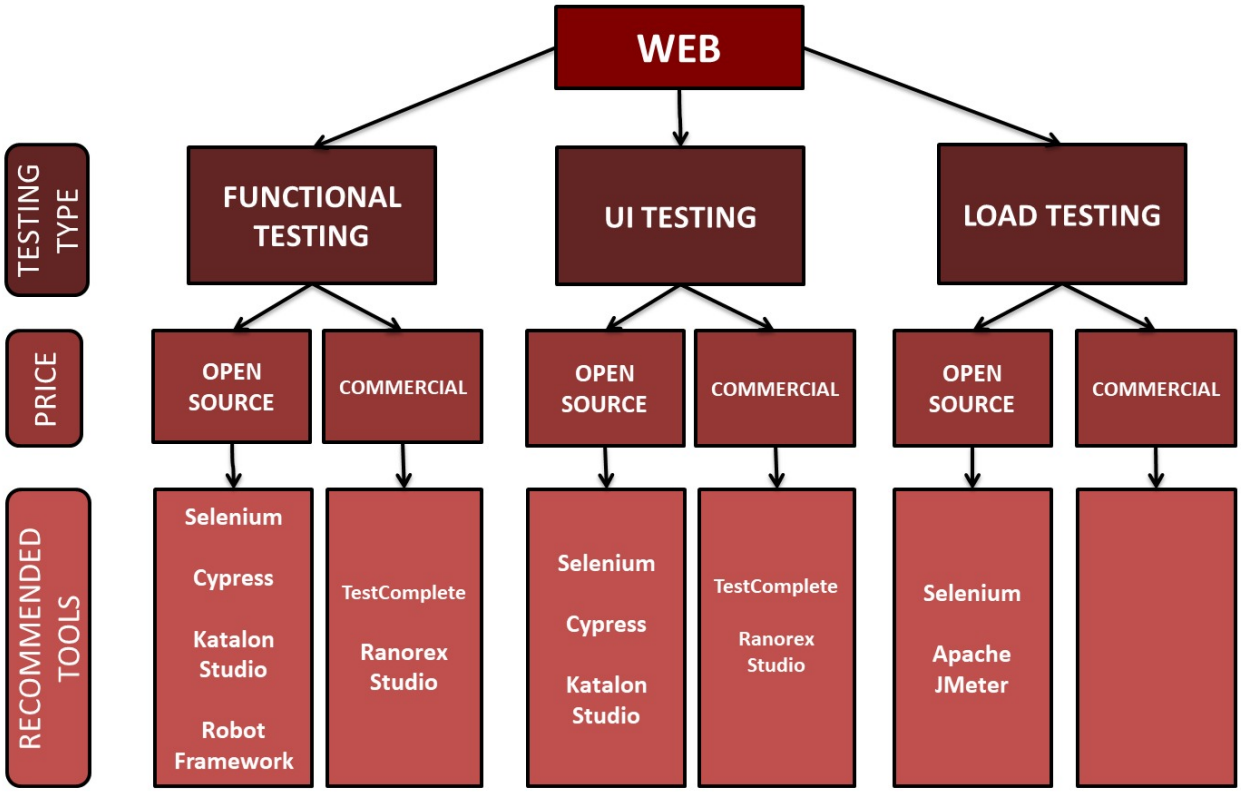

Figura 4.9: Guia prático: Web Testing

Capítulo 5

Conclusão

Neste trabalho foi elaborado uma revisão multivocal da literatura com o intuito de identificar ferramentas de geração de teste mencionadas na literatura e na industria. Algumas das ferramentas encontradas foram listadas e brevemente comentadas de acordo com os estudos de onde foram retiradas. Elaborou-se um survey com o objetivo de avaliar a familiaridade dos profissionais de software em relação a ferramentas mencionadas em literatura branca assim como cinza além de identificar as principais vantagens e desvantagens que os mesmos priorizam ao escolher uma determinada ferramenta de teste em um projeto.

As ferramentas mencionadas em literatura cinza são consideravelmente mais conhecidas e utilizadas pelos participantes sendo que as ferramentas Postman, Selenium e Cypress são conhecidas por respectivamente 87,6%, 85% e 72,5% dos participantes e utilizadas por respectivamente 72,5%, 42,5% e 40% dos mesmos. As vantagens mais valorizadas ao escolher uma ferramenta foram respectivamente: Logs visuais e resultados de execução de teste, comunidade grande e ativa de usuários e desenvolvedores, rastreabilidade nas execuções de teste e bibliotecas abrangentes de testes e integrações sendo considerados características essenciais por 52.5%, 37,5%, 37,5% e 35% dos participantes respectivamente. Em relação as desvantagens mais evitadas, as características de inconsistência, requerer manutenção regular, geração de relatórios dependente de terceiros e baixo desempenho foram consideradas inaceitáveis por 80%, 30%, 30% e 27.5% respectivamente.

Os resultados obtidos durante a revisão multivocal da literatura e a análise do survey foram combinados de forma a gerar um guia referencial prático para auxiliar profissionais na área de desenvolvimento de software a escolher uma ferramenta mais adequada a seus projetos. Os profissionais dessa forma podem identificar as ferramentas mais recomendadas de acordo com um cenário especifico levando em conta área (Api, mobile, web e desktop), tipo de teste (functional, regression, UI e Load) e preço (Open source e commercial). Após isso utilizam uma tabela comparativa de forma a avaliar a ferramenta mais adequada para o projeto dentre as recomendadas.

Em relação a trabalhos futuros, pode-se experimentar uma maior quantidade dos profissionais da área, a fim de que seja conduzida uma pesquisa em uma metodologia quantitativa. Além disso, é possível realizar a pesquisa de forma que os participantes avaliem sua experiência com determinadas ferramentas de testes e também objetive identificar áreas em que há grande demanda e que ferramentas não ofereçam suporte suficiente.

Referências

[1] Mendoza-Gonzalez, Ricardo, Sergio Luján-Mora, Salvador Otón, Mary Sánchez-Gordón, Mario Rodríguez-Díaz, and Ricardo Reyes-Acosta: *Guidelines to establish an office of student accessibility services in higher education institutions.* Sustainability, 14:2635, February 2022. x, 5

[2] Lewis, William E: *Software testing and continuous quality improvement.* CRC press, 2017. x, 7

[3] Syaikhuddin, Muhammad Miftakhul, Choirul Anam, Ade Rizky Rinaldi, and Moch El Bahar Conoras: *Conventional software testing using white box method.* Kinetik: game technology, information system, computer network, computing, electronics, and control, pages 65–72, 2018. x, 21

[4] Adams, Richard J., Palie Smart, and Anne Sigismund Huff: *Shades of grey: Guidelines for working with the grey literature in systematic reviews for management and organizational studies.* International Journal of Management Reviews, 19(4):432–454, October 2017, ISSN 1460-8545. x, 24

[5] Wohlin, Claes: *Guidelines for snowballing in systematic literature studies and a replication in software engineering.* In Shepperd, Martin J., Tracy Hall, and Ingunn Myrtveit (editors): *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014*, pages 38:1–38:10. ACM, 2014. `https://doi.org/10.1145/2601248.2601268`. x, 30, 31

[6] Bourque, Pierre and Richard E. Fairley (editors): *SWEBOK: Guide to the Software Engineering Body of Knowledge.* IEEE Computer Society, Los Alamitos, CA, version 3.0 edition, 2014, ISBN 978-0-7695-5166-1. `http://www.swebok.org/`. xi, 1, 11, 12, 13, 17, 18

[7] Rana, Isha, Pooja Goswami, and Himani Maheshwari: *A review of tools and techniques used in software testing.* Int. J. Emerg. Technol. Innovative Res, 6(4):262–266, 2019. xi, 12, 14

[8] Garousi, Vahid, Michael Felderer, and Mika V Mäntylä: *Guidelines for including grey literature and conducting multivocal literature reviews in software engineering.* Information and software technology, 106:101–121, 2019. xi, 27, 28, 29, 30

[9] Jones, Capers and Olivier Bonsignour: *The economics of software quality.* Addison-Wesley Professional, 2011. 1

[10] Kan, Stephen H., Victor R. Basili, and L. N. Shapiro: *Software quality: An overview from the perspective of total quality management.* IBM Syst. J., 33(1):4–19, 1994. https://doi.org/10.1147/sj.331.0004. 1

[11] Naik, Kshirasagar and Priyadarshi Tripathy: *Software testing and quality assurance: theory and practice.* John Wiley & Sons, 2011. 1, 2

[12] Ammann, Paul and Jeff Offutt: *Introduction to Software Testing.* Cambridge University Press, 2008, ISBN 978-0-521-88038-1. https://doi.org/10.1017/CBO9780511809163. 1

[13] Ahamed, S. S. Riaz: *Studying the feasibility and importance of software testing: An analysis.* CoRR, abs/1001.4193, 2010. http://arxiv.org/abs/1001.4193. 1

[14] Li, Yuanchun, Ziyue Yang, Yao Guo, and Xiangqun Chen: *Droidbot: a lightweight ui-guided test input generator for android.* In Uchitel, Sebastián, Alessandro Orso, and Martin P. Robillard (editors): *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, pages 23–26. IEEE Computer Society, 2017. https://doi.org/10.1109/ICSE-C.2017.8. 2, 3, 30, 37

[15] Tripathy, Priyadarshi and Kshirasagar Naik: *Software testing and quality assurance: theory and practice.* John Wiley & Sons, 2011. 2, 9

[16] Garousi, Vahid and Mika V. Mäntylä: *When and what to automate in software testing? a multi-vocal literature review.* Information and Software Technology, 76:92–117, 2016, ISSN 0950-5849. https://www.sciencedirect.com/science/article/pii/S0950584916300702. 2

[17] Poston, Robert M. and Michael P. Sexton: *Evaluating and selecting testing tools.* IEEE Software, 9(3):33–42, 1992. 3

[18] Raulamo-Jurvanen, Päivi, Mika Mäntylä, and Vahid Garousi: *Choosing the right test automation tool: a grey literature review of practitioner sources.* In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 21–30, 2017. 3

[19] Kaur, Manjit and Raj Kumari: *Comparative study of automated testing tools: Testcomplete and quicktest pro.* International Journal of Computer Applications, 24(1):1–7, 2011. 3

[20] Brereton, Pearl, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil: *Lessons from applying the systematic literature review process within the software engineering domain.* Journal of systems and software, 80(4):571–583, 2007. 4

[21] Linåker, Johan, Efi Papatheocharous, and Thomas Olsson: *How to characterize the health of an open source software project? a snowball literature review of an emerging practice.* In *Proceedings of the 18th International Symposium on Open Collaboration*, pages 1–12, 2022. 5

[22] Hopgood, F. R. A.: *"structured programming, " by O.-J. dahl, e. w. dijkstra and c. a. r. hoare (book review).* Int. J. Man Mach. Stud., 6(3):377, 1974. https://doi.org/10.1016/S0020-7373(74)80028-3. 9

[23] Neto, Arilo and Dias Neto: *Introdução a teste de software.* Engenharia de Software Magazine, June 2023. 10

[24] Ammann, Paul and Jeff Offutt: *Introduction to Software Testing.* Cambridge University Press, 2nd edition, 2016. 10

[25] Jorgensen, Paul C: *Software testing: a craftsman's approach.* CRC press, 2018. 10, 11

[26] Chauhan, Rasneet Kaur and Iqbal Singh: *Latest research and development on software testing techniques and tools.* International Journal of Current Engineering and Technology, 4(4):2368–2372, 2014. 10

[27] Myers, Glenford J, Tom Badgett, Todd M Thomas, and Corey Sandler: *The art of software testing*, volume 2. Wiley Online Library, 2004. 11, 12, 14

[28] Shaukat, Kamran, Usman Shaukat, Faran Feroz, Shahraiz Kayani, and Ali Akbar: *Taxonomy of automated software testing tools.* International Journal of Computer Science and Innovation, 1:7–18, 2015. 12, 14, 17

[29] Taley, Divyani Shivkumar and Bageshree Pathak: *Comprehensive study of software testing techniques and strategies: a review.* Int. J. Eng. Res, 9(08):817–822, 2020. 12

[30] Sawant, Abhijit A, Pranit H Bari, and PM Chawan: *Software testing techniques and strategies.* International Journal of Engineering Research and Applications (IJERA), 2(3):980–986, 2012. 12

[31] Gamido, Heidilyn Veloso and Marlon Viray Gamido: *Comparative review of the features of automated software testing tools.* International Journal of Electrical and Computer Engineering, 9(5):4473, 2019. 12, 17

[32] Khan, Mohd Ehmer and Farmeena Khan: *A comparative study of white box, black box and grey box testing techniques.* International Journal of Advanced Computer Science and Applications, 3(6), 2012. 14

[33] Kaur, Manpreet and Rupinder Singh: *A review of software testing techniques.* International Journal of Electronic and Electrical Engineering, 7(5):463–474, 2014. 14

[34] Bluemke, Ilona and Karol Kulesza: *A comparison of dataflow and mutation testing of java methods.* In *Dependable Computer Systems*, pages 17–30. Springer, 2011. 16

[35] Okezie, F, Isaac Odun-Ayo, and S Bogle: *A critical analysis of software testing tools.* In *Journal of Physics: Conference Series*, number 4 in *1378*, page 042030. IOP Publishing, 2019. 17, 47, 48

[36] Singh, Inderjeet and Bindia Tarika: *Comparative analysis of open source automated software testing tools: Selenium, sikuli and watir*. International Journal of Information & Computation Technology, 4(15):1507–1518, 2014. 17

[37] Tachiyama, Hiroki, Tetsuro Katayama, Yoshihiro Kita, Hisaaki Yamaba, and Naonobu Okazaki: *Prototype of test cases automatic generation tool bwdm based on boundary value analysis with vdm++*. In *International Conference on Artificial Life and Robotics*, pages 275–278, 2017. 19

[38] Hartman, Alan: *Model based test generation tools*. Agedis Consortium, URL: http://www. agedis. de/documents/ModelBasedTestGenerationTools_cs. pdf, 2002. 19

[39] Robinson, Harry: *Finite state model-based testing on a shoestring*. In *Proceedings of the 1999 International Conference on Software Testing Analysis and Review (STAR-WEST 1999)*. Citeseer, 1999. 19

[40] Cohen, D.M., S.R. Dalal, M.L. Fredman, and G.C. Patton: *The aetg system: an approach to testing based on combinatorial design*. IEEE Transactions on Software Engineering, 23(7):437–444, 1997. 19

[41] Maurer, P.M.: *Generating test data with enhanced context-free grammars*. IEEE Software, 7(4):50–55, 1990. 20

[42] Wijayasiriwardhane, Thareendhra, P.G. Wijayarathna, and Damitha Karunaratna: *An automated tool to generate test cases for performing basis path testing*. In *2011 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 95–101, September 2011, ISBN 978-1-4577-1113-8. 20

[43] Wang, Shuang and Jeff Offutt: *Comparison of unit-level automated test generation tools*. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 210–219, Fairfax, VA 22030, USA, 2009. IEEE. 21

[44] Wang, Wenyu, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie: *An empirical study of android test generation tools in industrial cases*. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 738–748, 2018. 21, 40, 41

[45] Shafique, Muhammad and Yvan Labiche: *A systematic review of state-based test tools*. International Journal on Software Tools for Technology Transfer, 17:59–76, 2015. 22

[46] Corradini, Davide, Amedeo Zampieri, Michele Pasqua, and Mariano Ceccato: *Empirical comparison of black-box test case generation tools for restful apis*. In *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 226–236, 2021. 22

[47] Gilchrist, Ian: *Software testing - testing across the entire software development lifecycle. by gerald D everett & raymond mcleod jr. published by IEEE press and wiley interscience, hoboken, nj, USA ISBN: 978-0-471-79371-7, 260 pages*. Softw. Test. Verification Reliab., 19(1):85–86, 2009. `https://doi.org/10.1002/stvr.384`. 23

[48] Finnegan, Gregory A: *Conference circuit: New frontiers in grey literature: Fourth international conference on grey literature.* College & Research Libraries News, 60(11):909–913, 1999. 23

[49] Mahood, Quenby, Dwayne Van Eerd, and Emma Irvin: *Searching for grey literature for systematic reviews: challenges and benefits.* Research synthesis methods, 5(3):221–234, 2014. 23

[50] Garousi, Vahid and Mika V Mäntylä: *A systematic literature review of literature reviews in software testing.* Information and Software Technology, 80:195–216, 2016. 24

[51] Paez, Arsenio: *Gray literature: An important resource in systematic reviews.* Journal of Evidence-Based Medicine, 10(3):233–240, 2017. 25

[52] Cook, Deborah J, Gordon H Guyatt, Gerard Ryan, Joanne Clifton, Lisa Buckingham, Andrew Willan, William Mcllroy, and Andrew D Oxman: *Should unpublished data be included in meta-analyses?: Current convictions and controversies.* Jama, 269(21):2749–2753, 1993. 25

[53] McAuley, Laura, Peter Tugwell, David Moher, *et al.*: *Does the inclusion of grey literature influence estimates of intervention effectiveness reported in meta-analyses?* The Lancet, 356(9237):1228–1231, 2000. 25

[54] Calderón, Alejandro, Mercedes Ruiz, and Rory V O'Connor: *A multivocal literature review on serious games for software process standards education.* Computer Standards & Interfaces, 57:36–48, 2018. 25

[55] Pati, Debajyoti and Lesa N Lorusso: *How to write a systematic review of the literature.* HERD: Health Environments Research & Design Journal, 11(1):15–30, 2018. 25

[56] Snyder, Hannah: *Literature review as a research methodology: An overview and guidelines.* Journal of business research, 104:333–339, 2019. 25

[57] Easterbrook, Steve, Janice Singer, Margaret Anne Storey, and Daniela Damian: *Selecting Empirical Methods for Software Engineering Research*, pages 285–311. Springer London, London, 2008, ISBN 978-1-84800-044-5. `https://doi.org/10.1007/978-1-84800-044-5_11`. 25

[58] Bertram, Dane: *Likert scales.* Retrieved November, 2(10):1–10, 2007. 32, 33

[59] Risi, Craig: *The pros and cons of different ui automation test tools - testcomplete*, 2023. `https://www.linkedin.com/pulse/pros-cons-different-ui-automation-test-tools-craig-risi/`, visited on 2023-07-17. 32, 33, 34, 35, 36

[60] Risi, Craig: *The pros and cons of different ui automation test tools - postman*, 2023. `https://www.linkedin.com/pulse/pros-cons-different-api-test-tools-postman-craig-risi/`, visited on 2023-07-17. 32, 34

[61] Risi, Craig: *The pros and cons of different ui automation test tools - cucumber*, 2023. https://www.linkedin.com/pulse/pros-cons-different-ui-automation-test-tools-cucumber-craig-risi/, visited on 2023-07-17. 32, 36

[62] Risi, Craig: *The pros and cons of different ui automation test tools - appium*, 2023. https://www.linkedin.com/pulse/pros-cons-different-ui-automation-test-tools-appium-craig-risi/, visited on 2023-07-17. 32, 34, 35, 36

[63] Risi, Craig: *The pros and cons of different ui automation test tools - tosca*, 2023. https://www.linkedin.com/pulse/pros-cons-different-ui-automation-test-tools-tosca-craig-risi/, visited on 2023-07-17. 32, 35, 36

[64] Risi, Craig: *The pros and cons of different ui automation test tools - uft*, 2023. https://www.linkedin.com/pulse/pros-cons-different-ui-automation-test-tools-uft-craig-risi/, visited on 2023-07-17. 32, 36

[65] Pacheco, Carlos and Michael D Ernst: *Randoop: feedback-directed random testing for java*. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 815–816, 2007. 37

[66] Fraser, Gordon and Andrea Arcuri: *Evosuite: automatic test suite generation for object-oriented software*. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 416–419, 2011. 37

[67] Wang, Shuang and Jeff Offutt: *Comparison of unit-level automated test generation tools*. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 210–219. IEEE, 2009. 37, 46

[68] Abbas, Rabiya, Zainab Sultan, and Shahid Nazir Bhatti: *Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege*. In *2017 international conference on communication technologies (comtech)*, pages 39–44. IEEE, 2017. 37

[69] Kumar, Yogesh *et al.*: *Comparative study of automated testing tools: selenium, soapui, hp unified functional testing and test complete*. Journal of Emerging Technologies and Innovative Research, 2(9):42–48, 2015. 37, 46

[70] Bertolino, Antonia and Andrea Polini: *Soa test governance: Enabling service integration testing across organization and technology borders*. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 277–286. IEEE, 2009. 37

[71] Machiry, Aravind, Rohan Tahiliani, and Mayur Naik: *Dynodroid: An input generation system for android apps*. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 224–234, 2013. 37, 40, 41, 42

[72] Choi, Wontae, George Necula, and Koushik Sen: *Guided gui testing of android apps with minimal restart and approximate learning*. Acm Sigplan Notices, 48(10):623–640, 2013. 37, 40, 42

[73] Galler, Stefan J and Bernhard K Aichernig: *Survey on test data generation tools: An evaluation of white-and gray-box testing tools for c#, c++, eiffel, and java.* International Journal on Software Tools for Technology Transfer, 16:727–751, 2014. 37, 41, 42, 43, 44, 45, 46

[74] Feng, Sidong, Haochuan Lu, Ting Xiong, Yuetang Deng, and Chunyang Chen: *Towards efficient record and replay: A case study in wechat.* arXiv preprint arXiv:2308.06657, 2023. 40

[75] Li, Yuanchun, Ziyue Yang, Yao Guo, and Xiangqun Chen: *Humanoid: A deep learning-based approach to automated black-box android app testing.* In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1070–1073. IEEE, 2019. 40, 41, 42

[76] Choudhary, Shauvik Roy, Alessandra Gorla, and Alessandro Orso: *Automated test input generation for android: Are we there yet?(e).* In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 429–440. IEEE, 2015. 40, 41, 42

[77] Amalfitano, Domenico, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon: *Using GUI ripping for automated testing of android applications.* In Goedicke, Michael, Tim Menzies, and Motoshi Saeki (editors): *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 258–261. ACM, 2012. https://doi.org/10.1145/2351676.2351717. 40, 43

[78] Hao, Shuai, Bin Liu, Suman Nath, William G. J. Halfond, and Ramesh Govindan: *PUMA: programmable ui-automation for large-scale dynamic analysis of mobile apps.* In Campbell, Andrew T., David Kotz, Landon P. Cox, and Zhuoqing Morley Mao (editors): *The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014*, pages 204–217. ACM, 2014. https://doi.org/10.1145/2594368.2594390. 40, 42

[79] Azim, Tanzirul and Iulian Neamtiu: *Targeted and depth-first exploration for systematic testing of android apps.* In Hosking, Antony L., Patrick Th. Eugster, and Cristina V. Lopes (editors): *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 641–660. ACM, 2013. https://doi.org/10.1145/2509136.2509549. 40, 46

[80] Bhoraskar, Ravi, Seungyeop Han, Jinseong Jeon, Tanzirul Azim, Shuo Chen, Jaeyeon Jung, Suman Nath, Rui Wang, and David Wetherall: *Brahmastra: Driving apps to test the security of third-party components.* In Fu, Kevin and Jaeyeon Jung (editors): *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 1021–1036. USENIX Association, 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/bhoraskar. 40

[81] Bierma, Michael, Eric Gustafson, Jeremy Erickson, David Fritz, and Yung Ryn Choe: *Andlantis: Large-scale android dynamic analysis.* CoRR, abs/1410.7751, 2014. `http://arxiv.org/abs/1410.7751`. 40

[82] Amalfitano, Domenico, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzung Ta, and Atif M Memon: *Mobiguitar: Automated model-based testing of mobile apps.* IEEE software, 32(5):53–59, 2014. 40, 41, 43

[83] Anand, Saswat, Mayur Naik, Mary Jean Harrold, and Hongseok Yang: *Automated concolic testing of smartphone apps.* In Tracz, Will, Martin P. Robillard, and Tevfik Bultan (editors): *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*, page 59, https://doi.org/10.1145/2393596.2393666, 2012. ACM. 40

[84] Mahmood, Riyadh, Nariman Mirzaei, and Sam Malek: *Evodroid: segmented evolutionary testing of android apps.* In Cheung, Shing-Chi, Alessandro Orso, and Margaret-Anne D. Storey (editors): *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 599–609, https://doi.org/10.1145/2635868.2635896, 2014. ACM. 40, 43

[85] Yang, Wei, Mukul R. Prasad, and Tao Xie: *A grey-box approach for automated gui-model generation of mobile applications.* In Cortellessa, Vittorio and Dániel Varró (editors): *Fundamental Approaches to Software Engineering - 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7793 of *Lecture Notes in Computer Science*, pages 250–265, https://doi.org/10.1007/978-3-642-37057-1_19, 2013. Springer. 40

[86] Clapp, Lazaro, Osbert Bastani, Saswat Anand, and Alex Aiken: *Minimizing GUI event traces.* In Zimmermann, Thomas, Jane Cleland-Huang, and Zhendong Su (editors): *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 422–434, https://doi.org/10.1145/2950290.2950342, 2016. ACM. 40, 41

[87] Gomez, Lorenzo, Iulian Neamtiu, Tanzirul Azim, and Todd D. Millstein: *RERAN: timing- and touch-sensitive record and replay for android.* In Notkin, David, Betty H. C. Cheng, and Klaus Pohl (editors): *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 72–81, https://doi.org/10.1109/ICSE.2013.6606553, 2013. IEEE Computer Society. 40

[88] Jensen, Casper Svenning, Mukul R. Prasad, and Anders Møller: *Automated testing with targeted event sequence generation.* In Pezzè, Mauro and Mark Harman (editors): *International Symposium on Software Testing and Analysis, ISSTA '13, Lugano, Switzerland, July 15-20, 2013*, pages 67–77, https://doi.org/10.1145/2483760.2483777, 2013. ACM. 40

[89] Kochhar, Pavneet Singh, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo: *Understanding the test automation culture of app developers.* In *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*, pages 1–10, https://doi.org/10.1109/ICST.2015.7102609, 2015. IEEE Computer Society. 40, 41, 44

[90] Mao, Ke, Mark Harman, and Yue Jia: *Sapienz: multi-objective automated testing for android applications.* In Zeller, Andreas and Abhik Roychoudhury (editors): *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, pages 94–105, https://doi.org/10.1145/2931037.2931054, 2016. ACM. 40, 41

[91] Mirzaei, Nariman, Hamid Bagheri, Riyadh Mahmood, and Sam Malek: *Sig-droid: Automated system input generation for android applications.* In *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*, pages 461–471. IEEE Computer Society, 2015. `https://doi.org/10.1109/ISSRE.2015.7381839`. 40, 41

[92] Qin, Zhengrui, Yutao Tang, Edmund Novak, and Qun Li: *Mobiplay: a remote execution based record-and-replay tool for mobile applications.* In Dillon, Laura K., Willem Visser, and Laurie A. Williams (editors): *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, pages 571–582, https://doi.org/10.1145/2884781.2884854, 2016. ACM. 40, 41

[93] Ravindranath, Lenin, Suman Nath, Jitendra Padhye, and Hari Balakrishnan: *Automatic and scalable fault detection for mobile applications.* In Campbell, Andrew T., David Kotz, Landon P. Cox, and Zhuoqing Morley Mao (editors): *The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014*, pages 190–203, https://doi.org/10.1145/2594368.2594377, 2014. ACM. 40, 41

[94] Yang, Shengqian, Dacong Yan, and Atanas Rountev: *Testing for poor responsiveness in android applications.* In *2013 1st international workshop on the engineering of mobile-enabled systems (MOBS)*, pages 1–6. IEEE, 2013. 40

[95] Jeon, Jinseong and Jeffrey S Foster: *Troyd: Integration testing for android.* Technical report, University of Maryland, 2012. 40

[96] Banerjee, Abhijeet, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury: *Detecting energy bugs and hotspots in mobile apps.* In Cheung, Shing-Chi, Alessandro Orso, and Margaret-Anne D. Storey (editors): *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 588–598. ACM, 2014. `https://doi.org/10.1145/2635868.2635871`. 40

[97] Yan, Dacong: *Program analyses for understanding the behavior and performance of traditional and mobile object-oriented software.* The Ohio State University, 2014. 40, 41

[98] Su, Ting, Jue Wang, and Zhendong Su: *Benchmarking automated gui testing for android against real-world bugs.* In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 119–130, 2021. 40, 41, 42, 43, 44, 46

[99] Dong, Zhen, Marcel Böhme, Lucia Cojocaru, and Abhik Roychoudhury: *Time-travel testing of android apps.* In Rothermel, Gregg and Doo-Hwan Bae (editors): *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 481–492. ACM, 2020. https://doi.org/10.1145/3377811.3380402. 40, 41, 44

[100] Wang, Wenyu, Wing Lam, and Tao Xie: *An infrastructure approach to improving effectiveness of android UI testing tools.* In Cadar, Cristian and Xiangyu Zhang (editors): *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, pages 165–176. ACM, 2021. https://doi.org/10.1145/3460319.3464828. 40

[101] Daragh, Faraz Yazdani Banafshe and Sam Malek: *Deep GUI: black-box GUI input generation with deep learning.* In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*, pages 905–916. IEEE, 2021. https://doi.org/10.1109/ASE51524.2021.9678778. 40, 41

[102] Almeida, Diego R, Patrícia DL Machado, and Wilkerson L Andrade: *Testing tools for android context-aware applications: a systematic mapping.* Journal of the Brazilian Computer Society, 25:1–22, 2019. 40, 41

[103] Yasin, Husam N, Siti Hafizah Ab Hamid, and Raja Jamilah Raja Yusof: *Droidbotx: Test case generation tool for android applications using q-learning.* Symmetry, 13(2):310, 2021. 40, 41, 44, 45

[104] Pilgun, Aleksandr, Olga Gadyatskaya, Yury Zhauniarovich, Stanislav Dashevskyi, Artsiom Kushniarou, and Sjouke Mauw: *Fine-grained code coverage measurement in automated black-box android testing.* ACM Transactions on Software Engineering and Methodology (TOSEM), 29(4):1–35, 2020. 40, 41

[105] Cai, Tianqin, Zhao Zhang, and Ping Yang: *Fastbot: A multi-agent model-based test generation system beijing bytedance network technology co., ltd.* In *AST@ICSE 2020: IEEE/ACM 1st International Conference on Automation of Software Test, Seoul, Republic of Korea, 15-16 July, 2020*, pages 93–96. ACM, 2020. https://doi.org/10.1145/3387903.3389308. 40, 41

[106] Köroglu, Yavuz and Alper Sen: *Reinforcement learning-driven test generation for android GUI applications using formal specifications.* CoRR, abs/1911.05403, 2019. http://arxiv.org/abs/1911.05403. 40

[107] Peng, Chao, Ajitha Rajan, and Tianqin Cai: *CAT: change-focused android GUI testing.* In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021*, pages 460–470. IEEE, 2021. https://doi.org/10.1109/ICSME52107.2021.00047. 40

[108] Wanwarang, Tanapuch, Nataniel P. Borges Jr., Leon Bettscheider, and Andreas Zeller: *Testing apps with real-world inputs.* In *AST@ICSE 2020: IEEE/ACM 1st International Conference on Automation of Software Test, Seoul, Republic of Korea, 15-16 July, 2020*, pages 1–10. ACM, 2020. https://doi.org/10.1145/3387903.3389310. 40, 41, 43

[109] Yasin, Husam N, Siti Hafizah Ab Hamid, Raja Jamilah Raja Yusof, and Muzaffar Hamzah: *An empirical analysis of test input generation tools for android apps through a sequence of events.* Symmetry, 12(11):1894, 2020. 40, 41

[110] Ma, Enze, Shan Huang, Weigang He, Ting Su, Jue Wang, Huiyu Liu, Geguang Pu, and Zhendong Su: *Automata-based trace analysis for aiding diagnosing gui testing tools for android.* ESEC/FSE, pages 1–12, 2023. 40, 41

[111] Ibrahim, Rosziati, Nurul Ain Aswini Abdul Jan, Sapiee Jamel, and Jahari Abdul Wahab: *Generating test cases using eclipse environment–a case study of mobile application.* International Journal of Advanced Computer Science and Applications, 12(4), 2021. 40, 41

[112] Sun, Jun: *A real-world, hybrid event sequence generation framework for android apps.* Computer Science and Engineering: Theses, Dissertations, and Student Research, 2021. 40, 41

[113] Osorio Riaño, Michael Stiven *et al.*: *Comparison and analysis between automatic exploration tools for android applications.* Universidad de los Andes repertory, 2020. 40

[114] Forte, Vincenzo Junior: *PoLiUToDroid: A Non-Invasive Automatic Black-Box UI Testing Technique for Android Mobile Applications based on a Novel Active Learning Approach.* PhD thesis, Politecnico di Torino, 2017. 40, 41

[115] Liu, Zhe, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang: *Fill in the blank: Context-aware automated text input generation for mobile GUI testing.* In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 1355–1367. IEEE, 2023. https://doi.org/10.1109/ICSE48619.2023.00119. 40, 41

[116] Feng, Sidong, Mulong Xie, and Chunyang Chen: *Efficiency matters: Speeding up automated testing with GUI rendering inference.* In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 906–918. IEEE, 2023. https://doi.org/10.1109/ICSE48619.2023.00084. 40, 41

[117] Lv, Zhengwei, Chao Peng, Zhao Zhang, Ting Su, Kai Liu, and Ping Yang: *Fastbot2: Reusable automated model-based GUI testing for android enhanced by reinforcement learning.* In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, pages 135:1–135:5. ACM, 2022. https://doi.org/10.1145/3551349.3559505. 40, 41

[118] Ran, Dezhi, Hao Wang, Wenyu Wang, and Tao Xie: *Badge: Prioritizing UI events with hierarchical multi-armed bandits for automated UI testing.* In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 894–905. IEEE, 2023. https://doi.org/10.1109/ICSE48619.2023.00083. 40, 41

[119] Sun, Jingling, Ting Su, Kai Liu, Chao Peng, Zhao Zhang, Geguang Pu, Tao Xie, and Zhendong Su: *Characterizing and finding system setting-related defects in android apps.* IEEE Trans. Software Eng., 49(4):2941–2963, 2023. 40

[120] Zhao, Yu, Brent Harrison, and Tingting Yu: *Dinodroid: Testing android apps using deep q-networks.* CoRR, abs/2210.06307, 2022. https://doi.org/10.48550/arXiv.2210.06307. 40, 41

[121] Hu, Jiajun, Lili Wei, Yepang Liu, and Shing-Chi Cheung: *ωtest: Webview-oriented testing for android applications.* In Just, René and Gordon Fraser (editors): *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, pages 992–1004. ACM, 2023. https://doi.org/10.1145/3597926.3598112. 40, 41

[122] Liu, Zhe, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang: *Chatting with GPT-3 for zero-shot human-like mobile automated GUI testing.* CoRR, abs/2305.09434, 2023. https://doi.org/10.48550/arXiv.2305.09434. 40, 41

[123] Su, Ting, Yichen Yan, Jue Wang, Jingling Sun, Yiheng Xiong, Geguang Pu, Ke Wang, and Zhendong Su: *Fully automated functional fuzzing of android apps for detecting non-crashing logic bugs.* Proceedings of the ACM on Programming Languages, 5(OOPSLA):1–31, 2021. 40, 41

[124] Ghorbani, Negar, Reyhaneh Jabbarvand, Navid Salehnamadi, Joshua Garcia, and Sam Malek: *Deltadroid: Dynamic delivery testing in android.* ACM Trans. Softw. Eng. Methodol., 32(4):84:1–84:26, 2023. https://doi.org/10.1145/3563213. 40, 41

[125] Peng, Chao, Zhao Zhang, Zhengwei Lv, and Ping Yang: *Mubot: Learning to test large-scale commercial android apps like a human.* In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2022, Limassol, Cyprus, October 3-7, 2022*, pages 543–552. IEEE, 2022. https://doi.org/10.1109/ICSME55016.2022.00074. 40, 41

[126] Ran, Dezhi, Zongyang Li, Chenxu Liu, Wenyu Wang, Weizhi Meng, Xionglin Wu, Hui Jin, Jing Cui, Xing Tang, and Tao Xie: *Automated visual testing for mobile apps in an industrial seting.* In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 55–64, 2022. 40

[127] Zhong, Yi, Mengyu Shi, Youran Xu, Chunrong Fang, and Zhenyu Chen: *Iterative android automated testing.* Frontiers Comput. Sci., 17(5):175212, 2023. https://doi.org/10.1007/s11704-022-1658-8. 40, 41

[128] Ngo, Chanh Duc, Fabrizio Pastore, and Lionel C. Briand: *Automated, cost-effective, and update-driven app testing.* ACM Trans. Softw. Eng. Methodol., 31(4):61:1–61:51, 2022. https://doi.org/10.1145/3502297. 40, 41

[129] Ngo, Chanh Duc, Fabrizio Pastore, and Lionel C. Briand: *Testing updated apps by adapting learned models.* CoRR, abs/2308.05549, 2023. https://doi.org/10.48550/arXiv.2308.05549. 40, 41

[130] Dong, Zhen, Marcel Böhme, Lucia Cojocaru, and Abhik Roychoudhury: *Timemachine: Time-travel testing of android apps.* 42nd International Conference on Software Engineering, 2020. 40, 41

[131] Guo, Wunan, Liwei Shen, Ting Su, Xin Peng, and Weiyang Xie: *Improving automated GUI exploration of android apps via static dependency analysis.* In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020*, pages 557–568. IEEE, 2020. https://doi.org/10.1109/ICSME46990.2020.00059. 40, 41

[132] Costa, Francisco Handrick da, Ismael Medeiros, Thales Menezes, João Victor da Silva, Ingrid Lorraine da Silva, Rodrigo Bonifácio, Krishna Narasimhan, and Márcio Ribeiro: *Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification.* J. Syst. Softw., 183:111092, 2022. https://doi.org/10.1016/j.jss.2021.111092. 40, 41

[133] Peng, Chao, Zhengwei Lv, Jiarong Fu, Jiayuan Liang, Zhao Zhang, Ajitha Rajan, and Ping Yang: *Hawkeye: Change-targeted testing for android apps based on deep reinforcement learning.* CoRR, abs/2309.01519, 2023. https://doi.org/10.48550/arXiv.2309.01519. 40, 41

[134] Zhao, Yan, Weihao Zhang, Enyi Tang, Haipeng Cai, Xi Guo, and Na Meng: *A lightweight approach of human-like playtesting.* CoRR, abs/2102.13026, 2021. https://arxiv.org/abs/2102.13026. 40, 41

[135] Zhao, Yu: *Automated testing and bug reproduction of android apps.* University of Kentucky Institutional Repository, 2020. 40, 41

[136] Sun, Jingling, Ting Su, Junxin Li, Zhen Dong, Geguang Pu, Tao Xie, and Zhendong Su: *Understanding and finding system setting-related defects in android apps.* In Cadar, Cristian and Xiangyu Zhang (editors): *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, pages 204–215. ACM, 2021. https://doi.org/10.1145/3460319.3464806. 40

[137] Wang, Jue, Yanyan Jiang, Chang Xu, Chun Cao, Xiaoxing Ma, and Jian Lu: *Combodroid: generating high-quality test inputs for android apps via use case combinations.* In Rothermel, Gregg and Doo-Hwan Bae (editors): *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 469–480. ACM, 2020. https://doi.org/10.1145/3377811.3380382. 40, 41

[138] Lai, Duling and Julia Rubin: *Goal-driven exploration for android applications*. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 115–127. IEEE, 2019. https://doi.org/10.1109/ASE.2019.00021. 40, 41

[139] He, Yuyu, Lei Zhang, Zhemin Yang, Yinzhi Cao, Keke Lian, Shuai Li, Wei Yang, Zhibo Zhang, Min Yang, Yuan Zhang, and Haixin Duan: *Textexerciser: Feedback-driven text input exercising for android applications*. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1071–1087. IEEE, 2020. https://doi.org/10.1109/SP40000.2020.00071. 40, 41

[140] Wang, Wenyu, Wei Yang, Tianyin Xu, and Tao Xie: *Vet: identifying and avoiding UI exploration tarpits*. In Spinellis, Diomidis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (editors): *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pages 83–94. ACM, 2021. https://doi.org/10.1145/3468264.3468554. 40, 41

[141] Yan, Jiwei, Hao Zhou, Xi Deng, Ping Wang, Rongjie Yan, Jun Yan, and Jian Zhang: *Efficient testing of GUI applications by event sequence reduction*. Sci. Comput. Program., 201:102522, 2021. https://doi.org/10.1016/j.scico.2020.102522. 40

[142] Doyle, Jordan, Takfarinas Saber, Paolo Arcaini, and Anthony Ventresque: *Improving mobile user interface testing with model driven monkey search*. In *14th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2021, Porto de Galinhas, Brazil, April 12-16, 2021*, pages 138–145. IEEE, 2021. https://doi.org/10.1109/ICSTW52544.2021.00034. 40, 41

[143] Zhao, Yan, Enyi Tang, Haipeng Cai, Xi Guo, Xiaoyin Wang, and Na Meng: *A lightweight approach of human-like playtest for android apps*. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022*, pages 309–320. IEEE, 2022. https://doi.org/10.1109/SANER53432.2022.00047. 40, 41

[144] Pilgun, Aleksandr: *Instruction Coverage for Android App Testing and Tuning*. PhD thesis, University of Luxembourg, Luxembourg City, Luxembourg, 2020. http://orbilu.uni.lu/handle/10993/45355. 40, 41

[145] Hatas, Veysel, Sevil Sen, and John A. Clark: *Efficient evolutionary fuzzing for android application installation process*. In *19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019, Sofia, Bulgaria, July 22-26, 2019*, pages 62–68. IEEE, 2019. https://doi.org/10.1109/QRS.2019.00021. 40, 41

[146] Behrang, Farnaz and Alessandro Orso: *Seven reasons why: An in-depth study of the limitations of random test input generation for android*. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*, pages 1066–1077. IEEE, 2020. https://doi.org/10.1145/3324884.3416567. 40, 41, 42, 44, 45

[147] Yang, Sen, Song Huang, and Zhanwei Hui: *Theoretical analysis and empirical evaluation of coverage indictors for closed source APP testing.* IEEE Access, 7:162323–162332, 2019. https://doi.org/10.1109/ACCESS.2019.2951941. 40, 41

[148] Gebrekrstos, Selam Welu, Teklit Berihu Gereziher: *Search-based test generation framework for android apps with support for multiobjective generation: Stgfa-smog.* Master's thesis 2022, 2022. 40, 41

[149] SPEK, Corné: *Analyzing the code coverage of android apps using the exerciser monkey.* LIACS Thesis Repository, 2021. 40, 41

[150] Huang, Song: *Equivalent version sets testing method for android applications based on code analysis.* International Journal of Performability Engineering, 2008. 40, 41

[151] Yu, Shengcheng, Chunrong Fang, Yuchen Ling, Chentian Wu, and Zhenyu Chen: *LLM for test script generation and migration: Challenges, capabilities, and opportunities.* CoRR, abs/2309.13574, 2023. https://doi.org/10.48550/arXiv.2309.13574. 40

[152] Escobar Velásquez, Camilo Andrés: *On improving analysis and testing of open-and closed-source android apps.* Universidad de los Andes repertory, 2023. 40, 41

[153] Pantoja Gómez, Camila, Edgar Camilo Díaz Suárez: *Automatic multi-platform interaction testing for android using reinforcement learning.* Universidad de los Andes repertory, 2022. 40, 41

[154] Valbuena Bautista, Daniel: *Automatic gui testing for android using reinforcement learning.* Universidad de los Andes repertory, 2023. 40, 41

[155] Moreno, Iván Arcuschin, Juan Pablo Galeotti, and Diego Garbervetsky: *An empirical study on how sapienz achieves coverage and crash detection.* J. Softw. Evol. Process., 35(4), 2023. https://doi.org/10.1002/smr.2411. 40, 41

[156] Pilgun, Aleksandr: *Don't trust me, test me: 100% code coverage for a 3rd-party android app.* In *27th Asia-Pacific Software Engineering Conference, APSEC 2020, Singapore, December 1-4, 2020*, pages 375–384. IEEE, 2020. https://doi.org/10.1109/APSEC51365.2020.00046. 40

[157] Leanidavich Arnatovich, Yauhen and Lipo Wang: *A systematic literature review of automated techniques for functional gui testing of mobile applications.* arXiv e-prints, pages arXiv–1812, 2018. 40, 41

[158] Wang, Jue, Yanyan Jiang, Ting Su, Shaohua Li, Chang Xu, Jian Lu, and Zhendong Su: *Detecting non-crashing functional bugs in android apps via deep-state differential analysis.* In Roychoudhury, Abhik, Cristian Cadar, and Miryung Kim (editors): *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, pages 434–446. ACM, 2022. https://doi.org/10.1145/3540250.3549170. 40, 41

[159] Pan, Minxue, Yifei Lu, Yu Pei, Tian Zhang, and Xuandong Li: *Preference-wise testing of android apps via test amplification.* ACM Trans. Softw. Eng. Methodol., 32(1):4:1–4:37, 2023. https://doi.org/10.1145/3511804. 40, 41

[160] Gross, Florian, Gordon Fraser, and Andreas Zeller: *Search-based system testing: high coverage, no false alarms.* In Heimdahl, Mats Per Erik and Zhendong Su (editors): *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, pages 67–77, https://doi.org/10.1145/2338965.2336762, 2012. ACM. 41, 43

[161] Gross, Florian, Gordon Fraser, and Andreas Zeller: *EXSYST: search-based GUI testing.* In Glinz, Martin, Gail C. Murphy, and Mauro Pezzè (editors): *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 1423–1426, https://doi.org/10.1109/ICSE.2012.6227232, 2012. IEEE Computer Society. 41

[162] Fraser, Gordon and Andrea Arcuri: *Whole test suite generation.* IEEE Transactions on Software Engineering, 39(2):276–291, 2012. 41

[163] Brunetto, Matteo, Giovanni Denaro, Leonardo Mariani, and Mauro Pezzè: *On introducing automatic test case generation in practice: A success story and lessons learned.* J. Syst. Softw., 176:110933, 2021. https://doi.org/10.1016/j.jss.2021.110933. 41

[164] Potuzak, Tomas and Richard Lipka: *Current trends in automated test case generation.* In Ganzha, Maria, Leszek A. Maciaszek, Marcin Paprzycki, and Dominik Slezak (editors): *Proceedings of the 18th Conference on Computer Science and Intelligence Systems, FedCSIS 2023, Warsaw, Poland, September 17-20, 2023*, volume 35 of *Annals of Computer Science and Information Systems*, pages 627–636, 2023. https://doi.org/10.15439/2023F9829. 41

[165] Devroey, Xavier, Sebastiano Panichella, and Alessio Gambi: *Java unit testing tool competition: Eighth round.* In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*, pages 545–548. ACM, 2020. https://doi.org/10.1145/3387940.3392265. 41, 43

[166] Just, René, Darioush Jalali, and Michael D. Ernst: *Defects4j: a database of existing faults to enable controlled testing studies for java programs.* In Pasareanu, Corina S. and Darko Marinov (editors): *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pages 437–440. ACM, 2014. https://doi.org/10.1145/2610384.2628055. 41

[167] Nistor, Adrian, Linhai Song, Darko Marinov, and Shan Lu: *Toddler: detecting performance problems via similar memory-access patterns.* In Notkin, David, Betty H. C. Cheng, and Klaus Pohl (editors): *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 562–571. IEEE Computer Society, 2013. https://doi.org/10.1109/ICSE.2013.6606602. 41

[168] Yoo, Shin and Mark Harman: *Regression testing minimization, selection and prioritization: a survey.* Software testing, verification and reliability, 22(2):67–120, 2012. 41

[169] Zadgaonkar, Hrushikesh: *Robotium automated testing for android.* Packt Publishing Birmingham, 2013. 41

[170] Adamsen, Christoffer Quist, Gianluca Mezzetti, and Anders Møller: *Systematic execution of android test suites in adverse conditions.* In Young, Michal and Tao Xie (editors): *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*, pages 83–93. ACM, 2015. https://doi.org/10.1145/2771783.2771786. 41

[171] Mirshokraie, Shabnam, Ali Mesbah, and Karthik Pattabiraman: *PYTHIA: generating test cases with oracles for javascript applications.* In Denney, Ewen, Tevfik Bultan, and Andreas Zeller (editors): *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 610–615. IEEE, 2013. https://doi.org/10.1109/ASE.2013.6693121. 41

[172] Anand, Saswat: *Techniques to facilitate symbolic execution of real-world programs.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2012. https://hdl.handle.net/1853/44733. 41, 44

[173] Park, Sangmin, B. M. Mainul Hossain, Ishtiaque Hussain, Christoph Csallner, Mark Grechanik, Kunal Taneja, Chen Fu, and Qing Xie: *Carfast: achieving higher statement coverage faster.* In Tracz, Will, Martin P. Robillard, and Tevfik Bultan (editors): *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*, page 35. ACM, 2012. https://doi.org/10.1145/2393596.2393636. 41

[174] Cheng, Yaxin: *Ape+: A faster ape with static model guided exploration.* University of Waterloo, 1, 2022. 41

[175] Hao, Dan, Lu Zhang, Xingxia Wu, Hong Mei, and Gregg Rothermel: *On-demand test suite reduction.* In Glinz, Martin, Gail C. Murphy, and Mauro Pezzè (editors): *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 738–748. IEEE Computer Society, 2012. https://doi.org/10.1109/ICSE.2012.6227144. 41

[176] Xu, Dianxiang, Weifeng Xu, Manghui Tu, Ning Shen, William Chu, and Chih Hung Chang: *Automated integration testing using logical contracts.* IEEE Transactions on Reliability, 65(3):1205–1222, 2015. 41, 42

[177] Zhang, Jie, Ziyi Wang, Lingming Zhang, Dan Hao, Lei Zang, Shiyang Cheng, and Lu Zhang: *Predictive mutation testing.* In Zeller, Andreas and Abhik Roychoudhury (editors): *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, pages 342–353. ACM, 2016. https://doi.org/10.1145/2931037.2931038. 41

[178] Bures, Miroslav: *Automated testing in the czech republic: the current situation and issues*. In Rachev, Boris and Angel Smrikarov (editors): *Proceedings of the 15th International Conference on Computer Systems and Technologies, CompSysTech '14, Ruse, Bulgaria, June 27-28, 2014*, pages 294–301. ACM, 2014. https://doi.org/10.1145/2659532.2659605. 41

[179] Tillmann, Nikolai and Jonathan de Halleux: *White-box testing of behavioral web service contracts with pex*. In Bultan, Tevfik and Tao Xie (editors): *Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications, held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), TAV-WEB 2008, Seattle, Washington, USA, July 21, 2008*, pages 47–48. ACM, 2008. https://doi.org/10.1145/1390832.1390840. 44

[180] Alferidah, Saja Khalid and Shakeel Ahmed: *Automated software testing tools*. In *2020 International Conference on Computing and Information Technology (ICCIT-1441)*, pages 1–4. IEEE, 2020. 46

[181] Umar, Mubarak Albarka and Chen Zhanfang: *A study of automated software testing: Automation tools and frameworks*. International Journal of Computer Science Engineering (IJCSE), 6:217–225, 2019. 47, 48

[182] Ateşoğulları, Dilara and Alok Mishra: *Automation testing tools: a comparative view*. International Journal on Information Technologies & Security, 12(4):63–76, 2020. 47, 48

[183] Islam, Nazia: *A comparative study of automated software testing tools*. Culminating Projects in Computer Science and Information Technology, 2016. 47

[184] Oliinyk, Bohdan and Vasyl Oleksiuk: *Automation in software testing, can we automate anything we want?* 2nd Student Workshop on Computer Science e Software Engineering, 2019. 47

[185] Gunasekaran, S and V Bargavi: *Survey on automation testing tools for mobile applications*. International Journal of Advanced Engineering Research and Science, 2(11):2349–6495, 2015. 47

[186] S, Ramya.: *Software engineering automated software testing*. International Journal of Engineering Research & Technology, 2016. 48

[187] Pragya Sen, Savitri Tangirala, Dr. Saba Farheen. N: *A comparative study of automation testing tools for mobile, web and desktop applications*. International Journal of Emerging Technologies and Innovative Research, 10:332–337, 2023. 48

[188] Advaith Aditya Chevuturu, Divyendra Pratap Mathur, Byreddy Joseph Prasanth Kumar Reddy: *A comparative survey on software testing tool*. International Journal of Engineering and Advanced Technology, 11, 2022. 48

[189] Dimoski, Davor, Bojana Koteska, LJupcho Pejov, and Anastas Mishev: *Testing restful apis–use case: Restful api for solving multidimensional time–independent schrödinger equation.* Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, 2022. 48

[190] Stresnjak, Stanislav and Zeljko Hocenski: *Usage of robot framework in automation of functional test regression.* In *Proc. 6th Int. Conf. Softw. Eng. Adv.(ICSEA)*, pages 30–34, 2011. 48, 49

[191] Grechanik, Mark, Qing Xie, and Chen Fu: *Maintaining and evolving gui-directed test scripts.* In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 408–418. IEEE, 2009. https://doi.org/10.1109/ICSE.2009.5070540. 54, 55

[192] Wohlin, Claes, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén: *Experimentation in software engineering.* Springer Science & Business Media, 2012. 59