STENIO HENRIQUE MACHADO SILVA

FERRAMENTAS PARA AUTOMATIZAÇÃO DE TESTES DE INTERFACE DE USUÁRIO PARA APLICAÇÕES ANDROID: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

LAVRAS – MG

2023

STENIO HENRIQUE MACHADO SILVA

FERRAMENTAS PARA AUTOMATIZAÇÃO DE TESTES DE INTERFACE DE USUÁRIO PARA APLICAÇÕES ANDROID:
UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Monografia apresentada ao Departamento de Ciência da Computação para obtenção do título de Bacharel em Sistemas de Informação.

Prof. Dr. Paulo Afonso Parreira Júnior
Orientador

LAVRAS – MG

2023

STENIO HENRIQUE MACHADO SILVA

Ferramentas para Automatização de Testes de Interface de Usuário para Aplicações Android: Um Mapeamento Sistemático da Literatura
Tools for User Interface Testing Automation for Android Applications: A Systematic Literature Mapping

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Sistemas de Informação, para obtenção do título de Bacharel.

APROVADA em 1 de Março de 2023.

Profa. Dra. Renata Teles Moreira	UFLA
Prof. Dr. Bruno de Abreu Silva	UFLA

Prof. Dr. Paulo Afonso Parreira Júnior
(Orientador)

LAVRAS - MG

2023

Aos meus pais, que sempre me incentivaram, ao meu irmão por toda a ajuda durante o curso e a minha namorada pela compreensão e carinho com que sempre me apoiou.

AGRADECIMENTOS

Aos meus pais, Cristianny e Wanderlei que sempre me incentivaram nos estudos e não mediram esforços para me ajudar nessa caminhada.

Ao meu irmão, Stanley, que me auxiliou em várias disciplinas, a minha irmã Vitória pelos incentivos.

A minha namorada, Maria Emanuely, que com muito amor esteve ao meu lado durante este processo.

Ao professor Paulo, por aceitar ser meu orientador e direcionar de forma didática e com muita paciência a execução deste trabalho.

Aos professores Renata e Bruno por aceitarem participar da banca do meu TCC e auxiliarem para conseguir alcançar os meus objetivos.

A todos os professores que lecionaram disciplinas e contribuíram de alguma forma para meu crescimento pessoal, profissional e acadêmico.

A todos os meus amigos e familiares que estiveram comigo nessa caminhada.

Muito obrigado!

RESUMO

O uso de dispositivos móveis tem sido uma crescente no mercado, com os *smartphones* sendo uma constante no dia a dia das pessoas e se tornando uma ferramenta essencial na vida de muita gente. Uma peça essencial para o aumento do uso dos *smartphones* são os aplicativos de diversos propósitos que estão disponíveis para serem baixados sempre que necessário. A qualidade de um aplicativo é um fator fundamental, que define se ele será bem aceito pelo público. Sendo assim, manter a qualidade é importante e os testes são parte intrínseca do processo de garantia da qualidade. Contudo, testes manuais são repetitivos, caros e demorados. Por isso, é necessário buscar ferramentas que automatizem essas ações e tornem os testes mais dinâmicos e com resultados confiáveis. Dessa forma, este trabalho buscou realizar um Mapeamento Sistemático da Literatura (MSL) para identificar quais ferramentas estão disponíveis atualmente para automatização de testes de GUI (*Graphical User Interface*) em aplicações para *Android*, dado que este é o sistema operacional mais utilizado no mercado de dispositivos móveis. A partir da execução do MSL, 12 artigos foram lidos e analisados e 21 ferramentas foram catalogadas e classificadas, destacando suas características, benefícios e desvantagens.

Palavras-chave: 1. Android, 2. Testes automatizados, 3. Interface gráfica de usuário

ABSTRACT

The use of mobile devices has been growing in the market, with smartphones being a constant in people's daily lives and becoming an essential tool. A crucial piece for the increase in the use of smartphones is the various purpose applications that are available for download whenever necessary. The quality of an application is a fundamental factor that defines whether it will be well accepted by the public. Therefore, maintaining quality is important and testing is an intrinsic part of the quality assurance process. However, manual tests are repetitive, expensive, and time-consuming. That is why it is necessary to seek tools that automate these actions and make tests more dynamic and with reliable results. Thus, this work aimed to conduct a Systematic Mapping of Literature (SML) to identify what tools are currently available for automating GUI (Graphical User Interface) testing in applications for Android, given that this is the most used operating system in the mobile device market. Based on the execution of the SML, 12 articles were read and analyzed, and 21 tools were cataloged and classified, highlighting their characteristics, benefits, and disadvantages.

Keywords: 1. Android, 2. Automated tests, 3. Graphical user interface

LISTA DE FIGURAS

Figura 2.1 – Exemplo de *script* de teste, em Java, da ferramenta “Espresso” 12

Figura 3.1 – Critérios de seleção de estudos da literatura cinza 17

Figura 4.1 – Quantidade de artigos por ano. 21
Figura 4.2 – Linguagens utilizadas nas ferramentas 25
Figura 4.3 – Quantidade de citações por ferramenta. 26
Figura 4.4 – Ferramentas mais pesquisadas no *Google* nos últimos doze meses 28

LISTA DE TABELAS

Tabela 3.1 – Quantidade de artigos aceitos e rejeitados por etapa na literatura formal. . . 19
Tabela 3.2 – Listagem de artigos selecionados a partir da literatura formal. 19
Tabela 3.3 – Lista de artigos selecionados a partir da literatura cinza. 20

Tabela 4.1 – Ferramentas para automação de testes de GUI identificadas 22
Tabela 4.2 – Características das ferramentas analisadas 23
Tabela 4.3 – Ferramentas que utilizam linguagens de programação para criação de casos de teste. 25
Tabela 4.4 – Vantagens e desvantagens das ferramentas identificadas. 29

SUMÁRIO

1 INTRODUÇÃO 8
2 REFERENCIAL TEÓRICO 10
2.1 Ferramentas de teste automatizado 10
2.2 Trabalhos relacionados 12
3 MAPEAMENTO SISTEMÁTICO DA LITERATURA 14
3.1 Planejamento 14
3.1.1 Questões de pesquisa 14
3.1.2 Método de busca 15
3.1.3 *Strings* de busca 16
3.1.4 Critérios de seleção 16
3.2 Execução do MSL 17
3.2.1 Literatura formal 18
3.2.2 Literatura cinza 19
4 RESULTADOS E DISCUSSÃO 21
4.1 Questão de Pesquisa Q1 21
4.2 Questão de Pesquisa Q2 24
4.3 Questão de Pesquisa Q3 26
4.4 Questão de Pesquisa Q4 28
5 AMEAÇAS A VALIDADE 31
6 CONSIDERAÇÕES FINAIS 32
REFERÊNCIAS 34

1 INTRODUÇÃO

Aplicativos para dispositivos móveis estão cada vez mais presentes no dia a dia das pessoas. Com o avanço da pandemia de COVID-19, o uso de *smartphones* cresceu 27% no ano de 2021, ultrapassando a casa de 242 milhões de dispositivos, somente no Brasil (MEIRELLES, 2022). Dada a vasta quantidade de aplicativos disponíveis no mercado, pensar na qualidade desses aplicativos é essencial para atrair novos usuários e reter os pré-existentes. Para garantir a qualidade de um software, tal como um aplicativo móvel, uma das estratégias mais preconizadas pela Engenharia de Software é a realização de testes. Além de proverem uma forma de detecção de defeitos, testes servem como documentação para o software e também para garantir que o software funcionará após sofrer alterações (VALENTE, 2020).

Segundo Srivastaval e Dwivedi (2015), uma das atividades que mais consomem recursos (esforço, tempo, dinheiro, entre outros) durante o ciclo de desenvolvimento de um software são os testes, principalmente, quando estes são realizados de forma manual. Testes manuais consistem em um ou mais analistas executarem diversos cenários de uso, "imitando" um usuário regular do software. Segundo Vocke (2018), testes manuais são repetitivos e tediosos e o tédio, por sua vez, leva a erros, diminuindo a qualidade do produto final. Uma maneira de tentar mitigar esses problemas é por meio da realização de testes automatizados (LäMSä, 2017; VOCKE, 2018). A automação dos testes pode reduzir a quantidade de erros decorrentes de falhas humanas durante a realização dos testes e permite utilizar a mão-de-obra dos analistas de testes em outras atividades, por exemplo, para elaboração de casos de testes mais eficazes, análise de requisitos mais detalhada, entre outros. A automação de testes permite ainda que o software possa sofrer alterações de larga escala de forma mais segura, pois permite que regressões sejam executadas, a fim de garantir que o comportamento visível externamente do software não foi alterado (VOCKE, 2018).

Um dos aspectos mais importantes das aplicações móveis, em geral, é a Interface Gráfica de Usuário, em inglês *Graphical User Interface* (GUI), pois ela corresponde à "maneira" com a qual o usuário interage com o aplicativo (MIN; CAI, 2018). Dessa forma, a realização de testes de GUI é essencial para que se tenha um aplicativo de melhor qualidade, do ponto de vista da experiência do usuário (AHO et al., 2011). Apesar de ser possível realizar testes de GUI manuais, esse tipo de teste sofre dos problemas já relatados anteriormente. Por esse motivo, diversos trabalhos têm sido feitos, a fim de se propor ferramentas para automação de teste de GUI para aplicativos móveis (BAEK; BAE, 2016).

Contudo, é difícil encontrar, na literatura, um catálogo que reúna as principais informações sobre essas ferramentas, classificando-as de acordo com suas características, vantagens e desvantagens. Tal catálogo poderia ser útil tanto para acadêmicos quanto para profissionais da indústria, interessados em pesquisar e/ou usufruir dos serviços oferecidos por esse tipo de ferramenta. Assim, neste trabalho foi realizado um Mapeamento Sistemático da Literatura (MSL) envolvendo a literatura formal, de produção científica, e também a chamada "literatura cinza", composta por trabalhos produzidos por profissionais da indústria, mas que não foram publicados em veículos tradicionais de divulgação científica. O MSL conduzido visou responder às seguintes questões de pesquisa:

- Q1. Quais são as ferramentas existentes na literatura para automação de testes de GUI para aplicativos Android e quais são as suas principais características (nome, geração em que ela se encaixa[1], se é paga ou gratuita, entre outros)?
- Q2. Qual é a metodologia/*framework* utilizado para elaboração de casos de testes nestas ferramentas?
- Q3. Qual a "popularidade" das ferramentas identificadas?
- Q4. Quais são as principais vantagens e desvantagens destas ferramentas?

É importante ressaltar que este trabalho possui enfoque no sistema operacional Android, devido à sua relevância e alcance no mercado de dispositivos móveis. Segundo Meirelles (2022), dos 242 milhões de dispositivos móveis existentes no Brasil em 2021, 90% usam o sistema operacional Android, da empresa Google. Os aplicativos disponíveis para este sistema operacional são disponibilizados por meio da *Google Play Store*[2], loja de aplicativos oficial do Android, que possuía mais de 2,6 milhões de aplicações disponíveis até dezembro de 2022 (APPBRAIN, 2022).

Esta monografia está dividida da seguinte forma: no Capítulo 2 é abordado o referencial teórico, apresentando conceitos básicos sobre testes automatizados para *Android*, bem como os trabalhos relacionados. O Capítulo 3 apresenta a definição de Mapeamento Sistemático da Literatura, de forma sucinta, e o protocolo do MSL realizado neste trabalho. O Capítulo 4, por sua vez, traz os resultados e discussões a respeito do tema deste MSL e o Capítulo 5 apresenta as considerações finais e sugestões para trabalhos futuros.

[1] Mais detalhes sobre as gerações de ferramentas de teste automatizado são apresentados no Capítulo 2.
[2] https://play.google.com/store

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentadas as bases que motivaram o trabalho, evidenciando quais são as referências utilizadas para elaboração desta monografia e os principais conceitos que envolem os testes automatizados de GUI.

2.1 Ferramentas de teste automatizado

A interface gráfica do usuário, do inglês *Graphical User Interface* (GUI) é a maneira como o usuário interage com o sistema. Em resumo, a GUI recebe eventos, tais como cliques, seleções, arraste, inserções de textos, entre outros, e altera o estado dos elementos na tela.

Os testes de GUI são definidos por Banerjee et al. (2013) como sendo os testes nos quais sequências de eventos são executados nos elementos da GUI. Há muitas possibilidades de execuções diferentes dos elementos, sendo necessário que sejam criadas técnicas capazes de minimizar o trabalho e otimizar os resultados das execuções de testes em vários cenários possíveis.

Evidências empíricas sugerem que aplicativos móveis não são testados com o cuidado devido (COPPOLA et al., 2019). Neste contexto, dada a grande participação no mercado de dispositivos Android e a alta demanda de aplicativos para estes tipos de dispositivos, fica evidente que a área de testes de aplicativos móveis precisa de atenção (COPPOLA; MORISIO; TORCHIANO, 2019).

Com base nisso, várias abordagens diferentes para testes automatizados de aplicativos móveis surgiram. Segundo Min e Cai (2018), essas abordagens deram origem a três gerações de ferramentas de testes automatizados.

As ferramentas que compõem a **primeira geração** buscam os elementos da interface do aplicativo a serem testados por meio de coordenadas da tela. Porém, o custo de manutenção desse tipo de teste é alto, pois uma simples mudança na tela do aplicativo gera a necessidade de reimplementação dos casos de teste. Já as ferramentas de **segunda geração** interagem com os elementos por meio do uso das chamadas "propriedades" dos elementos gráficos, tais como *índices*, *rótulos*, *identificadores*, entre outros. Com o uso destas propriedades, o elemento pode alterar de posição na tela sem que os testes precisem ser refeitos. Contudo, mudanças nas propriedades do elemento, por exemplo, o rótulo, também irá gerar necessidade de reescrita dos testes que utilizam tais propriedades. Por fim, as ferramentas de **terceira geração** utilizam tec-

nologias de reconhecimento de imagem para acessar os componentes gráficos a serem testados diretamente na tela. Apesar de melhorar alguns aspectos das ferramentas de segunda geração, as ferramentas da terceira geração possuem algumas limitações relacionadas ao tamanho e resolução da imagem da tela do aplicativo a ser testado.

O *Jacareto*[1] é um exemplo de ferramentas que pertence a primeira geração, baseada em coordenadas. Como foi possível observar no capítulo 4, as ferramentas que pertencem a essa geração tem caído cada vez mais em desuso, devido à sua falta de robustez a adaptabilidade.

Um exemplo de ferramenta de teste de segunda geração muito popular no mundo do desenvolvimento para Android é o *Espresso*, um *framework* de testes automatizados para Android, criado e mantido pelo *Google*. A ferramenta já vem instalada no *Software Developer Kit* do sistema Android, facilitando seu uso e propagação na comunidade. Os *scripts* da ferramenta *Espresso* são escritos em *Java* ou *Kotlin*, a linguagem de programação nativa para Android e possui três conceitos básicos: *ViewMatcher*, *ViewAction* e *ViewAssertion*. O *ViewMatcher* busca o elemento que se deseja testar dentro da tela em teste, enquanto o *ViewAction* realiza a ação naquele elemento e o *ViewAssertion*, por fim, faz a validação da informação retornada se está dentro do esperado.

A Figura 2.1 apresenta um caso de teste escrito em *Java*, utilizando o *framework* do *Espresso*, que testa o estado inicial de uma tela de *login*, verificando se a imagem e os campos de *login* e senha estão visíveis na tela. Nessa figura, é possível perceber a ação do *ViewMatcher*, a partir da função *onView*. Ele encontrará o objeto que combina com o parâmetro passado e retornará um objeto *ViewAction*, isto é, o elemento que será interagido. Já o *check* recebe um *ViewAssertion*, que validará a asserção passada como parâmetro, a função *matches* recebe um *ViewMatchers* e retornará um *viewAssertion* e o *isDisplayed* é o *ViewMatcher* utilizado para validar que a informação está visível na tela, retornando um *viewAssertion* para a função *check*, indicando se o elemento está visível ou não na tela, permitindo validar sua condição inicial. (COLANGELO, 2016).

Fonte: (COLANGELO, 2016)

[1] https://www.onworks.net/pt/software/app-jacareto

Figura 2.1 – Exemplo de *script* de teste, em Java, da ferramenta “Espresso”

```
@Test
public void shouldDisplayInitialState() {
  onView(withId(R.id.login_image)).check(matches(isDisplayed()));
  onView(withId(R.id.login_username)).check(matches(isDisplayed()));
  onView(withId(R.id.login_password)).check(matches(isDisplayed()));
  onView(withId(R.id.login_button)).check(matches(isDisplayed()));
}
```

Outras ferramentas existentes são o *Appium*[2] e *Ui Automator*[3], da segunda geração, e *SikuliX*[4] e *EyeAutomate*[5] da terceira geração.

2.2 Trabalhos relacionados

Esta seção discute alguns trabalhos relacionados à automação de testes de GUI para aplicativos móveis, apontando sua principal contribuição e as diferenças destes trabalhos para o presente trabalho.

Kong et al. (2019) traz uma Revisão Sistemática da Literatura sobre testes automatizados para dispositivos móveis, apresentando as pesquisas feitas sobre o assunto e quais eram os tipos de testes que essas pesquisas focavam. Assim sendo, os autores propuseram uma taxonomia de testes para aplicativos *Android*, contemplando os objetivos do teste, os alvos, os níveis e as técnicas. No trabalho de Kong et al. (2019), o foco principal estava na literatura formal, apresentando como a área tem se desenvolvido academicamente. No presente trabalho, o objetivo é apresentar um recorte específico de uma área da automação de testes para dispositivos móveis, os testes de *GUI*, além de buscar referências na “literatura cinza”, a fim de se conhecer o que tem sido alvo de atenção também na indústria de software.

Min e Cai (2018) realizaram uma comparação entre algumas ferramentas de testes automatizados de diferentes gerações. Em sua pesquisa, eles apresentaram testes de desempenho, tanto para a escrita de casos de teste, quanto para a execução dos mesmos. Quatro ferramentas foram testadas ao todo, sendo duas da segunda geração (*Appium*, *UI Automator*) e duas da

[2] https://appium.io/
[3] https://developer.android.com/training/testing/ui-automator?hl=pt-br
[4] http://sikulix.com/
[5] https://eyeautomate.com/

terceira geração (*EyeAutomate* e *SikuliX*)[6]. No presente trabalho, o foco é realizar uma busca mais ampla de ferramentas de teste automatizado, visando elaborar um catálogo das ferramentas existentes, contendo suas principais características, pontos fortes e pontos fracos.

Lämsä (2017) também realiza uma comparação entre ferramentas de testes automatizados para dispositivos móveis e lista várias ferramentas existentes na época em que a pesquisa foi realizada. O autor faz uma busca além da literatura formal, com intuito de ter uma listagem mais completa. Como diferencial, o trabalho proposto nesta monografia traz uma listagem atualizada de informações sobre ferramentas de teste automatizado para dispositivos móveis Android. Além disso, pontos fracos que existiam à época em que o trabalho de Lämsä (2017) foi conduzido podem não existir mais. Por fim, o presente trabalho procura fazer uma análise da popularidade das ferramentas de teste automatizado, o que não foi feito no trabalho de Lämsä (2017).

[6] Não foram encontradas pesquisas que citem a utilização dessa ferramenta durante a execução do MSL, por isso, ela não é mencionada no Capítulo 4.

3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Mapeamento Sistemático da Literatura (MSL) é um método de estudo inspirado na pesquisa médica (PETERSEN et al., 2008), que busca realizar uma revisão ampla da literatura, a partir de uma análise sistemática de estudos existentes. Trata-se de uma revisão da literatura planejada e bem documentada e possui as etapas de *planejamento*, *execução* e *resultados*. Por possuir uma metodologia bem definida e documentada, este método possibilita a validação dos resultados por meio da reprodutibilidade do processo, tornando-o mais confiável (SILVA, 2019).

3.1 Planejamento

No planejamento de um MSL, deve-se definir três pontos principais, a saber: *questões de pesquisa*, *método de busca* e *critérios de inclusão/exclusão*. Essa etapa busca definir o porquê da pesquisa e como se dará todo o processo de condução da mesma.

3.1.1 Questões de pesquisa

É a definição do "porquê" da pesquisa, indicando quais são as motivações para a existência da revisão da literatura, bem como direcionando sua execução. As questões de pesquisa elencadas neste trabalho, com suas respectivas justificativas, são apresentadas a seguir:

Q1. Quais são as ferramentas existentes na literatura para automação de testes de GUI para aplicativos Android e quais são as suas principais características (nome, geração em que ela se encaixa, se é paga ou gratuita, entre outros)? A existência de um catálogo sobre as ferramentas existentes, elencando suas características, pontos fortes e pontos fracos, é importante para que os leitores que pretendem entrar na área de qualidade de aplicativos para Android possam ter as principais informações reunidas em um só lugar, facilitando a busca e agilizando processos.

Q2. Qual é a metodologia/*linguagens* utilizado para elaboração de casos de testes nestas ferramentas? Busca-se entender quais são os conceitos utilizados para a criação dos casos de testes, por exemplo, se utiliza uma linguagem de programação própria ou de propósito geral e quais são elas. Saber essa informação é útil, pois as pessoas interessadas terão uma visão precisa sobre os conhecimentos necessários para utilizar cada ferramenta, podendo as-

sim, entender e direcionar seus estudos para adquirir os conhecimentos daquela que melhor lhe atende.

Q3. Qual a "popularidade" das ferramentas identificadas? Descobrir o que está "em alta" no mercado pode ajudar possíveis leitores deste trabalho a identificarem as ferramentas que possuem uma comunidade mais ativa e/ou uma quantidade maior de materiais de apoio, facilitando assim sua tomada de decisão.

Q4. Quais são as principais vantagens e desvantagens destas ferramentas? A justificativa para existência desta questão é que ela pode trazer uma visão mais realista sobre o uso da ferramenta, a partir do ponto de vista de seus usuários, o que muitas vezes não é encontrado em cursos sobre essas ferramentas, bem como na própria documentação das mesmas. Novamente, isso pode facilitar a tomada de decisão por parte de pessoas interessadas em ferramentas para teste de GUI para aplicações Android.

3.1.2 Método de busca

Tradicionalmente, as buscas de um MSL costumam ser realizadas apenas na denominada "literatura formal", que contempla os estudos publicados em eventos e periódicos científicos (SILVA, 2019). Contudo, isso restringe a abrangência dos resultados, pois são excluídos estudos divulgados fora dos veículos de publicação acadêmicos (GAROUSI; FELDERER; MäNTYLä, 2019), tais como em *websites*, *blogs*, fóruns, entre outros. Esses canais de divulgação fazem parte de um tipo de literatura conhecida como "literatura cinza". Uma vez que o tópico de pesquisa desta monografia possui um enfoque bastante prático e mercadológico, com o intuito de se obter melhores resultados, ambos os tipos de literatura foram consultados neste trabalho.

Para a literatura cinza, a fonte de busca utilizada foi o *Google*[1], conforme recomendado por Garousi, Felderer e Mäntylä (2019). Para a literatura formal, a busca foi feita usando o *Google Scholar*[2], que mescla resultados de várias fontes de estudos científicos, tais como *IEEE Xplore*, *ACM Digital Library*, *Scopus*, entre outras.

Uma vez que esses tais mecanismos de busca podem retornar milhares de resultados, ficaria inviável analisá-los em sua totalidade. Assim, foi utilizada uma heurística de parada recomendada por Garousi, Felderer e Mäntylä (2019), que faz uso do algoritmo de *PageRank* do *Google* para indicar o momento que a busca deve ser interrompida.

[1] https://www.google.com/
[2] scholar.google.com.br/

Essa heurística apresenta três possibilidades:

- Quando as pesquisas não encontram mais artigos relevantes.
- Quando apenas as *top* "N" páginas da busca são visitadas, isto é, escolhe-se uma quantidade de páginas para visitar, por exemplo, as 10 primeiras.
- Quando todas as evidências já foram extraídas.

Para este trabalho, foi utilizada a primeira possibilidade, isto é, foram visitados todos os artigos de cada página e sua relevância para a pesquisa foi verificada (ver Seção 3.1.4) até que houvesse uma página na qual nenhum dos artigos retornados fosse relevante para a pesquisa. Então, a consulta poderia ser interrompida, segundo essa heurística.

3.1.3 *Strings* de busca

As *strings* de busca são os termos utilizados nos mecanismos de buscas para encontrar os artigos que contenham informações relevantes para a pesquisa. Foram desenvolvidas *strings* iniciais, as quais foram avaliadas em uma busca preliminar e revisada pelo orientador da pesquisa. Por fim, chegou-se às seguintes *strings* com foco em ferramentas de testes automatizados de interface de usuário para aplicações Android:

- Inglês: *((tools OR "automated testing" OR "automation testing") AND ("GUI testing" OR "GUI tests") AND "Android"))*
- Português: *((ferramentas OR "testes automatizados") AND ("GUI" OR "interface gráfica de usuário" OR "interface gráfica com usuário") AND "Android"))*

3.1.4 Critérios de seleção

Os critérios de seleção de um MSL são definidos anteriormente à execução. Eles buscam definir as características que um estudo deve apresentar para que ele seja aceito ou não para compor o portfólio dos estudos a serem analisados.

Para a literatura formal foram definidos três critérios de seleção: (a) O texto completo precisa estar disponível para acesso e estar escrito em português ou inglês; (b) O estudo deve apresentar ao menos uma ferramenta de automação de testes para aplicativos Android; e (c) O estudo deve ser focado em testes de GUI. Para um estudo ser aceito, ele deveria satisfazer a todos os critérios acima.

Para a literatura cinza, foram utilizados os critérios propostos nos trabalhos de Garousi, Felderer e Mäntylä (2017), que propõe o uso de condições que atendam a requisitos específicos da literatura cinza (Tabela 3.1). Para que um estudo divulgado na literatura cinza fosse aceito, ele deveria atender a todos os critérios apresentados na Tabela 3.1.

Figura 3.1 – Critérios de seleção de estudos da literatura cinza

Aspecto	Critérios
Autoridade	• É um autor individual associado a uma organização respeitável? • O autor publicou outros trabalhos no campo?
Precisão	• A fonte tem um objetivo claramente declarado? • A fonte tem uma metodologia declarada? • A fonte é apoiada por referências autorizadas e documentadas?
Cobertura	• Existem limites claramente definidos? • O trabalho cobre uma questão específica? • O trabalho se refere a uma população em particular?
Objetividade	• O trabalho parece ser equilibrado na apresentação? • A declaração é uma opinião subjetiva? • As conclusões são tendenciosas?
Data	• Verifique a bibliografia: o material contemporâneo chave foi incluído? • Se nenhuma data é dada, mas pode ser verificada de perto, existe uma razão válida para a sua ausência? • O item tem uma data claramente declarada relacionada ao conteúdo?
Significância	• Enriquece ou acrescenta algo único à pesquisa? • Fortalece ou refuta a posição atual?

Fonte: (GAROUSI; FELDERER; MäNTYLä, 2017) apud (SILVA, 2019)

3.2 Execução do MSL

A execução do Mapeamento Sistemático da Literatura é a etapa na qual, de fato, será realizada a busca pelos artigos e serão aplicados os critérios de seleção de estudos. As buscas ocorreram no período de 20 de novembro a 13 de dezembro de 2022, tomando um total de 23 dias para o processo de busca e leitura dos artigos. Não houve limitação de data, sendo todos os artigos considerados relevantes inseridos no trabalho, independente de sua data de publicação.

3.2.1 Literatura formal

A busca na literatura formal consistiu na pesquisa pelas *strings* de busca no *Google Scholar*. Inicialmente, realizou-se a busca pela *string* em português, que retornou um total de 1750 resultados. A partir da leitura dos títulos e dos resumos dos artigos retornados, foi encontrado que apenas um artigo era relevante para o trabalho em questão. Os demais foram descartados, pois já no resumo ou no título deixavam claro não se tratar de testes para aplicativos Android, ou não focar em GUI. A busca foi interrompida na página 2, com base na heurística especificada na seção anterior.

Posteriormente, foi realizada a busca da *string* em inglês, retornando um total de 2210 resultados. O mesmo procedimento de leitura do título e do resumo foi conduzido. Nesse caso, foram selecionados um total de 39 artigos. A busca foi interrompida na página 5 da plataforma Google Scholar.

Em suma, 40 artigos da literatura formal foram encontrados como potenciais trabalhos para responder às questões definidas na Seção 3.1.1. Tendo os 40 artigos em mãos, foi realizada a leitura da introdução dos artigos selecionados, aplicando novamente os critérios de seleção (ver Seção 3.1.4). Durante essa fase, foi possível descartar cinco artigos dentre os selecionados. Em um destes artigos, ficava claro que o objetivo do trabalho era apresentar uma taxonomia das fragilidades de testes de GUI para Android, não trabalhando com nenhuma ferramenta. Havia também um estudo que propunha um MSL sobre quais temas têm sido abordados pela academia a respeito de testes de GUI. O terceiro artigo rejeitado apresentava uma abordagem para geração de testes automatizados, contudo o foco não era Android. Dois artigos foram excluídos por apresentarem técnicas para melhorar testes manuais de GUI, ou seja, não tinham enfoque em testes automatizados.

Os trinta e cinco artigos restantes foram lidos em sua totalidade e, novamente, os critérios de seleção foram aplicados. Após isso, vinte e três artigos foram rejeitados por não passarem nos critérios de seleção. Dois artigos selecionados não puderam ser lidos em sua totalidade, por não estarem disponíveis para download. Dois eram versões resumidas de outro artigo já selecionado. O restante dos artigos foi excluído por não apresentarem ferramentas de testes ou porque não foi possível encontrar uma página ativa para download da ferramenta proposta. Por fim, conforme pode ser visto na Tabela 3.1, 12 (doze) artigos foram aceitos por

se adequarem aos critérios de seleção propostos neste MSL. A listagem final dos artigos aceitos neste MSL, no que tange à literatura formal, se encontra na Tabela 3.2.

Tabela 3.1 – Quantidade de artigos aceitos e rejeitados por etapa na literatura formal.

Etapa	Aceitos	Rejeitados
Leitura do título e resumo dos artigos	40	3920
Leitura da introdução	35	5
Leitura completa dos artigos	12	23

Fonte: Do autor

Tabela 3.2 – Listagem de artigos selecionados a partir da literatura formal.

Referência	Título
(SU et al., 2017)	Guided, Stochastic Model-Based GUI Testing of Android Apps
(ARDITO et al., 2019)	Espresso vs. EyeAutomate: An Experiment for the Comparison of Two Generations of Android GUI Testing
(HU; NEAMTIU, 2011)	Automating GUI testing for Android applications
(LäMSä, 2017)	Comparison of GUI testing tools for Android applications
(MEILIANA et al., 2018)	Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study
(WEN et al., 2015)	PATS: A Parallel GUI Testing Framework for Android Applications
(COPPOLA; MORISIO; TORCHIANO, 2018)	Maintenance of Android Widget-Based GUI Testing: A Taxonomy of Test Case Modification Causes
(AMALFITANO et al., 2012)	Using GUI ripping for automated testing of Android applications
(LV et al., 2023)	Fastbot2: Reusable Automated Model-based GUI Testing for Android Enhanced by Reinforcement Learning
(SILVA, 2014)	Teste de software para aplicativos Android: um mapeamento sistemático da literatura
(ESBJÖRNSSON, 2015)	Android GUI Testing: A comparative study of open source Android GUI testing frameworks
(KROPP; MORALES, 2010)	Automated GUI Testing on the Android Platform

Fonte: Do autor

3.2.2 Literatura cinza

Para literatura cinza, conforme definido anteriormente na Seção 3.1.2, utilizou-se o mecanismo de busca *Google*. Ao pesquisar inicialmente pela *string* em português, foram retornadas duas páginas apenas, contendo 18 (dezoito) artigos, ao todo. Dos artigos retornados, realizou-se a leitura dos mesmos, aplicando os critérios de seleção propostos na Tabela 3.1. To-

dos os artigos foram rejeitados pela falta de data, metodologia ou por não ser possível verificar a autoridade do autor do artigo. Na pesquisa pela *string* em inglês, foram retornadas aproximadamente dezesseis páginas, porém, apenas na primeira página foram apresentados resultados interessantes para este MSL. Ao todo 10 artigos foram lidos, dos quais apenas 2 foram aceitos. Os demais foram excluídos devido à falta de autoridade do autor ou por não ser possível verificar a data da criação do artigo, entre outras.

A execução da busca pela literatura cinza, de ambas as *strings*, consistiu de um processo de leitura e avaliação crítica das páginas encontradas. Inicialmente, foi realizada a leitura rápida das páginas, em busca das referências das informações salientadas nos critérios de seleção, tais como data e autoridade. Posteriormente, foi realizada a leitura completa da página, realizando uma análise crítica e buscando extrair informações a respeito de sua precisão, cobertura, objetividade e significância. Embora muitas páginas falhavam em possuir autoridade confirmada do autor sobre o assunto e data, vários outros foram rejeitados pela falta de objetividade, apresentando análises enviesadas.

Dessa forma, ao finalizar a pesquisa na literatura cinza, houve o acréscimo de 2 (dois) novos artigos, os quais são apresentados na Tabela 3.3, juntamente com sua referência.

Tabela 3.3 – Lista de artigos selecionados a partir da literatura cinza.

Referência	**Título**
(HELP, 2023)	30 Best GUI Testing Tools For GUI Test Automation [2023 LIST]
(ASHIQ, 2022)	11 Best Automated UI Testing Tools In 2022

Fonte: Do autor

4 RESULTADOS E DISCUSSÃO

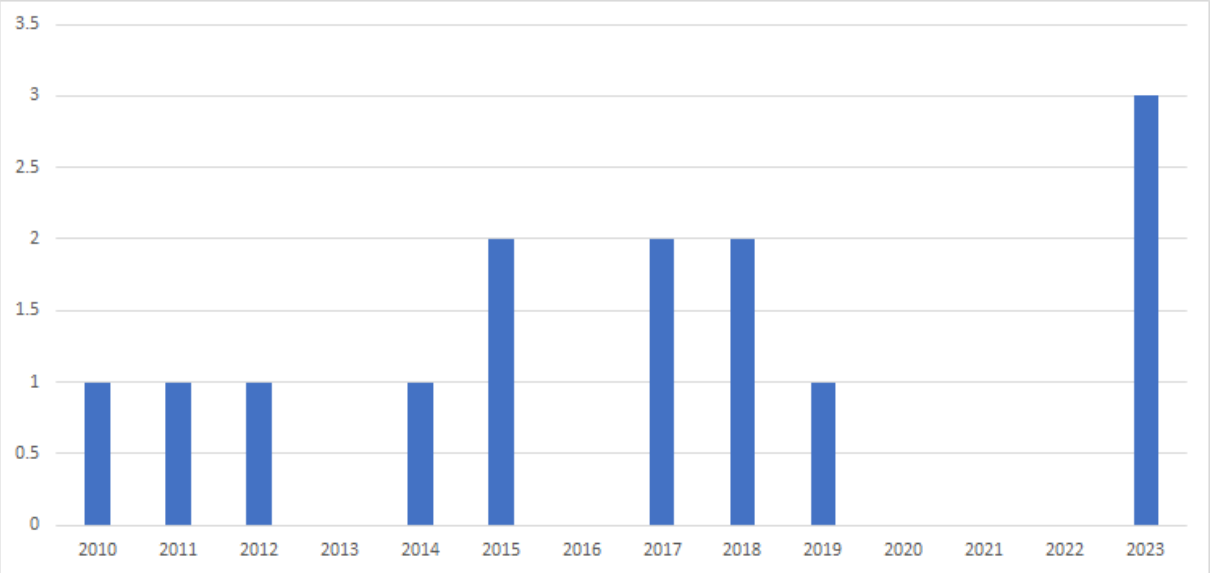
Neste capítulo, são apresentados os resultados obtidos após a leitura completa dos artigos selecionados. Por motivos de simplicidade e organização, os resultados são apresentados em forma de respostas às questões de pesquisa.

4.1 Questão de Pesquisa Q1

Na primeira questão de pesquisa (Q1), buscou-se responder quais são as ferramentas existentes para automação de testes de GUI em aplicativos *Android* e quais as principais características dessas ferramentas, a saber, seu nome, geração em que ela se encaixa, se é paga ou gratuita, entre outros.

Como não houve limitação de data de publicação dos artigos, foram retornados e inseridos artigos das mais diversas datas (Figura 4.3), sendo o mais antigo de 2010 e o mais recente de 2023, dando um período de 13 anos de publicações, um recorte de tempo bem interessante e deixa evidente que em 2023 já há um número considerável de pesquisas em cima do tema.

Figura 4.1 – Quantidade de artigos por ano.



Fonte: Do autor

A partir da análise das literaturas formal e cinza, 21 (vinte e uma) ferramentas foram encontradas, as quais são apresentadas na Tabela 4.1, com os artigos científicos ou páginas que as referenciam.

Tabela 4.1 – Ferramentas para automação de testes de GUI identificadas

Ferramenta	Artigos	Origem
Appium	(LäMSä, 2017) e (MEILIANA et al., 2018)	Literatura formal
Android Instrumentation Framewrok	(KROPP; MORALES, 2010)	Literatura formal
AndroidRipper	(AMALFITANO et al., 2012)	Literatura formal
Calabash	(LäMSä, 2017) e (MEILIANA et al., 2018)	Literatura formal
Espresso	(ARDITO et al., 2019), (LäMSä, 2017), (MEILIANA et al., 2018), (COPPOLA; MORISIO; TORCHIANO, 2018), (ESBJÖRNSSON, 2015) e (HELP, 2023)	Literatura formal e cinza
EyeAutomate	(ARDITO et al., 2019)	Literatura formal
FastBot2	(LV et al., 2023)	Literatura formal
Katalon Studio	(HELP, 2023)	Literatura cinza
LambdaTest	(ASHIQ, 2022)	Literatura cinza
Monkey	(HU; NEAMTIU, 2011)	Literatura formal
MonkeyRunner	(LäMSä, 2017)	Literatura formal
PATS	(WEN et al., 2015)	Literatura formal
Positron Framework	(KROPP; MORALES, 2010)	Literatura formal
Ranorex	(LäMSä, 2017) e (HELP, 2023)	Literatura formal e cinza
Robotium	(LäMSä, 2017), (ESBJÖRNSSON, 2015)	Literatura formal
Selendroid	(LäMSä, 2017)	Literatura formal
SPAG	(SILVA, 2014)	Literatura formal
STOAT	(SU et al., 2017)	Literatura formal
TEMA Tools	(SILVA, 2014)	Literatura formal
Tau	(LäMSä, 2017)	Literatura formal
Ui Automator	(ESBJÖRNSSON, 2015), (MEILIANA et al., 2018), (LäMSä, 2017)	Literatura formal

Fonte: Do autor

Algumas das ferramentas retornadas eram, na verdade, serviços em nuvem que rodam casos de testes em dispositivos reais e apresentam resultados detalhados da execução. Por exemplo, o *Katalon Studio* é um serviço de execução de testes em nuvem que permite que casos de testes sejam criados utilizando a ferramenta *Appium*. Da mesma forma, o *LambdaTest* provê um serviço em nuvem para testes multiplataformas de aplicações, incluindo aplicações *Android*. Este serviço suporta casos de testes escritos em várias ferramentas diferentes, tais como *Appium* e *Espresso*.

Outras ferramentas, tais como *FastBot2*, *Stoat*, *Monkey* e *AndroidRipper*, tinham como objetivo a geração de casos de testes de forma automatizada e não a execução desses casos de testes, propriamente ditos. Por fim, não foi possível encontrar informações sobre algumas ferramentas, tais como *Android Instrumentation Framework*, *Positron Framework*, *SPAG*, *TEMA*

Tools e *Tau*. Possivelmente, em algum momento essas ferramentas perderam a relevância e foram descontinuadas.

Das 21 ferramentas encontradas, as que possuíam informações disponíveis na Internet e permitiam ser utilizadas para criação de casos de teste e execução automática dos mesmos estão listadas na Tabela 4.2, com suas principais características.

Tabela 4.2 – Características das ferramentas analisadas

Ferramenta	**Geração**	**Licença/Modelo de negócio**	**Última Atualização**
Appium	2ª	Open-source	15/05/2022
Calabash	2ª	Open-source	20/01/2023
Espresso	2ª	Open-source	03/01/2023
EyeAutomate	3ª	Proprietário e gratuito	Não informado
MonkeyRunner	2ª	Open-source	28/12/2020
Ranorex	3ª	Proprietário e pago	Não informado
Robotium	2ª	Open-source	27/09/2016
Selendroid	2ª	Open-source	21/10/2015
Ui Automator	2ª	Open-source	11/01/2023

Fonte: Do autor

Foi percebido que as ferramentas retornadas na literatura formal eram, em sua maioria, ferramentas *open-source* e, mesmo as que não eram de código-aberto, eram ferramentas de uso gratuito. Isso ocorre devido ao foco que os artigos apresentados possuem. Como boa parte deles tentam apresentar novas metodologias de trabalho, é natural que as ferramentas sejam *open-source* para poderem ser estudadas e melhoradas.

Oito das ferramentas encontradas pertencem à segunda geração de ferramentas de testes de GUI e são *open-source*. Essa geração é conhecida como baseada em *layout* e utiliza de propriedades únicas dos elementos da interface gráfica para realizar a interação com o aplicativo em teste. Apenas duas ferramenta, *EyeAutomate*[1] e *Ranorex*[2] se enquadraram na terceira geração, baseada em reconhecimento de imagens; Ambas as ferramentas são proprietárias. A *Ranorex* é paga para uso e a *EyeAutomate* é gratuita para *download* e execução de comandos básicos. Nenhuma ferramenta pertencia à primeira geração, baseada em coordenadas. Ferramentas desse tipo são menos comuns, devido à falta de adaptabilidade e robustez desse tipo de abordagem (ARDITO et al., 2019). Ferramentas de segunda geração são mais comuns do

[1] https://eyeautomate.com/
[2] https://www.ranorex.com/

que as de terceira, apesar de que Ardito et al. (2019) indica em seu estudo uma predileção em um grupo de testes selecionados pelo *EyeAutomate*, principalmente quando o testador não é o desenvolvedor, a presença de mais ferramentas da segunda geração, evidencia uma falta de pesquisas focada nos usos das de terceira geração.

As ferramentas *MonkeyRunner*[3], *Robotium*[4] e *Selendroid*[5] aparentam estar obsoletas, visto que as versões disponíveis nos sites oficiais das plataformas estão há algum tempo sem atualização. A ferramenta *Appium*[6] tem a última versão oficial lançada em maio de 2022, contudo, o repositório do *GitHub* da ferramenta possui atualizações mais recentes (Jan/2023), que ainda não foram compiladas em uma versão oficial.

4.2 Questão de Pesquisa Q2

Quanto à segunda questão de pesquisa, o objetivo foi apontar como são criados os casos de teste nas ferramentas analisadas, isto é, se elas utilizam alguma linguagem de programação própria ou de propósito geral e quais são elas. De acordo com Ardito et al. (2019), existem várias abordagens para geração de testes de GUI, as quais podem ser classificadas em *aleatórias*, *baseadas em modelos* e *manuais*.

Nas ferramentas que trabalham com a abordagem aleatória, também chamada de randômica, são geradas entradas independentes e aleatórias e os resultados das saídas são comparados com as especificações do software (KONG et al., 2019). *AndroidRipper* e *Monkey* são ferramentas que utilizam dessa metodologia para gerar casos de teste. Quanto à abordagem baseada em modelos, os casos de teste são gerados automaticamente, com base em um modelo que descreve a funcionalidade do aplicativo em teste ((KONG et al., 2019)), por exemplo, uma Máquina de Estado Finita. As ferramentas *FastBot2* e *Stoat* foram desenvolvidas com base nesta abordagem.

As ferramentas que utilizam a escrita manual de códigos de casos de teste são a maioria. Isso se deve ao fato de ser uma metodologia mais tradicional, já amplamente utilizada para testes de GUI em plataformas web, por exemplo. A Tabela 4.3 apresenta as ferramentas deste

[3] https://developer.android.com/studio/test/monkeyrunner?hl=pt-br
[4] https://github.com/RobotiumTech/robotium
[5] http://selendroid.io/
[6] https://github.com/appium/appium-android-driver

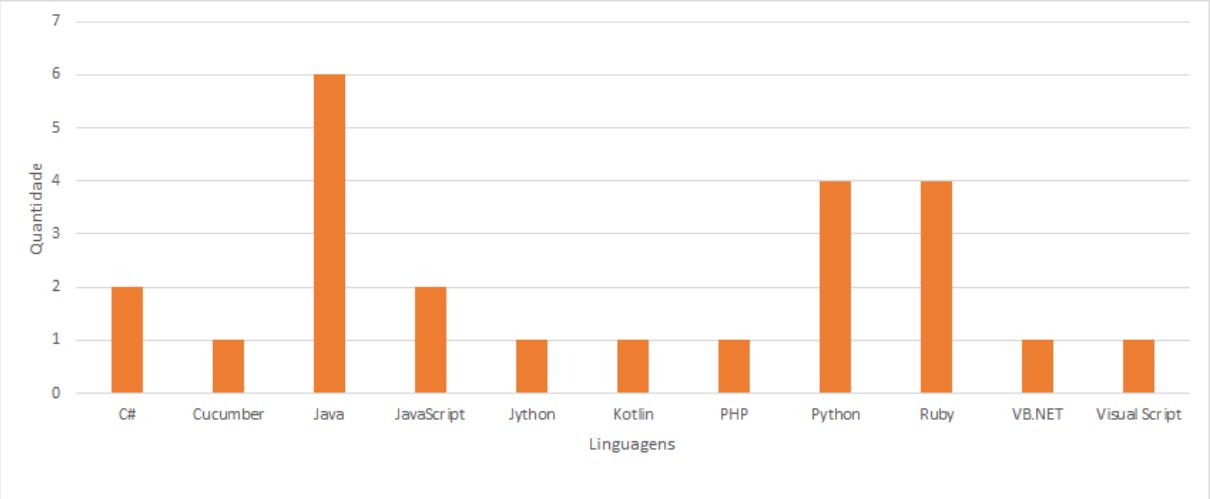
tipo, bem como as linguagens utilizadas por elas para a construção de casos de teste. O gráfico da Figura 4.2 apresenta a quantidade de utilização de cada linguagem.

Tabela 4.3 – Ferramentas que utilizam linguagens de programação para criação de casos de teste.

Ferramenta	Linguagem
Appium	C#, Java, JavaScript, PHP, Python e Ruby
Calabash	Cucumber e Ruby
Espresso	Java e Kotlin
EyeAutomate	Visual Script e Python
MonkeyRunner	Jython
Ranorex	C#, Java, Python, Ruby, JavaScript, VB.NET
Robotium	Java
Selendroid	Java, Python e Ruby
Ui Automator	Java

Fonte: Do autor

Figura 4.2 – Linguagens utilizadas nas ferramentas



Fonte: Do autor.

Analisando o gráfico da Figura 4.2, é possível perceber que *Java* e *Python* são as linguagens mais utilizadas para a escrita de casos de teste. *Java*, com 6 aparições, é a mais utilizada por ser multiplataforma e, dessa forma, permitir a reutilização de código em outros sistemas operacionais. A utilização de *Python* se dá, provavelmente, por ser uma linguagem simples de usar, com uma boa escalabilidade, demandando menos tempo de aprendizagem dos desenvolvedores. *Ruby* é outra linguagem bastante usada e se destaca, também, por ser simples de usar e de fácil leitura, facilitando o entendimento dos códigos de teste.

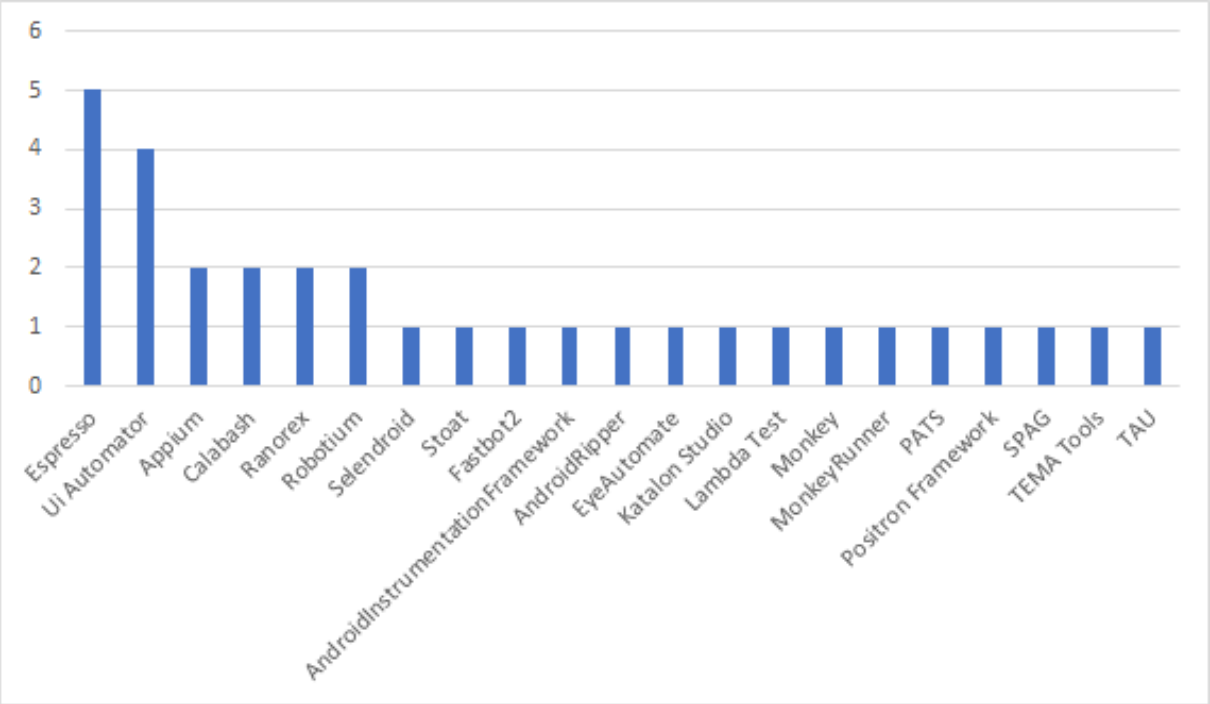
Outro aspecto interessante é a existência de uma implementação do *Python* que gera compilações para máquinas *Java*, o *Jython*, com ele é possível fazer aplicações híbridas que

utilizam tanto código *Java* quanto *Python*. Essa implementação traz a facilidade de se rodar os código em Python, que são mais simples e fáceis de aprender, nas máquinas virtuais *Java*, as JVM *(Java Virtual Machine)*, que permite facilidade e robustez.

4.3 Questão de Pesquisa Q3

A questão de pesquisa 3 tem o objetivo de definir a "popularidade" das ferramentas identificadas. Isso foi feito, inicialmente, com base na quantidade de aparições das ferramentas nos artigos selecionados. Com isso, pode-se inferir quais são as ferramentas que tem recebido o foco dos pesquisadores e profissionais da indústria nos últimos anos.

Figura 4.3 – Quantidade de citações por ferramenta.



Fonte: Do autor

Observando a quantidade de aparições de cada ferramenta no gráfico da Figura 4.3, pode-se notar que o *Espresso* foi a ferramenta que teve mais citações, seguido pelo *Ui Automator*. O motivo dessa alta popularidade do *Espresso* e do *Ui Automator* pode ter a ver com o fato de serem ferramentas desenvolvidas e mantidas pelo *Google*, o mesmo criador e mantenedor do Android. Além disso, o Google disponibiliza as ferramentas gratuitamente com o SDK (*Software Development Kit*) do Android, simplificando o processo de instalação e utilização das mesmas.

Outras ferramentas que tiveram mais de uma citação foram *Appium*, *Calabash*, *Ranorex* e *Robotium*. Isso possivelmente ocorre, pois *Appium*, *Calabash* e *Ranorex* são ferramentas multiplataformas, podendo ser usadas para teste de GUI em aplicações para *Android* assim como aplicações para *iOS* e web. Assim, o uso destas ferramentas em pesquisas e até mesmo no mercado é incentivado nos casos em que se precisa de aplicações híbridas, que funcionam em mais de uma plataforma para evitar a necessidade de reescritas de testes em diferentes linguagens. O *Robotium*, por sua vez, apesar de ser uma ferramenta específica para *Android*, é uma das primeiras ferramentas de testes automatizados de GUI para este sistema operacional. Sendo assim, é natural ser uma ferramenta que tenha destaque nas pesquisas.

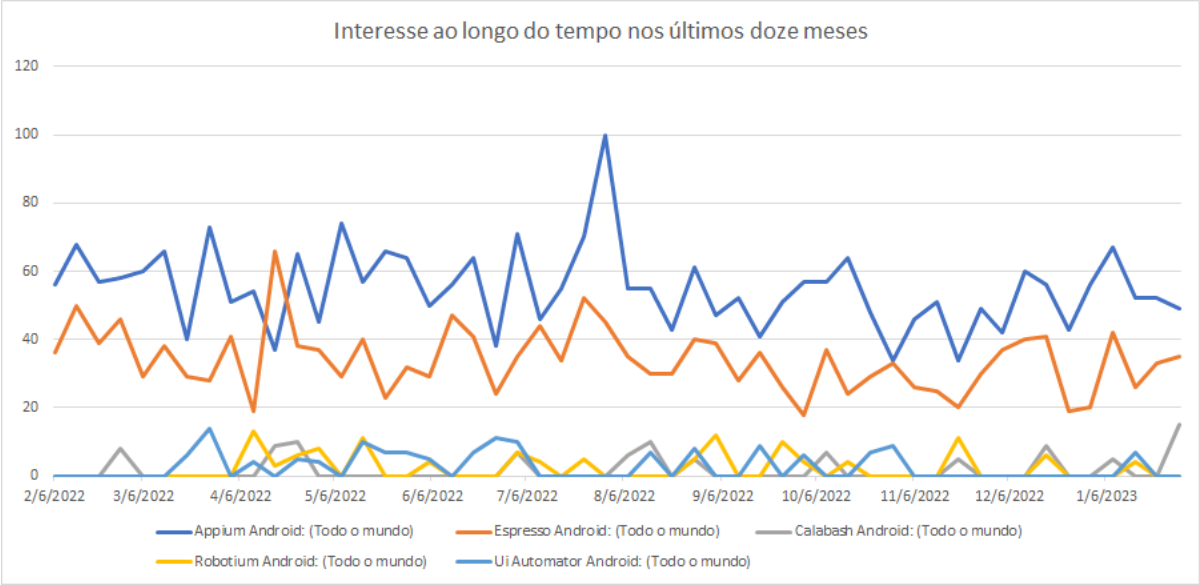
As demais ferramentas são citadas apenas uma vez. Nos casos da *FastBot2*, *PATS* e *Stoat*, estas são ferramentas experimentais/protótipos, que buscam exibir uma nova modelagem de testes de GUI. As outras são ferramentas de metodologia tradicional, mas que estão obsoletas.

É importante destacar que há uma limitação na metodologia utilizada para análise de popularidade das ferramentas: muitas vezes, as buscas na literatura formal retratam o que é pesquisado quanto ao tema abordado, contudo falham em apresentar um retrato do momento quando se fala de popularidade. Portanto, foi realizada uma busca no *Google Trends*[7] pelas ferramentas que possuem mais de uma citação.

Considerando que a pesquisa foi feita em Fevereiro de 2023, as ferramentas que mais apresentaram buscas, de acordo com os gráfico do *Google Trends* (Figura 4.4), nos últimos doze meses, foram *Appium* e *Espresso*, sendo o *Appium* mais popular do que o *Espresso*. Tal vantagem pode ser explicada pelo fato de a ferramenta *Appium* ser multiplataforma e permitir que os casos de testes sejam escritos em várias linguagens diferentes, tais como *Python* e *JavaScript*.

[7] Ferramenta que apresenta estatísticas sobre os termos mais pesquisados em um determinado período, permitindo também filtros por região. Disponível em: https://trends.google.com

Figura 4.4 – Ferramentas mais pesquisadas no *Google* nos últimos doze meses



Fonte: Google Trends

Appium e *Espresso* são, portanto, as ferramentas mais populares quando se fala em testes automatizados de GUI para *Android*, seguindo uma tendência levantada por Lämsä (2017).

As ferramentas de terceira geração, como o *EyeAutomate* não conseguiram ainda abarcar o mercado quando se fala de testes em *Android*. Quando se limita a busca por esse sistema operacional móvel, é visto que não há menções ao longo do último ano, evidenciando que as ferramentas de segunda geração mantêm-se populares no mundo da automatização de testes para *Android*. Isso ocorre por serem ferramentas de mais fácil acesso e que possuem mais conteúdos educacionais disponíveis, fazendo com que o novos entusiastas ou pesquisadores optem por elas.

4.4 Questão de Pesquisa Q4

Na quarta questão de pesquisa, foi realizado o levantamento sobre as principais vantagens e desvantagens destacadas pelos trabalhos selecionados, a respeito das ferramentas para automação de testes (Tabela 4.4).

Tabela 4.4 – Vantagens e desvantagens das ferramentas identificadas.

Ferramenta	Vantagens	Desvantagens
Appium	Documentação completa Compatível com versões antigas do *Android* Arquitetura cliente/servidor Multilinguagem Comunidade grande e ativa Oferece suporte a *WebView*	Testes são mais lentos Não suporta *Capture & Replay* Necessita de uma boa suíte de testes Problemas com "wait" nos métodos Problemas para lidar com listas, barras de navegações dinâmicas e permissões
Calabash	Boa documentação Linguagem de alto nível, fácil de ser aprendida Integração com nuvem de testes da Xamarim	Não testa o ciclo de vida completo da aplicação Não suporta *Capture & Replay*
Espresso	Fácil de aprendizado Suporta *Capture & Replay* Disponível no SDK do Android Compatível com versões antigas do *Android* Suporta emuladores e dispositivos reais Boa cobertura de testes Passível de ser mesclado com *Ui Automator* Testes são executados rapidamente Resultados confiáveis	Só testa dentro do AUT Não consegue tratar retomadas Difícil manutenção Requer muito código
EyeAutomate	Boa suítes de testes Abordagem intuitiva	Imprecisões da biblioteca de reconhecimento de imagem Suporte somente a emuladores
Robotium	Compatível com versões antigas do *Android* Testes são confiáveis Execução rápida	Não acessa todo o ciclo de vida do aplicativo Comunidade inativa Problemas para testar aplicações Só testa dentro do AUT
Selendroid	Compatível com versões antigas do *Android* Testa o ciclo de vida completo do aplicativo	API complexa Mal documentado Comunidade inativa
Ui Automator	Suporte completo ao ciclo de vida do app Boa documentação API simples Compatível com versões antigas do *Android*	Atualizações assíncronas da UI Problemas com *WebView*

Fonte: Do autor

Dadas as informações obtidas de vantagens e desvantagens, é possível concluir que o *Espresso* é a ferramenta com mais vantagens para utilização: possui uma API fácil de ser aprendida e bem documentada, além de permitir *Capture & Replay*, que consiste em gravar com a ferramentas as ações no AUT (*Application Under Testing*) e a própria ferramenta gerar os casos de teste e executá-los. A grande desvantagem do *Espresso* é o fato de não suportar eventos fora do sistema em teste, dessa forma, deixando de lado casos de teste no qual é necessário a

interação com elementos fora do aplicativo, como interrupções do sistema por chamadas, entre outros. Contudo, tal problema é passível de ser resolvido com o uso combinado ao *Ui Automator*, que permite romper a barreira do AUT. Apesar de requerer bastante código para criação dos casos de teste, a boa documentação e API simplificada são uma boa vantagem e seu tempo de execução mais rápido que as demais também auxiliam para o que o *Espresso* seja uma boa opção.

O *Appium*, apesar de sua popularidade, possui bastantes problemas, como a execução dos testes ser mais lenta e que, apesar de tratar as interrupções do sistema, o uso de "waits" nos métodos causam vários problemas de retomadas. O que o torna uma ferramenta passível de ser considerada para os testes em aplicações *Android* é sua boa documentação e a comunidade grande e ativa, que auxilia bastante na resolução de problemas. Sendo assim, apesar de ter muitos problemas, é uma opção considerável.

Já o *EyeAutomate*, que apresenta a metodologia *Visual-Based*, terceira geração, teve vários problemas para o reconhecimento de imagens, devido a imprecisões das bibliotecas que não conseguem lidar com elementos muito pequenas, ou grandes diferenças de cores. E em muitos casos, mesmo quando a captura da tela está dentro dos padrões, em alguns momentos, não é possível reconhecer alguns elementos, sendo necessário realizar novas capturas de tela maiores, elevando o tempo de desenvolvimentos dos casos de teste. Além de só ser possível de ser utilizado em emuladores para *Android*. É possível concluir que o *EyeAutomate* ainda tem vários pontos de melhoria para que seus testes em aplicações *Android* sejam mais completos e precisos.

Um problema geral das ferramentas de segunda geração é a dificuldade em encontrar propriedades não ambíguas para selecionar no momento da criação dos casos de teste, atrasando consideravelmente o processo de codificação. Essa é um problema geral da segunda geração, por ser *text-based* depende das propriedades inseridas pelos desenvolvedores no momento da criação dos elementos e, nem sempre, há uma preocupação em criar propriedades únicas para cada elemento, tornando seu acesso complicado.

As demais ferramentas não foram apresentadas na Tabela 4.4, pois não houve citações de vantagens e desvantagens quanto a sua utilização nos artigos analisados neste MSL.

5 AMEAÇAS A VALIDADE

As ameças a validade da monografia são os pontos que podem afetar os resultados obtidos por este trabalho. Algumas das ameaças são listadas a seguir.

A calibragem das *strings* de busca pode não ter sido tão efetiva e vários trabalhos com resultados interessantes para responder às questões de pesquisa podem não ter sido inclusos nesse trabalho. Para mitigar esse problema, a calibragem das *strings* foram realizadas com o professor orientador de forma ser realizada várias buscas com várias *strings* diferentes até obtermos *strings* que retornou o maior número de resultados possíveis. Ainda, sim, alguns artigos podem não ter sido incluídos.

A leitura e aplicação dos critérios de seleção poderia ser enviesada, dado que somente uma pessoa teria feito todo o processo de seleção, contudo, para mitigar esse problema houve uma revisão por pares dos artigos, tendo sido feito uma avaliação sobre os resultados obtidos e ajustados conforme a necessidade no momento de revisão.

6 CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi identificar e catalogar as ferramentas existentes para automação de testes de GUI em aplicações para *Android*. Dessa forma, foi realizado um Mapeamento Sistemático da Literatura (MSL) com referências tanto acadêmicas quanto informais, buscando evidências sobre as ferramentas existentes, abordando sua popularidade e as principais vantagens e desvantagens encontradas durante sua utilização.

Com este mapeamento, foi percebido que o tema tem sido alvo de inúmeras pesquisas acadêmicas, principalmente aquelas que visam apresentar novas metodologias para criação de casos de testes automáticos. Na pesquisa informal, porém, há um problema de qualidade nos artigos, sendo grande parte dos resultados retornado pela pesquisa no *Google* descartada por não possuírem referências ou nem terem seus autores apontados. Vale destacar que a heurística utilizada para interromper as buscas, tanto no *Google Scholar* quanto no *Google*, apesar de ser consolidada e utilizada em outros artigos científicos, pode ter descartado artigos que seriam relevantes às nossas questões de pesquisa, sendo então descartados e não abordados neste trabalho.

Dentre as ferramentas retornadas, foi percebido que o *Espresso* e o *Appium* são as ferramentas mais populares para automatização de testes de GUI em aplicativos *Android* e os motivos são os mais variados, como utilização mais simplificada e boa documentação. Ambas são ferramentas gratuitas e *open-source*, sendo que o *Espresso* possui o *Google* como mantenedor, disponibilizada junto ao SDK do *Android Studio*.

Para trabalhos futuros, sugere-se que seja feita uma análise comparativa prática entre o *Espresso* e o *Appium*, utilizando-os em projetos reais da indústria e com testadores e desenvolvedores que estejam no mesmo nível de experiência de uso em ambas as ferramentas, a fim de avaliar:

- Tempo de aprendizado de uso da ferramenta;
- Tempo de escrita de caso de teste;
- Tempo de execução das ferramentas;
- Resultados obtidos, comparando a quantidade de erros encontrados pelas ferramentas e falsos positivos, além das dificuldades de uso.

Além disso, outro trabalho interessante seria criar um processo de implementação de testes automatizados em empresas desenvolvedoras de software para *Android* utilizando uma das ferramentas e destacar todos os desafios encontrados nessa implementação de nova metodologia de trabalho.

Há também a possibilidade de atualizar este trabalho com uma revisão sistemática da literatura, buscando utilizar as ferramentas, fazendo experimentos para comparar as evidências das mesmas e comparar os resultados obtidos com o que foi obtido da literatura já existente.

REFERÊNCIAS

AHO, P. et al. Automated java gui modeling for model-based testing purposes. In: **2011 Eighth International Conference on Information Technology: New Generations**. [S.l.: s.n.], 2011. p. 268–273.

AMALFITANO, D. et al. Using gui ripping for automated testing of android applications. In: **2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering**. [S.l.: s.n.], 2012. p. 258–261.

APPBRAIN. **Number of Android apps on Google Play**. Worldwide, 2022. Disponível em: <https://www.appbrain.com/stats/number-of-android-apps>.

ARDITO, L. et al. Espresso vs. eyeautomate: An experiment for the comparison of two generations of android gui testing. In: **Proceedings of the Evaluation and Assessment on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2019. (EASE '19), p. 13–22. ISBN 9781450371452. Disponível em: <https://doi.org/10.1145/3319008.3319022>.

ASHIQ, F. **11 Best Automated UI Testing Tools In 2022**. [S.l.], 2022. Acessado: 22 de Jan, 2023 22h18min. Disponível em: <https://www.lambdatest.com/blog/top-ui-automated-testing-tools/>.

BAEK, Y.-M.; BAE, D.-H. Automated model-based android gui testing using multi-level gui comparison criteria. In: **Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2016. (ASE 2016), p. 238–249. ISBN 9781450338455. Disponível em: <https://doi.org/10.1145/2970276.2970313>.

BANERJEE, I. et al. Graphical user interface (gui) testing: Systematic mapping and repository. **Information and Software Technology**, v. 55, n. 10, p. 1679–1694, 2013. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584913000669>.

COLANGELO, H. **Testes no Android com Espresso — Parte 2**. [S.l.], 2016. Acessado: 13 de fev, 2023. Disponível em: <https://medium.com/@heitorcolangelo/testes-no-android-com-espresso-parte-2-5180ee03ed9a#.7hwn7r3fe>.

COPPOLA, R.; MORISIO, M.; TORCHIANO, M. Maintenance of android widget-based gui testing: A taxonomy of test case modification causes. In: **2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.: s.n.], 2018. p. 151–158.

COPPOLA, R.; MORISIO, M.; TORCHIANO, M. Mobile gui testing fragility: A study on open-source android applications. **IEEE Transactions on Reliability**, v. 68, n. 1, p. 67–90, 2019.

COPPOLA, R. et al. Scripted gui testing of android open-source apps: evolution of test code and fragility causes. **Empirical Software Engineering**, p. 3205, 3248, 2019. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-019-09722-9>.

ESBJÖRNSSON, L. **Android GUI Testing : A comparative study of open source Android GUI testing frameworks**. 2015. 48 p.

GAROUSI, V.; FELDERER, M.; MäNTYLä, M. **Guidelines for including the grey literature and conducting multivocal literature reviews in software engineering**. 2017.

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

HELP, S. T. **30 Best GUI Testing Tools For GUI Test Automation [2023 LIST]**. [S.l.], 2023. Acessado: 22 de Jan, 2023. Disponível em: <https://www.softwaretestinghelp.com/best-gui-testing-tools/>.

HU, C.; NEAMTIU, I. Automating gui testing for android applications. In: **Proceedings of the 6th International Workshop on Automation of Software Test**. New York, NY, USA: Association for Computing Machinery, 2011. (AST '11), p. 77–83. ISBN 9781450305921. Disponível em: <https://doi.org/10.1145/1982595.1982612>.

KONG, P. et al. Automated testing of android apps: A systematic literature review. **IEEE Transactions on Reliability**, v. 68, n. 1, p. 45–66, 2019.

KROPP, M.; MORALES, P. **Automated GUI Testing on the Android Platform**. 2010. 67-72 p.

LV, Z. et al. Fastbot2: Reusable automated model-based gui testing for android enhanced by reinforcement learning. In: **Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2023. (ASE '22). ISBN 9781450394758. Disponível em: <https://doi.org/10.1145/3551349.3559505>.

LäMSä, T. Comparison of gui testing tools for android applications. **Tese de Mestrado para o Departamento de Processamento de Informações da Universidade de Oulu, Finlândia**, Oulu, maio 2017. Disponível em: <http://jultika.oulu.fi/files/nbnfioulu-201706142676.pdf>.

MEILIANA et al. Comparison analysis of android gui testing frameworks by using an experimental study. **Procedia Computer Science**, v. 135, p. 736–748, 2018. ISSN 1877-0509. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050918315011>.

MEIRELLES, F. S. **Panorama do Uso de TI no Brasil - 2022**. São Paulo, 2022. Acessado: 22 de Dez, 2022. Disponível em: <https://portal.fgv.br/artigos/panorama-uso-ti-brasil-2022>.

MIN, Y.; CAI, S. Comparing different approaches of gui testing for mobile applications on android platform. **Dissertação de Mestrado em Engenharia de Software do Instituto de Tecnologia de Blekinge**, Karlskrona, set. 2018. Disponível em: <https://www.diva-portal.org/smash/get/diva2:1262909/FULLTEXT02.pdf>.

PETERSEN, K. et al. Systematic mapping studies in software engineering. **Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering**, v. 17, 06 2008.

SILVA, L. T. B. d. Teste de software para aplicativos android: um mapeamento sistemático da literatura. **TCC-Sistemas de informação**, 2014. Trabalho de conclusão de curso de sistemas

de informação da UFPB. Disponível em: <https://repositorio.ufpb.br/jspui/handle/123456789/17057>.

SILVA, N. R. Um mapeamento sistemático da literatura sobre ferramentas para validação de dados em testes de regras de negócio. **Artigo de Graduação apresentado ao Departamento de Ciência da Computação para obtenção de Bacharel em Sistemas de Informação**, Universidade Federal de Lavras - UFLA, Lavras, MG, Brasil, 2019.

SRIVASTAVAL, J.; DWIVEDI, T. Software testing strategy approach on source code applying conditional coverage method. **International Journal of Software Engineering & Applications**, v. 6, p. 25–31, maio 2015.

SU, T. et al. Guided, stochastic model-based gui testing of android apps. In: **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2017. (ESEC/FSE 2017), p. 245–256. ISBN 9781450351058. Disponível em: <https://doi.org/10.1145/3106237.3106298>.

VALENTE, M. T. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**. Independente, 2020. 395 p. Acessado: 22 de Dez, 2022. Disponível em: <https://engsoftmoderna.info/cap8.html>.

VOCKE, H. The practical test pyramid. fev. 2018. Acessado: 30 de Dez, 2022. Disponível em: <https://martinfowler.com/articles/practical-test-pyramid.html>.

WEN, H.-L. et al. Pats: A parallel gui testing framework for android applications. In: **2015 IEEE 39th Annual Computer Software and Applications Conference**. [S.l.: s.n.], 2015. v. 2, p. 210–215.