Universidade Federal de Pernambuco
Centro de Informática

Análise comparativa de ferramentas open-source para testes funcionais GUI mobile em ambiente de CI

Gustavo Silveira da Costa

Recife
Agosto, 2023

Universidade Federal de Pernambuco
Centro de Informática

Gustavo Silveira da Costa

Análise comparativa de ferramentas open-source para testes funcionais GUI mobile em ambiente de CI

Trabalho de graduação apresentado para o programa de Bacharelado em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco em cumprimento parcial dos requisitos para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Prof. Dr. Breno Alexandro Ferreira de Miranda*

Recife
Agosto, 2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Costa, Gustavo Silveira da.
Análise comparativa de ferramentas open-source para testes funcionais GUI mobile em ambiente de CI / Gustavo Silveira da Costa. - Recife, 2023.
27p : il., tab.

Orientador(a): Breno Alexandro Ferreira de Miranda
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.

1. Testes de software. I. Miranda, Breno Alexandro Ferreira de. (Orientação). II. Título.

000 CDD (22.ed.)

Agradecimentos

Gostaria de agradecer principalmente a minha família pelo suporte durante todos esses anos. Sem este não teria sido possível chegar até aqui. Aos colegas de turma durante a graduação pela oportunidade e companheirismo ao aprendermos juntos. Deixar aqui também meu agradecimento a todos os professores pela paciência e conhecimento que foi compartilhado durante esses anos. Por fim agradecer ao orientador, prof. Breno Miranda por toda sua ajuda e suas ideias durante o trabalho realizado.

Abstract

Mobile applications have become an integral part of modern society, driving the demand for robust and efficient mobile UI testing frameworks. With the growing adoption of Continuous Integration (CI) practices, mobile developers seek reliable solutions that can seamlessly integrate into their CI pipelines. This paper presents a comparative analysis of popular mobile UI testing frameworks concerning their suitability for integration into CI environments. The study investigates four widely used mobile UI testing frameworks: Detox, Appium, Calabash, and Maestro. A set of evaluation criteria is established, encompassing aspects such as popularity, ease of use, and test execution speed. The results highlight the strengths and limitations of each testing framework concerning its seamless integration into CI workflows. Overall, this comparative analysis aims to assist mobile developers and organizations in making informed decisions when selecting a mobile UI testing framework for their CI processes. The findings provide insights into the strengths and weaknesses of each framework, enabling developers to align their testing strategies with CI best practices and ultimately improve the quality of their mobile applications.

Keywords: mobile testing, testing frameworks, continuous integration

Resumo

Aplicações mobile hoje são uma parte chave e integral da sociedade, fazendo com que aumente a demanda por frameworks de teste funcionais de interface gráfica (UI) robustos e eficientes. Com o aumento da adoção de práticas de Integração Contínua (CI), desenvolvedores e equipes que trabalham com desenvolvimento móvel vem cada vez mais buscando soluções para automação de testes que possam ser executadas em ambientes de CI. Este trabalho apresenta uma análise comparativa de ferramentas populares para testes funcionais UI focado em sua capacidade de integração em esteiras de CI. O trabalho investiga quatro ferramentas de testes UI: Detox, Appium, Calabash e Maestro. Um conjunto de critérios foi estabelecido para a comparação, envolvendo aspectos como popularidade, facilidade de uso e tempos de execução. Os resultados mostram os pontos fortes e limitações de cada framework em relação a sua capacidade de integração em ambientes de CI. Em geral, essa análise comparativa visa ajudar desenvolvedores mobile e organizações em tomar decisões bem informadas quando selecionando uma ferramenta de testes adequada para seus processos de CI. As descobertas fornecem clareza sobre os pontos fortes e fracos de cada framework, permitindo que desenvolvedores possam alinhar suas estratégias de testes com boas práticas de CI e consequentemente aumentar a qualidade das aplicações móveis desenvolvidas.

Palavras-chave: testes funcionais mobile, frameworks de teste, integração contínua

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Testes de software é uma das etapas mais importantes no processo de desenvolvimento de uma aplicação mobile. Essa etapa se torna ainda mais importante devido às dificuldades e burocracias impostas pelas lojas da Google (Play Store) e Apple (App Store) para a realização do deploy de uma nova aplicação ou atualização da mesma [1].

Se torna necessário um processo de validação e qualidade de software bastante robusto como forma de garantir a qualidade da aplicação final. Dentre os diferentes testes de software, testes funcionais automatizados através da interface gráfica do usuário (em inglês, graphical user interface - GUI) surgem como uma das opções mais interessantes. Esses testes usam scripts simulando ações do usuário e são executados interagindo com a interface gráfica. São executados na mesma versão que será submetida à loja, se diferenciando dos testes unitários que testam partes do código de forma isolada.

A execução desses testes pode ser feita dentro de um processo de integração contínua (Continuous Integration - CI), que são processos, ou esteiras, que são executados durante etapas de desenvolvimento, como por exemplo a submissão de um pull request ou o merge de uma branch na outra no repositório. Dado que a execução de esteiras de CI durante o desenvolvimento de aplicações mobile ainda é algo incomum [2], este trabalho tem como objetivo fazer uma comparação de ferramentas de testes funcionais GUI para aplicações mobile disponíveis e como se comportam em um ambiente de CI.

Existem inúmeras ferramentas para isso, sendo boa parte delas pagas e de código fechado. Para esse trabalho foram comparadas apenas as que são open-source e que possuem versões novas lançadas no ano de 2023.

A principal contribuição deste trabalho é trazer informações que venham a ajudar profissionais da área a escolher qual das ferramentas a adotar em seu ambiente com suas particularidades. A forma de cada equipe atua é algo que varia bastante nas empresas, desde quem é responsável pela execução dos testes a até mesmo a prioridade que é dado a implementação dos mesmos. Com as informações fornecidas pelo trabalho visamos simplificar a análise que costuma ser feita sobre qual é a ferramenta mais adequada antes do início do mesmo.

Este trabalho está organizado da seguinte forma: na Seção 2 são apresentados alguns conceitos básicos e trabalhos relacionados. Na Seção 3 são apresentadas todas as ferramentas e aplicações utilizadas no trabalho. Os critérios que foram selecionados para comparar as ferramentas são apresentados na Seção 4 e os resultados obtidos são apresentados na Seção 5. Na Seção 6 são discutidos os resultados de cada ferramenta. Finalmente, as conclusões e direções para trabalhos futuros são apresentadas na Seção 7.

CAPÍTULO 2

Contexto e Trabalhos Relacionados

2.1 Integração Contínua (CI)

Integração contínua (em inglês, Continuous Integration ou CI) é uma prática de desenvolvimento de software onde diferentes desenvolvedores em uma equipe integram suas alterações com maior frequência em um repositório central [3].

Essas integrações costumam ocorrer através de pull requests e merges em branchs específicas no repositório e é comum a configuração de scripts de testes e validações para que ocorram após ou antes destas integrações ocorrerem, de forma automática. Esses scripts de teste e validação são referidos como esteiras de CI e costumam rodar em máquinas virtuais, que clonam o repositório em uma branch específica e executam os scripts pré definidos dentro do projeto.

Tem o objetivo de permitir detectar erros de integração o mais rápido possível, possibilitando suas correções e minimizando o possível impacto na aplicação. Para aplicações web ou de backend, essas esteiras costumam rodar dentro de containers de docker pré definidos em máquinas virtuais, seja em serviços da nuvem ou em alguma máquina pessoal. Para as aplicações mobile, devido a necessidade de ter a SDK de Android instalada para o testes em Android e o Xcode configurado para o caso de iOS, a execução de esteiras de CI se torna um pouco mais complexa, pois requer máquinas mais poderosas do que aquelas que conseguem rodar imagens mais simples de docker.

Devido a esta dificuldade, neste trabalho para configurar as esteiras de CI foi utilizado um serviço de CI chamado Bitrise [4]. Este serviço fornece máquinas, como Macbook Pro M1, conectadas na nuvem que permitem a integração ao repositório. Com o ambiente de CI do Bitrise configurado e o projeto hosteado no Github, foi configurado para que a cada integração, seja via pull request ou merge, um conjunto de testes e validações seja executado de forma automática e reportando os resultados no próprio repositório.

As ferramentas de testes funcionais de GUI foram comparadas ao serem executadas no ambiente de CI do Bitrise conectado ao repositório central do Github. O objetivo do trabalho é determinar se existe alguma ferramenta mais adequada para a execução nesse ambiente.

2.2 Testes funcionais de GUI

Em desenvolvimento de software, testes funcionais são aqueles que validam que o software funciona corretamente de acordo com as especificações do projeto. Verifica que todos os requisitos especificados foram atendidos e o funcionamento do mesmo está de acordo com o esperado [5].

Testes de interface gráfica ou GUI são aqueles que verificam se o design da aplicação está de acordo com as especificações, ou seja, testam a aplicação a partir de sua interface. Em aplicações mobile esses testes são de extrema importância devido aos diferentes tamanhos de dispositivos e telas disponíveis no mercado. Garantir que a aplicação funcione corretamente em todos eles requer um esforço maior da equipe de desenvolvimento.

Dentro deste contexto, testes funcionais de GUI são aqueles que verificam se a aplicação está funcionando corretamente e atendendo aos requisitos especificados através de ações e interações com a sua interface gráfica. São testes que descrevem ações a serem executadas na tela da aplicação, similares a ações que o usuário final tomaria, e em seguida faz asserções do resultado esperado com base nos requisitos descritos.

2.3 Trabalhos Relacionados

Mohammad, et al [6] comparou diferentes ferramentas utilizando nove critérios de qualidade focando em ferramentas que sejam executadas usando Windows. Algumas das ferramentas já foram descontinuadas e os ambientes de CI raramente utilizam Windows. O trabalho não faz menção a flakiness e nem mede tempos de execução e popularidade.

Okezie, et al [7] comparou ferramentas não só voltadas a aplicações mobile como também aplicações web. Analisou ferramentas pagas, sem ser open source e não faz menção a ambientes de CI. Também não utilizou suscetibilidade a flakiness, popularidade ou tempos de execução como critério de comparação.

Amalfitano, et al [5] foca na comparação de diferentes técnicas de teste, não necessariamente nas ferramentas. Das quais testes funcionais de GUI é apenas uma das opções. O trabalho é focado apenas em aplicações Android.

Abusalim, et al [8], fez uma análise comparativa entre diferentes técnicas de testes de software para aplicações mobile. Dentre elas, testes funcionais, de performance, de segurança e etc. O estudo presente visa comparar ferramentas voltadas apenas a testes funcionais de GUI, ou seja, utilizando apenas uma das técnicas comparadas.

Arif, et al [9] fez uma análise comparativa entre ferramentas de teste funcionais e mencionou outros tipos de testes automatizados que podem ser feitos. Porém das ferramentas utilizadas, algumas já não são mais suportadas oficialmente e outras que foram utilizadas nesse trabalho ainda não existiam. A comparação também foi feita sem levar em conta execução em ambientes de CI e nenhuma das ferramentas chegou a ser implementada.

CAPÍTULO 3

Design da Avaliação

3.1 Aplicação Mobile para Testes

Foi desenvolvida uma aplicação mobile com algumas funcionalidades usando uma tecnologia híbrida chamada React-Native. Essa tecnologia foi escolhida por sua capacidade de gerar aplicações android e ios a partir de uma mesma base de código. Essa aplicação está disponível em um repositório do GitHub [10].

A aplicação se conecta a dois serviços externos reais e públicos. O primeiro é reqres.in [11] usado para autenticação e o segundo é do instituto de arte de Chicago [12]. Através deste, é possível buscar uma listagem de artes e exibir detalhes ao selecionar a mesma.

As telas da aplicação desenvolvida são exibidas na Figura 3.1.

As funcionalidades implementadas foram:

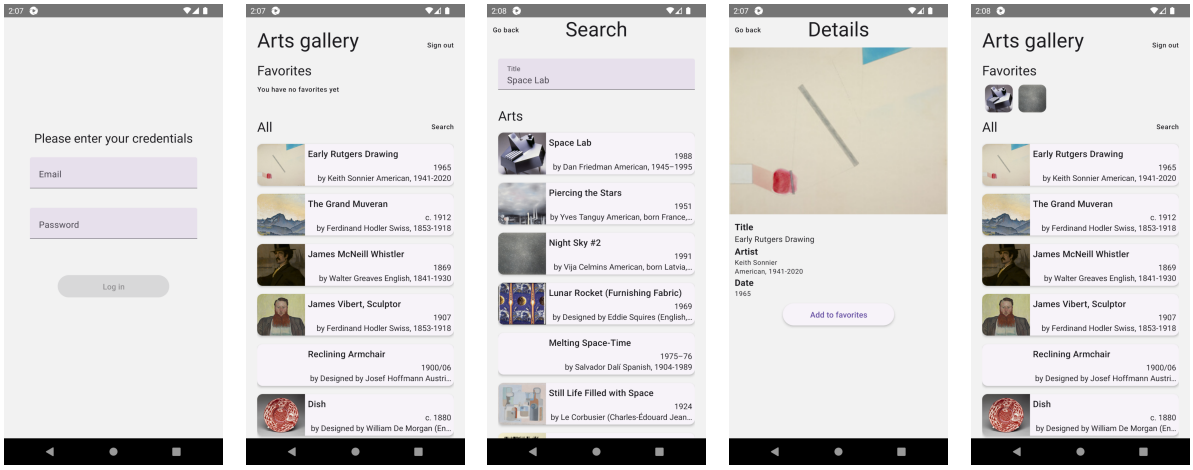
- Login
- Visualizar uma lista de artes
- Visualizar uma lista de artes favoritas
- Ver detalhes da arte ao selecionar
- Poder marcar/desmarcar arte como favorita

3.2 Cenários de testes

A partir dessas funcionalidades foram criados dois cenários de teste: autenticação e funcionalidades de busca e favoritos. Esses cenários de testes foram implementados usando as diferentes ferramentas, sendo executadas na aplicação desenvolvida.

Autenticação (CT #1)

1. Inserir email válido no input de email
2. Inserir senha válida no input de senha
3. Apertar no texto "Log in"
4. Esperar enquanto dados são carregados
5. Verificar que texto "Arts gallery" está visível
6. Apertar no texto "Sign out"



(a) Tela de login (b) Tela inicial sem favoritos (c) Tela de buscas (d) Tela de detalhes (e) Tela inicial com favoritos

Figura 3.1: Telas da aplicação mobile desenvolvida para o estudo

7. Verificar que o texto "Please enter your credentials" está visível

Busca e favoritos (CT #2)

1. Inserir email válido no input de email
2. Inserir senha válido no input de senha
3. Apertar no texto "Log in"
4. Esperar enquanto dados são carregados
5. Verificar que texto "Arts gallery" está visível
6. Apertar no texto "Search"
7. Inserir "Space Lab" no input de busca
8. Esperar enquanto dados são carregados
9. Apertar na arte com nome Space Lab
10. Verificar que o texto "Details" está visível
11. Apertar no texto "Add to favorites"
12. Apertar no texto "Go back" na tela de detalhes
13. Apertar no texto "Go back" na tela de buscas
14. Verificar a presença da arte na listagem de favoritos
15. Apertar na arte a partir da listagem de favoritos
16. Verificar que o texto "Details" está visível

17. Apertar no texto "Remove from favorites"
18. Apertar no texto "Go back"
19. Verificar que o texto "You have no favorites yet" está visível.

3.3 Ferramentas selecionadas

Os critérios adotados para a seleção das ferramentas a serem comparadas foram:

i) Código aberto e disponível no GitHub;
ii) Suporte a uma das duas plataformas (Android ou iOS);
iii) Pode ser executada em ambiente de CI;
iv) Possui novas versões lançadas no ano de 2023.

A partir destes critérios selecionamos quatro frameworks de teste:

- Detox (Android e iOS) [13]
- Appium (Android e iOS) [14]
- Calabash (Android) [15]
- Maestro (Android e iOS) [16]

Existem outras ferramentas bastante utilizadas, porém a maioria destas para testes funcionais de GUI são proprietárias e requerem planos de assinatura anuais. Existem também diversas ferramentas de código aberto que não possuem atualizações recentes, sendo as últimas de alguns anos atrás.

Importante ressaltar também que Calabash possui uma versão para iOS, mas o tutorial se encontra desatualizado e não foi possível completar a sua configuração, fazendo com que apenas a versão em android tenha sido implementada nessa comparação.

3.4 Configuração de CI

O serviço Bitrise fornece diferentes máquinas para execução das esteiras de CI. Para este trabalho foram utilizadas duas máquinas padronizadas para as execuções dos testes, uma para os testes em Android e outra para os testes em iOS.

Para os testes em Android foi utilizada uma imagem de docker com as dependências de Android instaladas e rodando em Ubuntu 20.04, com 4 vCPU @3.1GHz e 16GB RAM.

Para os testes em iOS foi utilizada uma máquina virtual com processador M1, 8CPU @3.2 GHz e 12GB RAM. Esta vem com macOS 13.2 (Ventura) e Xcode 14.3.2.

CAPÍTULO 4

Critérios de Comparação

4.1 Popularidade

Ao comparar ferramentas, um dos principais pontos levados em consideração é a popularidade da mesma. As ferramentas mais usadas costumam possuir uma comunidade maior, principalmente no contexto de ferramentas de código aberto. Estar disponível a mais tempo no mercado, sendo usada por grandes empresas também aumenta a chance da mesma ser adotada.

Com uma comunidade mais robusta, tirar dúvidas a respeito da implementação, ou resolver qualquer tipo de erro que venha a ocorrer se torna uma atividade mais fácil. Essas comunidades muitas vezes também são responsáveis por desenvolver melhorias e ferramentas adicionais que complementam a solução original.

Para tentar medir e comparar as ferramentas selecionadas seguindo critérios de maturidade e popularidade, foram definidas as seguintes métricas.

4.1.1 Métricas do GitHub

Todas as ferramentas comparadas são de código aberto e estão disponíveis no Github. Se aproveitando disso, para medir a popularidade da ferramenta usaremos algumas informações disponíveis no seu respectivo repositório. As métricas usadas serão os números de estrelas e de forks que o repositório possui.

Estrelas no Github servem como uma forma de marcar o repositório como de interesse, sendo uma funcionalidade que se equivale a favoritar o mesmo em sua conta no Github. A métrica de quantas estrelas um repositório possui é útil para quantificar o tamanho da comunidade que tem interesse naquela ferramenta.

A quantidade de forks serve para medir e comparar a quantidade de contribuições da comunidade à ferramenta. Um fork de um repositório de código aberto costuma ser criado por desenvolvedores que tenham interesse em contribuir para o mesmo, seja através de correções de bugs ou desenvolvimento de novas features.

A partir de um fork, o desenvolvedor que não tem acesso ao repositório principal, pode criar a sua cópia local e fazer alterações. A partir deste, é possível testar ideias e propostas antes de submeter essas para validação da equipe que mantém a ferramenta, através de pull requests. A quantidade de forks de um repositório serve como uma métrica interessante para tentar quantificar as contribuições de uma comunidade no desenvolvimento de uma ferramenta de código aberto.

4.1.2 Stack Overflow

O site Stack Overflow é uma plataforma web para que desenvolvedores e profissionais da área de TI possam discutir e tirar dúvidas a respeito de problemas técnicos relacionados a diversas tecnologias.

Dentro do site, as perguntas incluem tags, que servem para identificar e classificar os tópicos

que estão sendo discutidos. Como o Stack Overflow é um dos principais sites de discussão de problemas técnicos, costuma ser a principal referência para tirar dúvidas a respeito do funcionamento de alguma tecnologia, ou relacionado a um erro técnico específico.

Uma vez que as perguntas que usam uma tag específica são de usuários tentando utilizar aquela tecnologia, a quantidade de perguntas associadas a uma tag com o nome da ferramenta foi usada como uma métrica de popularidade da mesma.

4.2 Facilidade de uso

Em equipes de desenvolvimento ágeis, a automação de testes funcionais pode ser responsabilidade de uma equipe de testes e não da equipe de desenvolvimento. Essa separação de responsabilidades serve dois propósitos, permitir que os desenvolvedores foquem apenas nas entregas de funcionalidades e também garante que a validação de requisitos seja feita por uma equipe separada.

Muitas vezes essa equipe de testes não possui conhecimento na tecnologia que está sendo utilizada para desenvolver a aplicação. Isso faz com que a escrita de testes se torne mais difícil uma vez que a equipe responsável não vai ter conhecimento necessário para entender detalhes técnicos da implementação.

Levando em consideração esse cenário, um dos critérios utilizados para avaliar as ferramentas é a facilidade de uso da mesma.

4.2.1 Linguagem de escrita dos testes

Em qualquer ferramenta utilizada, os cenários de teste têm que ser traduzidos em comandos ou scripts que a ferramenta seja capaz de interpretar e executar. A linguagem em que esses scripts são escritos é um dos pontos a ser considerado na adoção de uma ferramenta de testes.

Na maioria dos casos, os testes são escritos utilizando uma linguagem de programação específica a partir de um cliente fornecido pela ferramenta. Outras ferramentas oferecem suporte a múltiplas linguagens, se tornando mais versáteis. As linguagens em que é possível escrever um cenário de testes na ferramenta será um dos critérios na avaliação.

Existem ferramentas porém que permitem que a escrita dos cenários de teste sejam feitas utilizando uma linguagem ubíqua. Linguagem ubíqua é uma linguagem que é definida para servir de ponto comum entre os stakeholders e desenvolvedores. Contém um conjunto de regras e vocabulários com definições específicas pertencendo aquele domínio e que possa ser entendida por todas as pessoas envolvidas no desenvolvimento do software [17].

Existe uma biblioteca chamada Cucumber, que permite definir uma linguagem ubíqua a partir da linguagem Gherkin. Com Cucumber é possível definir comandos específicos, chamados steps, que são escritos usando Gherkin e que por sua vez executam funções definidas pela equipe responsável pelos testes [18].

Cucumber faz o mapeamento entre a linguagem ubíqua e os comandos a serem executados pela ferramenta de teste, fazendo com que os cenários de teste possam ser escritos usando uma linguagem muito próxima da natural. Com isso, o conhecimento dos aspectos técnicos da ferramenta deixa de ser um requisito para a escrita dos cenários de teste, sendo responsabilidade apenas da pessoa responsável pela definição e escrita dos steps.

No contexto de testes funcionais, o cenário de teste ser escrito em uma linguagem ubíqua é um ponto que facilita a implementação e utilização do mesmo. Isso se dá, pois a partir do momento em que a equipe de testes é capaz de escrever cenários utilizando uma linguagem

ubíqua os cenários de teste se tornam mais próximos dos critérios de aceitação definidos pelos stakeholders.

Para a comparação de facilidade de uso, a possibilidade de integração da ferramenta de testes com a ferramenta Cucumber será levada em conta, especificamente se existe uma integração nativa ou se requer um esforço adicional.

4.2.2 Grey vs Black Box Testing

Ao falarmos de testes de software, existem duas técnicas relevantes para o trabalho, Black Box e Grey Box testing. Em Black Box testing a aplicação é tratada como uma caixa preta e a pessoa responsável pelo teste não precisa saber de nada a respeito do funcionamento da mesma para poder executar os cenários de teste [19]. Já em Grey Box testing é necessário um entendimento mínimo a respeito de como a aplicação está estruturada para a criação e execução dos cenários.

Isso faz com que ferramentas Black Box sejam mais fáceis de usar, dado que não é necessário nenhum conhecimento interno sobre a aplicação, ou sobre a tecnologia utilizada para gerar a mesma. Em um contexto de aplicações mobile, uma ferramenta que usa a técnica Black Box necessita apenas da versão da aplicação para executar os testes (APK em android, IPA em iOS). Enquanto as ferramentas Grey Box em alguns casos pode ser necessário gerar uma versão específica para os testes. Uma das consequências dessa diferença é que as ferramentas black box podem ser implementadas em um repositório distinto ao da aplicação.

Devido a essa diferença, se uma ferramenta adota Black Box ou Grey Box testing é um dos fatores que será levado em consideração na comparação relativa a facilidade de uso.

4.3 Tempos de Execução e Falhas

Uma vez que as ferramentas foram devidamente configuradas e com os cenários de teste corretamente implementados, foi configurado no ambiente de CI uma esteira para cada ferramenta em cada uma das plataformas suportadas, ou seja: detox em iOS, detox em android, appium em iOS, appium em android, maestro em android, maestro em iOS e calabash em android.

Com a configuração concluída, foram executados dez testes para cada cenário definido. A partir dessas execuções, foram extraídas três métricas: tempo de execução total médio da esteira, tempo médio apenas da etapa de execução de teste na esteira e uma porcentagem que representa quantas vezes o cenário falhou devido a flakiness nessas dez execuções.

Durante uma esteira de CI, é necessário que o projeto seja configurado antes de executar os testes, como se fosse a primeira vez que estivermos rodando o mesmo. Isso significa que antes de executar o teste em si, precisamos instalar as dependências necessárias e gerar a versão da aplicação que será testada. Ao comparar o tempo de execução total da esteira, o objetivo é determinar se alguma das ferramentas requer uma configuração mais complexa que requer mais tempo. O tempo de execução do teste se refere ao tempo apenas para executar os scripts de teste, sem levar em conta o tempo de instalar dependências ou gerar a aplicação a ser testada.

Por fim, sabendo que os testes estão corretamente implementados e a aplicação funciona, caso algum teste falhe, será analisado se a falha ocorreu devido a flakiness [20]. Tendo uma porcentagem de quantos testes falharam, o objetivo é determinar se alguma ferramenta é mais ou menos suscetível a flakiness.

CAPÍTULO 5

Resultados

Os resultados obtidos estão organizados conforme os critérios definidos para a comparação.

Com o objetivo de apoiar a verificação e replicação independente, disponibilizamos os artefatos produzidos como parte deste trabalho. O pacote de replicação inclui, entre outros, o aplicativo fictício que foi criado usando o React-Native, instruções para configurar os frameworks de teste no ambiente de CI e resultados adicionais:

`https://github.com/gustavos60/TG-2023`

5.1 Popularidade

Na Tabela 5.1 podem ser encontrados os dados obtidos a partir das métricas estabelecidas para popularidade. Esses dados foram obtidos em 09/07/2023 através dos repositórios no github [21]–[24] e pelo site Stack Overflow [25]–[28].

Tabela 5.1: Métricas de popularidade

Ferramenta	Estrelas	Forks	Perguntas no Stack Overflow
Calabash	1.7k	624	595
Detox	10.5k	1.9k	615
Appium	16.6k	5.9k	7697
Maestro	4.1k	141	4

É possível observar pelas métricas obtidas que a ferramenta Appium é a mais adotada dentre as ferramentas testadas. A quantidade mais do que dez vezes maior de perguntas no StackOverflow somados ao número de forks no repositório são indicativos de que para essa ferramenta existe uma comunidade bastante ativa.

5.2 Facilidade de Uso

Na Tabela 5.2 podem ser encontrados os dados referentes a facilidade de uso. As informações foram obtidas através das documentações de cada ferramenta.

A partir das informações de facilidade de uso podemos observar mais vantagens para a adoção de Appium. É a única que está disponível em múltiplas linguagens, pode ser integrada com Cucumber e também é uma ferramenta black box, fazendo com que o projeto de automação não precise ficar no mesmo repositório que a aplicação. Essa separação é importante quando a equipe responsável pelos testes não é a mesma equipe que desenvolve a aplicação.

Tabela 5.2: Métricas de facilidade de uso

Ferramenta	Linguagens de escrita de testes	Suporte nativo a Cucumber	Pode ser integrado ao Cucumber	Black box ou Grey box
Calabash	Gherkin	Sim	–	Black Box
Detox	Javascript	Não	Sim	Grey Box
Appium	Java, Python, Ruby, C#, JavaScript, PHP	Não	Sim	Black Box
Maestro	yaml	Não	Não	Black Box

5.3 Tempos de Execução e Falhas

Na tabela 5.3 estão as informações coletadas após executar os testes em android dez vezes no ambiente de CI.

Tabela 5.3: Tempos de execução em Android

	Calabash		**Detox**		**Appium**		**Maestro**	
Teste	**Total**	**Testes**	**Total**	**Testes**	**Total**	**Testes**	**Total**	**Testes**
#1	*7m 38s*	*42s*	*9m 15s*	*45s*	*8m 24s*	*56s*	*9m 12s*	*1m 39s*
#2	*8m 14s*	*44s*	*9m 5s*	*42s*	*8m 23s*	*59s*	*9m 45s*	*1m 47s*
#3	*8m 20s*	*44s*	*9m 2s*	*35s*	*9m 36s*	*59s*	*9m 38s*	*1m 44s*
#4	*11m 26s*	*46s*	*9m 12s*	*35s*	*10m 40s*	*55s*	*10m 2s*	*1m 46s*
#5	*8m 22s*	*46s*	*8m 48s*	*46s*	*11m 53s*	*56s*	*9m 22s*	*1m 42s*
#6	*7m 35s*	*43s*	*9m 18s*	*45s*	*8m 11s*	*55s*	*9m 30s*	*1m 41s*
#7	*7m 43s*	*43s*	*9m 18s*	*38s*	*8m 47s*	*56s*	*9m 5s*	*1m 43s*
#8	*11m 18s*	*46s*	*9m 28s*	*39s*	*9m 5s*	*55s*	*9m 24s*	*1m 42s*
#9	*11m 9s*	*44s*	*9m 12s*	*37s*	*7m 59s*	*55s*	*9m 16s*	*1m 41s*
#10	*8m*	*44s*	*9m 30s*	*36s*	*8m 6s*	*55s*	*8m 41s*	*1m 43s*

Na tabela 5.4 estão as informações coletadas após executar os testes em iOS dez vezes no ambiente de CI.

Na tabela 5.5 se encontram os dados derivados da tabela 5.3, e na tabela 5.6 se encontram os dados derivados da tabela 5.4, contendo os tempos médios de execução e o percentual de falhas, usado para tentar medir a suscetibilidade a flakiness em suas respectivas plataformas.

Tabela 5.4: Tempos de execução em iOS

	Detox		Appium		Maestro	
Teste	**Total**	**Testes**	**Total**	**Testes**	**Total**	**Testes**
#1	*7m 35s*	*1m 3s*	*9m 4s*	*2m 16s*	*9m 56s*	*2m 13s*
#2	*7m 20s*	*1m 3s*	*9m 19s*	*2m 11s*	*Falhou*	*Falhou*
#3	*7m 17s*	*1m 3s*	*9m 20s*	*2m 34s*	*10m 20s*	*2m 11s*
#4	*7m 40s*	*1m 4s*	*9m 44s*	*2m 39s*	*10m 10s*	*2m 39s*
#5	*7m 12s*	*1m 1s*	*9m 25s*	*2m 35s*	*10m 3s*	*2m 21s*
#6	*7m 41s*	*1m 4s*	*9m 39s*	*2m 49s*	*10m 23s*	*2m 25s*
#7	*7m 18s*	*1m 2s*	*9m 42s*	*2m 33s*	*9m 23s*	*1m 58s*
#8	*7m 11s*	*1m 1s*	*9m 22s*	*2m 18s*	*10m 20s*	*2m 17s*
#9	*7m 20s*	*1m 3s*	*9m 47s*	*2m 44s*	*10m 12s*	*2m 13s*
#10	*7m 37s*	*1m 4s*	*9m 55s*	*2m 29s*	*9m 30s*	*1m 50s*

Tabela 5.5: Resultados para tempos de execução e falhas em Android

	Calabash	**Detox**	**Appium**	**Maestro**
Tempo total (média)	*8m 58s*	*9m 13s*	*9m 6s*	*9m 23s*
Tempo de teste (média)	*44s*	*40s*	*56s*	*1m 43s*
Percentual de Falhas	*0%*	*0%*	*0%*	*0%*

Tabela 5.6: Resultados para tempos de execução e falhas em iOS

	Detox	**Appium**	**Maestro**
Tempo total (média)	*7m 25s*	*9m 32s*	*10m 2s*
Tempo de teste (média)	*1m 3s*	*2m 31s*	*2m 14s*
Percentual de Falhas	*0%*	*0%*	*10%*

CAPÍTULO 6

Discussão

6.1 Ferramentas

Na tabela 6.1 é exibida a classificação final com base nos resultados exibidos na Seção 5. Calabash ficou em último no critério de facilidade pois não foi possível fazer a integração da ferramenta em iOS. Similarmente, Maestro ficou em último no critério de tempo e falhas pois foi a única ferramenta a apresentar falhas por flakiness. Com relação a Appium, o fato de poder ser implementado em múltiplas linguagens e ser integrado ao Cucumber fez com que seja o primeiro no critério de facilidade adotado.

Tabela 6.1: Classificação final por critério

	Popularidade	**Facilidade**	**Tempo de execução**
Appium	*1st*	*1st*	*3rd*
Detox	*2nd*	*3rd*	*1st*
Maestro	*4th*	*2nd*	*4th*
Calabash	*3rd*	*4th*	*2nd*

Maestro

A ferramenta Maestro se destacou por ter sido a mais simples de configurar. Sua instalação pode ser feita com a execução de apenas um script e a detecção do dispositivo a ser utilizado para testes é feita de forma automática, sem precisar de nenhum arquivo de configuração ou instalação de dependências extras. A forma como os testes são escritos, utilizando a linguagem YAML e passos pré definidos e bem documentados também facilitam bastante a sua utilização.

Os principais aspectos negativos que foram observados para tal ferramenta se observam pelos resultados de popularidade e tempos de execução. Por ser uma ferramenta nova, ainda tem pouca adoção e consequentemente possui uma pequena comunidade. Com apenas quatro perguntas postadas no StackOverflow até então, caso seja encontrado algum problema na implementação da ferramenta que não esteja coberto pela documentação, é de se esperar que demore um pouco até que a solução seja encontrada.

Nos tempos de execução percebemos que dentre as ferramentas testadas, Maestro apresentou problemas em Android, mais do que três vezes mais lenta do que Detox e em iOS apesar de ter sido mais rápida do que Appium, é mais do que duas vezes mais lenta do que Detox. Apresentou também problemas de flakiness, sendo a única das ferramentas testadas em que houve falha durante a execução dos testes. Essa falha aconteceu devido a uma limitação da ferramenta na hora de digitar um valor no campo de e-mail, fazendo com que o valor digitado estivesse incorreto e a autenticação falhasse. Como foi uma falha intermitente e devido a uma limitação da ferramenta, esta foi atribuída a flakiness.

Apesar da lentidão nos testes, o tempo total ficou bem próximo das outras ferramentas, o que reforça o ponto do quão simples é a sua configuração. Não ser possível integrar com o Cucumber é um dos pontos negativos, porém a linguagem utilizada apesar de possuir definições mais restritas, é bastante clara e não requer muito conhecimento prévio para adotar.

Detox

Detox se destacou por ter sido a ferramenta mais rápida em tempos de execução quando comparada às outras. No Android a diferença foi pequena, porém no caso do iOS chegou a ser duas vezes mais rápido do que as outras ferramentas testadas.

O fato de ser a única ferramenta grey-box traz pontos positivos e negativos. Como pontos positivos estão alguns mecanismos internos de sincronização que a ferramenta implementa, que servem para atacar os casos mais comuns de flakiness, que são demoras em respostas de backend ou animações mais lentas. Por possuir informações a respeito do funcionamento interno da aplicação, na execução dos testes, se a ferramenta detecta que existem requisições ou animações pendentes, ela espera seu término antes de executar os próximos passos.

Como ponto negativo está a sua configuração mais complexa e o fato de ser necessário gerar uma versão específica para a execução dos testes. Isso faz com que a implementação de Detox tenha que ser feita no mesmo repositório em que a aplicação é gerada, o que não é o caso em ferramentas black box, que precisam apenas da versão já pronta.

A nível de popularidade é a segunda mais adotada, ficando atrás apenas de Appium. O fato de os testes serem escritos apenas em Javascript pode ser minimizado com a integração com a versão de javascript do Cucumber.

Appium

Pelos critérios de popularidade podemos perceber que das ferramentas testadas, Appium é a mais adotada.O número de perguntas no StackOverflow sendo mais de dez vezes superior a próxima ferramenta demonstra que existe uma grande comunidade ao redor da mesma. Esta comunidade é responsável por projetos e integrações que complementam a utilização da ferramenta e também para auxiliar outras pessoas com dúvidas que possam surgir durante a sua implementação.

É a única das ferramentas que está disponível em múltiplas linguagens, podendo também ser integrada ao Cucumber. Sua configuração é mais simples do que a de Detox e um pouco mais complexa do que a do Maestro, porém não é difícil de ser feita. Por ser uma ferramenta black box, o projeto de automação pode ser feito em um repositório separado do qual a aplicação é gerada. Sendo necessário apenas a versão gerada para rodar os testes.

Ao olharmos os tempos de execução, em Android é 40% mais lenta do que Detox e em iOS é a mais lenta de todas. Essa lentidão se dá devido a etapa de testes precisar gerar uma aplicação iOS responsável pela instrumentação. Em ambientes de CI, essa geração não pode ser pulada, porém fora do mesmo, existe a possibilidade de otimizar esse tempo e trazer para tempos mais próximos do Detox.

Calabash

Dentre as ferramentas testadas, Calabash apresentou a maior dificuldade para configuração. Não foi possível completar a configuração da versão iOS da mesma devido a tutoriais desatualizados e em alguns casos indisponíveis. Apesar da ferramenta possuir atualizações recentes,

os tutoriais não são atualizados a alguns anos. A configuração da mesma em Android foi feita após alguma dificuldade e os resultados a nível de tempo de execução e flakiness foram muito bons.

A ferramenta foi apenas 10% mais lenta do que Detox na execução de testes, porém apresentou um tempo total bem próximo das outras. O fato da ferramenta possuir suporte nativo para Cucumber com um conjunto de passos já implementados e fornecer a possibilidade de definição e criação de outros passos é também um ponto bastante positivo.

Com relação a popularidade foi possível perceber que essa ferramenta já foi bastante utilizada, porém recentemente tem sido descontinuada em favor de outras soluções.

6.2 Ameaças à validade

Neste trabalho, todos os testes foram realizados em apenas uma aplicação de pequeno porte, criada especificamente para a análise. Como a aplicação possui poucas funcionalidades, uma das ameaças à validade do trabalho é o fato dos testes executados serem curtos e a aplicação ser de pouca complexidade. Caso fossem utilizados mais aplicativos durante a análise ou o aplicativo fosse mais complexo, poderiam ser obtidos resultados diferentes.

Uma outra ameaça é em relação ao critério de facilidade de uso. Este é um critério bem subjetivo que depende muito da experiência prévia e do conhecimento técnico de quem vai implementar os testes. No trabalho foram selecionados critérios mensuráveis e objetivos, porém a depender da equipe que vai implementar, pode ser que uma ferramenta se destaque como mais fácil devido ao conhecimento maior em determinada linguagem de programação, ou por experiência prévia em alguma delas.

Quando falamos de facilidade de uso, um dos pontos relevantes e que não foi tratado no trabalho é em relação a dificuldade de configuração e manutenção para cada ferramenta. Esse ponto é de extrema relevância quando tratamos da escolha de que ferramenta será adotada. Porém durante a implementação não foi percebida diferença significativa entre as mesmas. É algo que depende muito da experiência prévia do profissional e do conhecimento técnico do mesmo, portanto o tópico seria melhor tratado com pesquisas de profissionais do que através da escolha de algum critério objetivo. A não menção desse tópico no trabalho também se entende como uma ameaça a validade do mesmo.

CAPÍTULO 7

Conclusão e Trabalhos Futuros

De uma forma geral a ferramenta Appium pode ser recomendada, principalmente se a implementação dos testes for feita por uma equipe que não seja a de desenvolvimento. A maior facilidade de implementação e comunidade mais ativa são vantagens mais atrativas do que o tempo de execução.

É importante observar que com Detox os tempos de execução menores combinados a um conjunto de instruções mais claros e declarativos, somados ao fato que flakiness é tratado pela ferramenta e não pela equipe responsável pelos testes, fazem com que também seja uma boa opção.

Calabash, por sua vez, não pode ser recomendada devido a comunidade menor e tutoriais desatualizados. Os resultados obtidos não justificam sua implementação quando comparados às outras ferramentas.

Maestro, por ser a mais simples de implementar, surge como uma excelente alternativa quando os testes são necessários e devem ser implementados de forma mais rápida ou por equipes de desenvolvimento. O fato da mesma poder ser configurada sem a necessidade de adicionar dependências ao projeto ou precisar de arquivos de configuração a tornam bastante vantajosa.

Trabalhos futuros podem vir a focar e se aprofundar em cada um dos critérios selecionados ou na criação de outros critérios que se enxergue como relevante. No que diz respeito a tempos de execução e falhas, as ferramentas podem ser testadas as em uma aplicação de maior porte e com mais funcionalidades, o que geraria informações mais ricas sobre como as ferramentas se comportam com relação a flakiness.

Facilidade de uso poderia ser medida através de pesquisas com profissionais que já usam ou já tentaram usar alguma das ferramenta. Essa pesquisa poderia se aprofundar em qualquer um dos aspectos selecionados ou até mesmo outros que não foram comparados. A importância da possibilidade de integrar uma ferramenta com Cucumber ou outras ferramentas de BDD também é um tópico que poderia ser aprofundado em uma pesquisa.

Referências Bibliográficas

[1] M. Nayebi, B. Adams, and G. Ruhe, "Release practices for mobile apps–what do users and developers think?" In *2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner)*, IEEE, vol. 1, 2016, pp. 552–562 (cit. on p. 1).

[2] L. Cruz, R. Abreu, and D. Lo, "To the attention of mobile software developers: Guess what, test your app!" *Empirical Software Engineering*, vol. 24, pp. 2438–2468, 2019 (cit. on p. 1).

[3] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007 (cit. on p. 2).

[4] *Bitrise*, https://bitrise.io/, Acessado em 29/06/2023 (cit. on p. 2).

[5] D. Amalfitano, N. Amatucci, A. M. Memon, P. Tramontana, and A. R. Fasolino, "A general framework for comparing automatic testing techniques of android mobile apps," *Journal of Systems and Software*, vol. 125, pp. 322–343, 2017 (cit. on pp. 2, 3).

[6] D. R. Mohammad, S. Al-Momani, Y. M. Tashtoush, and M. Alsmirat, "A comparative analysis of quality assurance automated testing tools for windows mobile applications," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, 2019, pp. 0414–0419 (cit. on p. 3).

[7] F. Okezie, I. Odun-Ayo, and S. Bogle, "A critical analysis of software testing tools," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1378, 2019, p. 042 030 (cit. on p. 3).

[8] S. W. AbuSalim, R. Ibrahim, and J. A. Wahab, "Comparative analysis of software testing techniques for mobile applications," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1793, 2021, p. 012 036 (cit. on p. 3).

[9] K. S. Arif and U. Ali, "Mobile application testing tools and their challenges: A comparative study," in *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 2019, pp. 1–6. DOI: 10.1109/ICOMET.2019.8673505 (cit. on p. 3).

[10] *Repositório do projeto*, https://github.com/gustavos60/TG-2023, Acessado em 29/06/2023 (cit. on p. 4).

[11] *Reqres.in*, https://reqres.in, Acessado em 29/06/2023 (cit. on p. 4).

[12] *Art institute of chicago api*, https://api.artic.edu/docs/, Acessado em 29/06/2023, 2023 (cit. on p. 4).

[13] *Detox*, https://wix.github.io/Detox/, Acessado em 29/06/2023 (cit. on p. 6).

[14] *Appium*, https://appium.io/docs/en/2.0/, Acessado em 29/06/2023 (cit. on p. 6).

[15] *Calabash*, https://github.com/calabash, Acessado em 29/06/2023 (cit. on p. 6).

[16] *Maestro*, https://maestro.mobile.dev/, Acessado em 09/07/2023 (cit. on p. 6).

[17] *What is ubiquitous language*, https://tanzu.vmware.com/developer/blog/ubiquitous-language/, Acessado em 09/07/2023 (cit. on p. 8).

[18] M. Wynne, A. Hellesoy, and S. Tooke, *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017 (cit. on p. 8).

[19] *Difference between black-box testing and gray-box testing*, https://www.geeksforgeeks.org/difference-between-black-box-testing-and-gray-box-testing/, Acessado em 09/07/2023 (cit. on p. 9).

[20] G. Pinto, B. Miranda, S. Dissanayake, M. d'Amorim, C. Treude, and A. Bertolino, "What is the vocabulary of flaky tests?" In *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20, Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 492–502, ISBN: 9781450375177. DOI: 10.1145/3379597.3387482. [Online]. Available: https://doi.org/10.1145/3379597.3387482 (cit. on p. 9).

[21] *Github - appium*, https://github.com/appium/appium, Acessado em 09/07/2023 (cit. on p. 10).

[22] *Github - detox*, https://github.com/wix/Detox, Acessado em 09/07/2023 (cit. on p. 10).

[23] *Github - calabash-android*, https://github.com/calabash/calabash-android, Acessado em 09/07/2023 (cit. on p. 10).

[24] *Github - maestro*, https://github.com/mobile-dev-inc/maestro, Acessado em 09/07/2023 (cit. on p. 10).

[25] *Stack overflow - appium*, https://stackoverflow.com/questions/tagged/appium, Acessado em 09/07/2023 (cit. on p. 10).

[26] *Stack overflow - detox*, https://stackoverflow.com/questions/tagged/detox, Acessado em 09/07/2023 (cit. on p. 10).

[27] *Stack overflow - detox*, https://stackoverflow.com/questions/tagged/calabash, Acessado em 09/07/2023 (cit. on p. 10).

[28] *Stack overflow - maestro*, https://stackoverflow.com/questions/tagged/maestro, Acessado em 09/07/2023 (cit. on p. 10).