**AUTOMAÇÃO DE TESTES EM DISPOSITIVOS MÓVEIS**

OLIVEIRA, Camila Gabriel[[1]](#footnote-0)

GONÇALVES, Luciano Bibiano[[2]](#footnote-1)

PERUCCI, Camilo César[[3]](#footnote-2)

Centro Universitário Hermínio Ometto – FHO, Araras – SP, Brasil

**Resumo**

Os aplicativos móveis desempenham um papel fundamental no cotidiano das pessoas, facilitando a interação entre clientes e empresas em uma variedade de contextos.Seja para realizar compras online, acessar serviços bancários, ou mesmo para entretenimento e comunicação, essas ferramentas tornaram-se indispensáveis na vida moderna. No entanto, a crescente demanda por funcionalidades cada vez mais complexas e a diversidade de dispositivos e sistemas operacionais disponíveis tornam essencial garantir a qualidade e o desempenho desses aplicativos. Nesse cenário, a automação de testes em dispositivos móveis surge como uma solução fundamental para assegurar a funcionalidade e a compatibilidade dos aplicativos em diferentes ambientes. Ao automatizar os processos de teste, é possível realizar verificações de forma mais eficiente e abrangente, garantindo que o aplicativo funcione conforme o esperado em uma variedade de cenários. O objetivo deste estudo é identificar e analisar ferramentas de teste automatizado em dispositivos móveis, destacando sua importância tanto para a indústria de desenvolvimento de software quanto para a experiência do usuário. Para isso, foi adotado uma abordagem metodológica que inclui pesquisa bibliográfica, definição de critérios como código aberto, compatibilidade com a linguagem de programação *Python* e especialização em aplicações móveis e seleção das ferramentas. A pesquisa identificou que tanto o *Robot Framework* quanto o *Appium* atendem a esses critérios, pois oferecem suporte para as plataformas *Android* e *iOS*, seus scripts podem ser desenvolvidos utilizando a linguagem *Python* e contam com uma documentação fácil de entender e amigável, tornando-os ótimas opções para automatizar testes em dispositivos móveis. A seleção foi conduzida por meio de um estudo experimental, no qual são avaliadas diversas ferramentas de automação disponíveis no mercado, que melhor se adaptam aos requisitos específicos do projeto. Sugere-se a necessidade de avaliação das ferramentas para identificar qual seja mais adequada para a comunidade de engenharia de software.

*Palavras-chave: Automação de testes móveis, qualidade de aplicativos móveis, ferramentas de automação, eficiência dos aplicativos móveis.*

**1 Introdução**

* 1. **Contextualização**

A proliferação de dispositivos móveis e aplicativos está redefinindo fundamentalmente nossa interação com a tecnologia, no centro dessa revolução, a automação de testes em dispositivos móveis surge como uma peça essencial para garantir a qualidade e o sucesso dos aplicativos. Como enfatizado por Pressman (2011), a qualidade do software é uma interseção de três elementos essenciais. Primeiramente, destaca-se a importância de uma gestão eficiente ao longo do ciclo de desenvolvimento, permitindo a implementação de práticas que garantam a excelência. Em seguida, é imperativo que o produto atenda plenamente às demandas do usuário, oferecendo-lhe uma experiência satisfatória. Por fim, mas não menos crucial, o software deve proporcionar benefícios tangíveis ao desenvolvedor, como a construção de uma reputação empresarial sólida, derivada do êxito nos dois aspectos mencionados anteriormente.

A crescente demanda por automação tem impulsionado o setor de tecnologia, onde a automação desponta como uma solução versátil, abrangendo desde a validação da qualidade do código-fonte até os mais elevados níveis de integração, como os testes de interface. Esta pesquisa se propõe a explorar a automação de testes móveis, abordando metodologias, ferramentas, desafios e benefícios. Ela busca aprimorar a eficiência dos testes móveis, levando em consideração a diversidade de dispositivos e sistemas operacionais. O estudo também enfatiza a importância da automação de testes em um mundo móvel e interconectado, fornecendo uma visão abrangente do campo e suas implicações para os profissionais de tecnologia.

**1.2 Tema de Pesquisa**

Automação de testes em dispositivos móveis, análise da ferramenta Robot Framework e Appium.

**1.3 Motivações e Justificativas**

O crescente uso de dispositivos móveis transformou significativamente a maneira como as pessoas interagem com a tecnologia, tornando os aplicativos móveis uma parte essencial da vida cotidiana. A qualidade e o desempenho dessas aplicações são fundamentais, pois qualquer falha pode levar à frustração dos usuários, perdas financeiras e danos à reputação das marcas. Portanto, a automação de testes em dispositivos móveis é essencial para garantir a funcionalidade perfeita dos aplicativos em diferentes dispositivos e sistemas operacionais. Este estudo justifica-se pela importância da automação de testes tanto para a indústria de desenvolvimento de software quanto para a experiência do usuário final.

**1.4 Objetivos**

**Objetivo Geral:** Este trabalho tem como objetivo identificar e analisar ferramentas de testes automatizados em dispositivos móveis.

**Objetivos Específicos:**

* Realizar uma revisão da literatura visando a compreensão dos conceitos essenciais e das melhores abordagens associadas à automação de testes em dispositivos móveis.
* Definir critérios para realizar uma análise comparativa das ferramentas identificadas.
* Desenvolver um protótipo de software e elaborar uma rotina de testes automatizados nas ferramentas escolhidas.

**2 Revisão Bibliográfica**

* 1. **Conceitos Relacionados**

**Evolução dos Smartphones nos Sistemas Operacionais**

A estrutura de um sistema computacional compreende dois elementos fundamentais: o hardware e o software. Dentro desse contexto, os sistemas operacionais desempenham um papel de extrema importância, sendo programas complexos e essenciais que gerenciam de forma eficiente os componentes necessários para o funcionamento adequado de um computador. Além disso, eles proporcionam uma interface gráfica que simplifica a interação do usuário com o hardware. Entre as categorias mais comuns de sistemas operacionais, destacam-se aqueles projetados para servidores, sistemas embarcados e os desenvolvidos para dispositivos móveis. A categorização dos sistemas móveis é determinada pelo seu funcionamento em dispositivos como smartphones e tablets (TANENBAUM, 2008).

Em 1993, a IBM lançou o primeiro smartphone, visando combinar as funções dos personal digital assistants (PDAs) com os telefones celulares em um único dispositivo. Embora inicialmente não tenha tido sucesso e o projeto tenha sido encerrado, ao longo do tempo o conceito evoluiu, ganhando destaque especialmente no meio empresarial. Entre 2003 e 2007, a maioria dos telefones celulares já podia acessar a internet, seja por redes móveis ou Wi-Fi. Em 2007, a Apple lançou o iPhone, marcando um ponto de viragem na história dos smartphones. No ano seguinte, a T-Mobile lançou o primeiro smartphone a utilizar o sistema operacional Android, desenvolvido pelo Google como concorrente direto do iPhone. Com a popularização dos smartphones, surgiram os aplicativos, programas que exigiam um ambiente de execução compatível com um sistema operacional completo, atualmente dominado por dois sistemas: Android, coordenado pelo Google, e iOS, exclusivo para iPhones da Apple. Cada sistema possui suas próprias características, como interfaces de programação de aplicativos (APIs), bibliotecas e padrões para arquivos binários. Para incentivar o desenvolvimento de aplicativos, as empresas que lançam sistemas operacionais também fornecem um conjunto de ferramentas aos desenvolvedores, criando assim um ecossistema ao redor da plataforma. Além disso, cada sistema especifica uma ou mais linguagens de programação padrão para sua plataforma, como Java e Kotlin para Android, e Objective-C e Swift para iOS.(SIMAS, 2019).

**Testes de software**

Os testes de software têm como objetivo principal garantir que um programa atenda às necessidades dos usuários e identificar possíveis defeitos em seu funcionamento antes de sua disponibilização para uso. Durante a realização dos testes, são empregados dados para a execução do software. Os resultados obtidos são analisados com o intuito de identificar erros, anomalias ou quaisquer informações não funcionais relacionadas à sua performance, utilização de memória, entre outros aspectos" (SOMMERVILLE, 2011).

As atividades associadas aos testes de software podem ser conceptualizadas como uma série de procedimentos destinados a identificar potenciais defeitos. A simples execução dos testes não é suficiente para garantir sua eficácia; é imperativo adotar estratégias que os tornem eficazes. Uma dessas abordagens envolve a aplicação de técnicas, as quais podem ser utilizadas tanto no código-fonte quanto no sistema que já possui uma interface. Vale ressaltar que a responsabilidade pela condução dos testes recai tanto sobre os desenvolvedores quanto sobre os analistas de testes (PRESSMAN, 2011).

**Automação de testes em aplicativos móveis**

falar mais sobre automação

Segundo NETO, (2007 apud [IEEE 610, 1990]), algumas definições fundamentais sobre as capacidades de identificação dos testes podem ser resumidas em três termos essenciais: defeito, erro e falha. As definições a serem empregadas neste contexto seguem a terminologia estabelecida pela Engenharia de Software, conforme padronizada pelo IEEE - Instituto de Engenheiros Elétricos e Eletrônicos.

Ainda segundo NETO, (2007 apud [IEEE 610, 1990]), defeito, erro e falha são definidos como:

* Um defeito é um ato de inconsistência causado por alguém ao tentar executar uma ação, resolver algum problema ou utilizar algum tipo de ferramenta, como, por exemplo, de comandos ou instruções incorretas
* Um erro é o resultado concreto de um defeito em termos de software. É uma diferença explícita entre o resultado obtido e o que era esperado. Em outras palavras, qualquer resultado inesperado ou incorreto durante a execução do programa é considerado um erro.
* Uma falha é o comportamento totalmente inesperado do software. Uma falha pode ser causada por diversos erros, embora alguns erros possam nunca resultar em uma falha.

Figura 1 - Defeito x erro x falha

Diagrama

Descrição gerada automaticamente

Fonte: Neto, 2007.

Conforme Neto (2007) a Figura 1 representa de maneira clara a distinção fundamental entre os conceitos: defeitos, erros e falhas. Defeitos são elementos inerentes ao universo físico, associados à implementação prática de um software e geralmente são provocados por intervenção humana, como o uso inadequado de uma tecnologia. Estes defeitos têm o potencial de desencadear erros no produto, ou seja, podem resultar na construção de um software que se desvia das especificações originalmente planejadas, pertencentes ao universo de informações e os erros, que são consequências dos defeitos, manifestam-se como comportamentos inesperados no software e têm um impacto direto no usuário final, afetando o universo do usuário. Em muitos casos, esses erros podem tornar a aplicação inutilizável.

Atualmente, existem várias maneiras de testar um projeto, no entanto, há técnicas eficazes que continuam sendo amplamente utilizadas em desenvolvimentos, tanto em linguagens estruturadas quanto em orientadas a objetos. A seguir, exploraremos mais sobre essas técnicas, que ainda são bastante empregadas atualmente, e mencionaremos cenários específicos para a técnica funcional.

* 1. **Técnicas de testes e tipos de testes**

**Processo para teste de software**

O teste de software desempenha um papel fundamental na garantia de que um programa funcione conforme sua proposta inicial. Este processo é de suma importância para assegurar que o software não chegue ao usuário final com defeitos ou lacunas significativas. De acordo com Sommerville (2011, p. 144), o processo de teste aborda dois objetivos distintos:

a) Garantir que o software atenda aos requisitos do desenvolvedor e do cliente, com testes específicos para cada requisito em softwares customizados, ou com testes abrangentes para todas as características do sistema em softwares genéricos;

b) Identificar e corrigir comportamentos indesejáveis do software, como falhas, interações inadequadas com outros sistemas, processamentos incorretos e corrupção de dados.

**Desenvolvimento guiado por testes - TDD**

O Desenvolvimento Guiado por Testes (TDD), também conhecido como "Test-Driven Development", é uma abordagem ágil de desenvolvimento que focaliza na habilidade de projetar testes e desenvolvê-los até que o componente esteja totalmente codificado e todos os testes sejam executados sem erros. (apud Pressman & Maxim, 2016, citado em Maschietto, 2020).

O *Test Driven Development* (TDD), conforme descrito por Maschietto (2020), é um modelo centrado em testes que visa diagnosticar o funcionamento do código, onde os testes ocupam uma posição central. Os testes unitários são fundamentais, aplicados em todas as partes do código para avaliar pequenas porções. O modelo segue uma abordagem simplificada em três fases: vermelho (teste automatizado falha), verde (código passa no teste) e refatoração (melhoria da estrutura interna do componente). O ciclo do TDD começa com a elaboração de testes automatizados que inevitavelmente falham, seguidos pela implementação do código para fazê-los passar, e então a refatoração do código para aprimorar sua estrutura interna. Os testes de regressão garantem que alterações no código não causem falhas em unidades já criadas e testadas, pois são reexecutados automaticamente. O TDD promove simplicidade, tanto em sua execução quanto na documentação da equipe, reduzindo o tempo necessário para treinamento. Ferramentas de automação de testes e frameworks de mocks facilitam a prática do desenvolvimento guiado por testes, aumentando a produtividade e a confiabilidade do código. Além disso, o TDD incentiva a qualidade do produto, através da refatoração, e favorece a flexibilidade e estabilidade do código, permitindo adaptações ágeis a novos requisitos e alterações

**Desenvolvimento guiado por comportamento - BBD**

Cezerino e Nascimento (2016) destacam o surgimento do Desenvolvimento Orientado por Comportamento (BDD) como uma resposta à necessidade premente de orientar os desenvolvedores desde o ponto de partida até a compreensão das falhas nos testes, bem como os aspectos a serem testados. Esta abordagem não apenas oferece direcionamento, mas também promove uma linguagem compartilhada entre todos os envolvidos no projeto, facilitando a comunicação e a compreensão mútua. Os princípios basilares do BDD abraçam a ideia de realizar apenas o estritamente necessário, priorizando a entrega de valor aos interessados e mantendo o foco no comportamento desejado. Adicionalmente, essa metodologia estimula o uso de uma linguagem unificada para a análise, eliminando a necessidade de tradução e garantindo um entendimento comum entre os membros da equipe.

**Tipos de testes**

Os tipos de testes consistem em técnicas que auxiliam os testadores a validar rotinas e funções do software, observando sua performance, segurança, usabilidade e outras características importantes que compõem a qualidade do produto. A seguir, temos alguns dos principais tipos de testes e suas características.

Diagrama

Descrição gerada automaticamente

(Fonte: Pressman, 2006)

**Teste Estrutural (Teste caixa-branca)**

Essa técnica de teste trabalha diretamente com o código fonte do software para verificar aspectos como: se as condições são testadas corretamente, como os dados são tratados, se os loops funcionam bem, e se os caminhos lógicos estão corretos (PRESSMAN, 2005). Os critérios de avaliação podem variar dependendo da complexidade e da tecnologia utilizada no desenvolvimento do software, e podem incluir outras questões além das mencionadas anteriormente.

O testador tem acesso ao código fonte do aplicativo e pode criar códigos para conectar diferentes partes do software.

Nesse tipo de teste, olhando diretamente para o código do programa e criando casos de teste que cobrem todas as possíveis situações que o software pode enfrentar. Assim, garantindo que todas as diferentes opções criadas pelas condições do código sejam verificadas.

**Teste Funcional (Teste caixa-preta)**

O método de teste funcional aborda o componente de software a ser testado como uma caixa-preta, sem considerar seu comportamento interno. Dados de entrada são fornecidos, o teste é executado e o resultado é comparado a um resultado esperado previamente definido. O teste é considerado válido se o resultado obtido for igual ao esperado. Essa abordagem é aplicável a vários elementos de software, como métodos, funções internas, programas, componentes individuais ou conjuntos de programas e componentes, assim como funcionalidades completas. Essa abordagem é aplicável em todos os níveis de teste (PRESSMAN, 2005).

**Teste de desempenho**

Os testes de desempenho têm como principal objetivo simular situações de estresse realistas e verificar se o software suporta altas demandas de trabalho. Eles avaliam a capacidade do software de lidar com muitos usuários utilizando a aplicação simultaneamente, ou observam o tempo de comunicação e resposta com o servidor em um desenvolvimento web, por exemplo, para determinar quando essa comunicação se torna inaceitável para a liberação do software. Esses testes visam encontrar falhas que possam prejudicar o desempenho da aplicação. Seu objetivo é verificar se o software está dentro ou fora dos padrões de qualidade de desempenho.(MYERS, 2012, p.126).

**Teste de compatibilidade**

Com a existência de diferentes sistemas operacionais e navegadores, os softwares devem ser capazes de se ajustar conforme cada dispositivo ou navegador, respeitando suas circunstâncias e configurações específicas. Cabe aos testes de compatibilidade validar a adequação e desempenho do software em cada dispositivo e navegador.(MYERS, 2012).

**Análise do valor limite**

Foi observado que muitos erros de software tendem a ocorrer nos extremos dos valores de entrada em vez do meio. Para lidar com isso, um critério de teste explora esses extremos, chamados de limites, de cada conjunto de valores possíveis. Por exemplo, se uma condição de entrada tem uma faixa de valores de A a B, os casos de teste devem incluir os valores A e B, bem como valores imediatamente acima e abaixo deles. Por exemplo, se o intervalo é de 1 a 10, os casos de teste incluiriam 1, 10, 0 e 11.

Esse método ajuda a identificar e corrigir potenciais erros nos extremos dos valores de entrada (PRESSMAN, 2005).

**Teste de aceitação**

O teste de aceitação é o último estágio antes de disponibilizar o software para os usuários. Ele é o responsável por garantir que o software está pronto para ser usado, verificando se todas as funcionalidades foram desenvolvidas corretamente. Basicamente, esse teste procura por falhas no sistema como um todo, para garantir que ele funcione de forma estável. Além disso, ele também confirma se o sistema segue as regras de negócio definidas nos requisitos do projeto. (GAIDARGI, 2021).

**Teste de usabilidade**

O teste de usabilidade é um dos principais testes para a liberação de software. Basicamente, são realizadas simulações do comportamento do usuário final da aplicação. Segundo Rubin (1994), é importante medir tanto o desempenho quanto as preferências do usuário. A usabilidade é avaliada em relação a determinados tipos de usuários e tarefas (Nielsen, 1993).

De acordo com Mili e Tchier (2015), existem cinco conceitos que podem ser considerados para validar a usabilidade, os quais são apresentados a seguir:

* Facilidade de Uso: Verifica se o software é fácil de usar, considerando elementos como simplicidade, padrões de interação e acesso a ajuda. Deve ser compreensível para diferentes tipos de usuários.
* Facilidade de Aprendizado: Avalia se o software é intuitivo e fácil de aprender, observando seus padrões de interação e protocolos.
* Customização: Avalia a capacidade do software de se adaptar aos requisitos e preferências do usuário final, permitindo personalização.
* Calibrabilidade: Analisa o controle que o usuário final tem sobre os ajustes operacionais do software.
* Interoperabilidade: Mede a capacidade do software de funcionar com outras aplicações, utilizando métricas quantitativas e qualitativas.

Uma vantagem do teste de usabilidade é sua capacidade de identificar falhas no sistema que testes automáticos e regressivos podem não detectar.

**Teste de segurança**

As empresas de produtos e serviços digitais em geral estão preocupadas com a segurança de seus sistemas, devido ao aumento dos consumidores preocupados com sua privacidade e dados e por conta de novas leis, como por exemplo a LGPD (Lei Geral de Proteção de Dados Pessoais).

Os testes de segurança são realizados para garantir que as especificações sejam válidas. Esses testes devem abranger diferentes aspectos da aplicação, como os mecanismos de

proteção de memória, a base de dados e outros elementos, adaptando-se de acordo com o tipo de programa que foi desenvolvido. (MAXIM e PRESSMAN, 2016).

**Teste de automatizado**

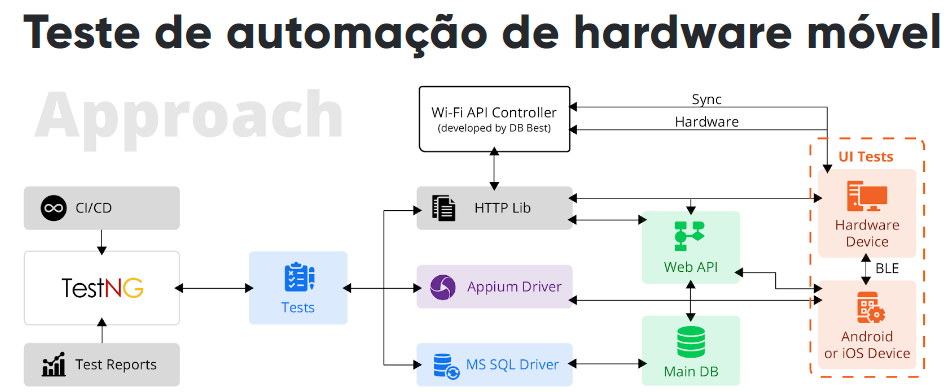
As atividades mais complexas da Engenharia de Software não absolvem a equipe de desenvolvimento da responsabilidade de testar seus produtos. Nesse sentido, a automação de testes surge como um aliado valioso para agilizar as verificações diárias de atualizações do projeto, com o objetivo de reduzir falhas durante o processo por eliminar a execução manual. A automação de testes consiste no uso de ferramentas de software que escrevem e executam testes, reproduzindo os processos realizados pelos usuários no dia a dia (Maxim e Pressman, 2016). Os autores defendem a prática da automação, destacando sua capacidade de melhorar a qualidade dos testes e reduzir custos e tempo de desenvolvimento.

Os benefícios da automação são evidentes ao serem aplicados em operações do sistema como um todo, especialmente com o auxílio dos testes de regressão, o que resulta na redução do tempo e dos custos de desenvolvimento, seja para verificar novos componentes desenvolvidos ou validar os existentes no produto.

Segundo Duval (2007), uma das principais vantagens da automação é o feedback em tempo real, que permite aos usuários, testadores e desenvolvedores identificar possíveis anomalias que precisam ser corrigidas. Além disso, a automação gera documentações úteis que podem servir como guia para o desenvolvimento e correção de anomalias.

**Testes automatizados em dispositivos móveis**

Os testes automatizados em aplicações móveis desempenham um papel importante na redução de tarefas manuais e custosas associadas à validação da segurança, estabilidade e usabilidade do sistema. Geralmente, eles são compostos por um script de automação desenvolvido pelo testador ou desenvolvedor e são executados em um ambiente controlado por técnicas de testes pré-definidas, seja em dispositivos Android, iOS ou mesmo em um emulador web. Esses testes geram, ao final, um relatório detalhado com os resultados obtidos para análise e tomada de decisões. A representação visual dessa estrutura pode ser observada na imagem a seguir.(Maxim e Pressman 2016).



Fonte: Mobile Automation Testing Steps and Process (2020).

* 1. **Ferramentas e Tecnologias**

**Ferramentas de automação**

Atualmente, existe uma variedade de frameworks de automação de dispositivos móveis que estão disponíveis no mercado, cada um com características

próprias para atender a diversas demandas, dependendo da complexidade e necessidade do projeto. Em alguns casos, utilizando mais de um framework para o mesmo projeto. Entre os mais populares, destacam-se: Selenium, Cypress, Playwright, Webdriver, Appium e Robot Framework. A seguir, apresenta-se um breve resumo de cada framework e suas características:

**Selenium**

O framework *Selenium*, conforme documentação (Selenium, 2023), é um conjunto de ferramentas destinadas à automatização em navegadores web, com foco principal na automação de testes em aplicações web. Ele também pode ser empregado para automatizar interfaces web. O framework inclui extensões para emulação e integração de interfaces web com o usuário, simulando as atividades rotineiras executadas pelos usuários finais da aplicação, como inserção de dados, marcação de checkboxes, cliques, entre outras ações.

Além disso, o *Selenium* oferece um servidor de distribuição para alocar navegadores de forma escalável, capaz de lidar eficientemente com um grande número de navegadores simultaneamente. Essa infraestrutura também proporciona suporte para as implementações da especificação *W3C WebDriver*, permitindo que o código-fonte seja escrito de maneira intercambiável, ou seja, possa ser utilizado em diferentes navegadores da web sem a necessidade de grandes modificações.

**Cypress.io**

De acordo com a documentação oficial do (Cypress, 2023), sua característica principal que o diferencia de outros frameworks é sua arquitetura única e sua interface mais amigável aos usuários. Enquanto a maioria dos frameworks de automação de testes segue a arquitetura do Selenium, onde os comandos são executados remotamente pela rede, o Cypress opera de forma oposta, executando os comandos no mesmo loop de execução das aplicações. Ao lado do Cypress, há um servidor node em execução em segundo plano, responsável pela comunicação e sincronização das tarefas, garantindo acesso tanto ao frontend quanto ao backend da aplicação e garantindo uma resposta imediata aos eventos executados pela ferramenta.

Essa capacidade do Cypress de compreender e interagir com todos os níveis da aplicação, desde os mais superficiais até os mais profundos, permite que ele acesse e controle não apenas o que acontece dentro do navegador, mas também os processos que ocorrem fora dele, como no nível do sistema operacional.

Graças a essa amplitude de atuação, os feedbacks fornecidos pelo Cypress durante a execução dos testes tendem a ser mais consistentes em comparação com outras ferramentas de automação. Isso é crucial, pois proporciona uma compreensão mais abrangente e precisa do comportamento da aplicação, tanto em relação à interação com o usuário quanto aos processos internos da aplicação.

**Webdriver**

A ferramenta WebDriver é uma interface de programação que auxilia na automação de comandos para navegadores web. Seu foco principal está na automação de testes em aplicações web, permitindo que desenvolvedores criem scripts que simulem a interação de um usuário real, realizando ações como clicar em botões, preencher campos e navegar entre páginas, entre outras atividades.

Os scripts de automação são geralmente escritos em uma linguagem de programação como JavaScript e TypeScript. O WebDriver comunica-se com o navegador e recebe informações de volta, possibilitando a realização eficiente dos testes em diferentes tipos de navegadores. Existem outras implementações, incluindo o WebDriver do Chrome, WebDriver do Firefox, WebDriver do Safari, entre outros, cada um específico para um navegador(Baker, 2023).

**Appium**  
*Appium* é um *framework open source* para automação de testes em aplicações móveis, web ou híbridas. Além de ser multiplataforma, permitindo a automação de testes em aplicações *iOS* *e Android* por meio de uma API chamada *WebDriver*, que se baseia no *Selenium*. O uso dessa API possibilita a reutilização dos scripts em diversas plataformas. No entanto, uma de suas desvantagens é a falta de suporte incorporado para o desenvolvimento dos scripts, sendo necessário que o programador os desenvolva na linguagem de sua escolha. Basicamente, o *Appium* funciona como um servidor web que utiliza uma arquitetura cliente/servidor para expor uma API REST. Através de uma conexão com um cliente, o *Appium* recebe os scripts de automação da interface do usuário (UI) e os executa sequencialmente na aplicação móvel. Como resposta, ele envia uma mensagem HTTP que representa o resultado dos comandos executados, por exemplo, um XML da UI contendo a conclusão dos resultados das ações executadas. Essa abordagem permite a codificação de cenários de teste em qualquer linguagem de programação que tenha suporte a clientes HTTP (APPIUM, 2023).

**Robot Framework**

O Robot Framework é uma estrutura de automação de código aberto amplamente reconhecida e adotada para testes e automação de processos. Sendo suportado pela Robot Framework Foundation, essa ferramenta é confiável e utilizada por diversas empresas líderes no desenvolvimento de software. Sua natureza aberta e extensível possibilita a integração com várias ferramentas, oferecendo soluções flexíveis e poderosas sem incorrer em custos de licenciamento. Com uma sintaxe intuitiva e extensível por meio de bibliotecas implementadas em diversas linguagens, como Python e Java, o Robot Framework possui um ecossistema robusto de bibliotecas e ferramentas. Esta característica garante a adaptabilidade da ferramenta a uma ampla gama de necessidades e cenários de automação. Além disso, o Robot Framework disponibiliza um boletim informativo mensal, que mantém os usuários atualizados sobre tópicos relevantes, promovendo o aprendizado contínuo e a comunidade em torno da ferramenta. Em resumo, o Robot Framework destaca-se como uma opção poderosa e versátil para automação de testes e processos, oferecendo uma combinação única de facilidade de uso, flexibilidade e suporte comunitário, conforme o site oficial do Robot Framework (Robot Framework, 2024).

Na Tabela 1 a seguir são apresentadas características e funcionalidades de cada ferramenta.

Tabela

Descrição gerada automaticamente com confiança média

Fonte: tabela desenvolvida pelos autores.

Na tabela a seguir encontram-se as vantagens e desvantagens de cada ferramenta.

Texto

Descrição gerada automaticamente com confiança baixa

Fonte: tabela desenvolvida pelos autores.

* 1. **Critérios de seleção das ferramentas que serão analisadas**

A seleção do Robot Framework e do Appium para a realização de testes em dispositivos móveis neste Trabalho de Conclusão de Curso (TCC) se baseia em uma análise cuidadosa das necessidades e requisitos do projeto. Ambas as ferramentas são reconhecidas por sua eficácia, flexibilidade e facilidade de uso, tornando-as escolhas ideais para atender aos objetivos de automação de testes em aplicativos móveis.

O Robot Framework se destaca pela sua linguagem simples e intuitiva, facilitando a redação e manutenção dos testes, o que é crucial para garantir a acessibilidade e colaboração de toda a equipe, independentemente de sua experiência técnica. Sua extensibilidade permite uma integração suave com outras ferramentas e bibliotecas, possibilitando a criação de testes altamente personalizados para atender às necessidades específicas do projeto. Além disso, o suporte nativo ao BDD (Desenvolvimento Orientado por Comportamento) promove a colaboração entre desenvolvedores, testadores e stakeholders na definição e execução de testes com base nos comportamentos esperados do sistema.

Por outro lado, o Appium oferece uma série de vantagens significativas para testes em dispositivos móveis. Com suporte para Android e iOS, ele permite a criação de testes uma vez e a execução em diversas plataformas, economizando tempo e recursos. Sua flexibilidade para escrever testes em várias linguagens de programação e acesso às APIs de nível de sistema do dispositivo móvel contribuem para a criação de testes abrangentes que abordam uma variedade de cenários de uso. Além disso, o suporte contínuo para novas versões do Android e iOS, juntamente com uma comunidade ativa de desenvolvedores e vasta documentação disponível, garantem que o Appium seja uma escolha sólida para automação de testes em dispositivos móveis.

Ao unir o Robot Framework e o Appium para a realização de testes em dispositivos móveis neste TCC, pretende-se fornecer uma abordagem robusta e eficaz para garantir a qualidade e confiabilidade dos aplicativos móveis desenvolvidos, contribuindo para o sucesso do projeto e agregando valor ao conhecimento acadêmico na área de testes de software.

* 1. **Trabalhos relacionados**

Segundo Sá e Silva (2016), a garantia de desempenho de aplicativos em smartphones é crucial para os desenvolvedores de software, dada a ampla variedade de modelos disponíveis. Este estudo investigou ferramentas de automação e execução de testes em dispositivos móveis, além de técnicas para garantir a qualidade do software. Foram realizadas análises e comparações de ferramentas, seguidas pela implementação de scripts de teste automatizados utilizando Selenium, Appium, TestNG e SauceLabs, com programação em JAVA. A execução dos testes ocorreu em um serviço de emulação de dispositivos móveis baseado em computação em nuvem, demonstrando ser uma alternativa viável para simplificar as tarefas de teste, apesar de ser mais lenta que em dispositivos reais.

A monografia de Freitas (2022) discute a importância dos testes automatizados no desenvolvimento de aplicativos móveis para a plataforma Android, devido à crescente dependência desses aplicativos em seu funcionamento eficaz, decorrente do aumento do uso generalizado de softwares. Isso justifica investimentos em ações para melhorar sua qualidade, incluindo atividades de teste. O estudo observa uma tendência crescente de empresas e desenvolvedores em direção à qualidade de software, com foco em testes unitários, manuais e automação. O trabalho propõe a apresentação do Robot Framework, uma ferramenta de automação de testes implementada em Python, destacando seus principais módulos e sua aplicação no desenvolvimento de testes para a plataforma Android. Além disso, inclui uma revisão sistemática abrangente de estudos que exploram diversas abordagens sobre automação de testes em Android.

**3 Metodologia**

A concepção deste projeto seguirá as etapas subsequentes em seu processo de evolução:

**Revisão de literatura:** elaborar uma revisão bibliográfica abrangente sobre a automação de testes em dispositivos móveis, proporcionando uma base teórica sólida e apresentando conceitos e tecnologias pertinentes. Além disso, busca-se aprimorar os elementos do projeto, esclarecendo de forma inequívoca os objetivos da pesquisa, identificando as melhores práticas em automação de testes e realizando uma comparação entre diferentes ferramentas de automação e os testes manuais.

**Identificar e Selecionar ferramentas de testes:** a seleção das ferramentas de automação de testes será realizada com a definição de critérios específicos, avaliando a facilidade de uso, compatibilidade com plataformas móveis e disponibilidade de recursos. Para ilustrar a aplicação prática, será conduzido um estudo de caso, começando pela criação de cenários de teste adequados a dispositivos móveis e pela definição

de métricas de desempenho e qualidade para registrar resultados e coletar dados relevantes.

**Desenvolvimento da engenharia de software:** neste momento, será conduzida uma análise da engenharia de software, desde a fase inicial de levantamento de requisitos até elaboração de diagramas. Essa etapa é de extrema relevância para garantir que o projeto avance de forma eficaz e integrada.

**Desenvolvimento do aplicativo para validação da ferramenta:** Dentro do contexto do estudo de caso, estaremos desenvolvendo um protótipo de aplicativo móvel que simula uma carteira digital. As principais telas incluirão funcionalidades de cadastro, login e inserção de dados.

**Preparação do ambiente de testes:** A preparação do ambiente começa com a instalação da linguagem de programação Python. Em seguida, são instaladas as ferramentas de automação Appium e Robot Framework. Além disso, o Android Studio será instalado para manipular protótipos de aplicativos.

**Criação dos casos de testes:**

Aplicação dos **Testes:** nesta etapa os testes serão realizados e os dados coletados.   
**Análise e Resultados:** a análise será realizada, comparando os resultados obtidos com os testes manuais e avaliando a eficácia da automação em termos de economia de tempo e melhoria da qualidade. Nessa etapa as ferramentas serão validadas.

**4 Resultados**

Definição das ferramentas Robot Framework e Appium.

**5 Considerações Finais**

Descreva as considerações finais sobre os tópicos descritos neste documento. Discuta se os objetivos mencionados foram alcançados, quais as dificuldades encontradas durante o desenvolvimento do projeto, o que ainda pode ser (ou realmente será feito), entre outras reflexões.

**Referências Bibliográficas**

IEEE Std 610.12-1990, **IEEE Standard Glossary of Software Engineering**.

LEOPARDI, M. T.; BECK, C. L. C.; NIETSCHE, E. A.; GONZALES, R. M. B. **Metodologia da pesquisa na saúde**. Santa Maria: Pallotti, 2001. 344p.

NETO, Arilo. Introdução a teste de software. **Engenharia de Software Magazine**, v. 1, p. 22, 2007.

PRESSMAN, Roger S. Engenharia de Software: Uma Abordagem Profissional. 7ªEdição, McGraw-Hill. 780 p., 2011.

**Resolução N° 196/96 sobre Pesquisa envolvendo seres humanos**. Ministério da Saúde – Conselho Nacional de Saúde. Brasília, 1996.

RINGSVEN MK, Bond D. Gerontology and leadership skills for nurses. 2nd ed. Albany (NY): Delmar Publishers; 1996.

SEVERINO, A. J. **Metodologia do trabalho científico**. São Paulo: Cortez, 2000. 279p.

TANENBAUM, A. S. (2008). **Sistemas operacionais modernos**. (pp. 1-15). Pearson Prentice Hall.

VIEIRA, S. **Como escrever uma tese**. São Paulo: Pioneira, Thomson Learning, 2004. 102p.

VIEIRA, S.; HOSSNE, W. S. **Metodologia científica para área de saúde**. Rio de Janeiro: Elsevier, 2001. 192p.

BADKAR, Akshay. Popular Test Automation Frameworks. BrowserStack, 2023. Disponível em: https://www.browserstack.com/guide/best-test-automationframeworks. Acesso em 01 de mai. de 2023.

PLAYWRIGHT (org.) Getting started. Disponível em: < https://playwright.dev/>. Acesso em: 23 abr. 2023.

CYPRESS. Cypress.io. What you'll learn. [S.l.]. CYPRESS, 2022. Disponível em: https://docs.cypress.io/guides/overview/key-differences#What-you-ll-learn. Acesso em: 9 jun. 2023.

ROBOT FRAMEWORK FOUNDATION, **Robot Framework**, 2022. Introduction.

Disponível em: https://robotframework.org/. Acesso em 11 de março de 2024.

ROBOT FRAMEWORK USER GUIDE. **Robot Framework 6.0.2**, 2023. Disponível

em:https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.ht

ml. Acesso em 11 de março de 2024.

CHINNASWAMY, C N; NAGENDRA Mandara; Sreenivas, T H. Robot Framework: A

boon for Automation. **International Journal of Scientific Development and**

**Research (IJSDR)**, 2018. Disponível em:

https://www.ijsdr.org/papers/IJSDR1811080.pdf. Acesso em 11 de março de 2024

Robot Framework. Disponível em: <https://robotframework.org/>. Acesso em: 11 mar. 2024.

Sá, E. C. de; Silva, R. (2016). Automação de teste para dispositivos móveis e execução dos scripts de teste automatizados na nuvem. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade do Sul de Santa Catarina, Florianópolis, Santa Catarina, Brasil.

FREITAS, Nayane Araujo Machado de. Testes automatizados para desenvolvimento mobile em Android. 2022. Monografia (Graduação em Ciência da Computação) - Universidade Federal de Alagoas, Arapiraca, Alagoas.

MYERS, G.; SANDLER, C.; BADGETT, T. The art of Software Testing. 3rd ed. Hoboken: John Wiley & Sons, 2012.

POLO, Rodrigo Cantú. Validação e teste de software. 1. ed. São Paulo: Contentus, 2020. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 01 abr. 2024.

SOMMERVILLE, I. Engenharia de software. 9. ed. São Paulo: Pearson Prentice Hall,2011.

MASCHIETTO, Luís G.; RODRIGUES, Thiago N.; BIANCO, Clicéres M D.; et al. **Processos de Desenvolvimento de Software**. [Grupo A, 2020. *E-book.* ISBN 9786556900520.

GAIDARGI, Juliana. Tudo sobre teste de aceitação. Infonova, 2021. Disponível em: https://www.infonova.com.br/tutoriais/tudo-sobre-teste-de-aceitacao/. Acesso em 14 de abr. de 2023.

Rubin, J. (1994). Handbook of Usability Testing. New York: John Wiley and Sons. Shackell, B. (1986). Ergonomics in Design for Usability. E. Harrison e A. Monk (eds), People and Computers: Designing for Usability. Proceedings of the Second Conference of the British Computer Society. University of York, 44-64.

Nielsen, J. e Lyngbaek, U. (1990). Two field studies of hypermedia usability. In R. McAleese e C. Green (eds), Hypertext: state of the Art. Oxford: Intellect, 64-72.

MILI, A.; TCHIER, F. Software Testing: Concepts and Operations. Ed. Wiley, 2015.

MAXIM, B.R e PRESSMAN, R. S. Engenharia de Software – Uma Abordagem Profissional. 8ª. ed. Porto Alegre: AMGH Editora Ltda, 2016.

SOMMERVILLE, I. Engenharia de software. [S. l.]: Pearson Prentice Hall, 2011.

SIMAS, Victor L.; BORGES, Olimar T.; COUTO, Júlia M C.; et al. **Desenvolvimento para dispositivos móveis - Volume 2**. [Digite o Local da Editora]: Grupo A, 2019. *E-book.* ISBN 9788595029774.

DUVAL, Paul M. Continuous Integration: Improving Software Quality and Reducing Risk.7ª edição. Boston: Addison-Wesley Professional, 2007.

Cezerino, A., & Nascimento, F. P. (2016). Utilização da técnica de desenvolvimento orientado por comportamento (BDD) no levantamento de requisitos. Revista Interdisciplinar Científica Aplicada, 10(3), 40-51.

MOBILE Automation Testing Steps and Process, Utor, 2020. Disponível em: https://utor.com/topic/mobile-automation-steps. Acesso em 30 de abr. 2023

1. FHO|UNIARARAS. Camila Gabriel de Oliveira, 2024, camilagabrieldeoliveira@alunos.fho.edu.br [↑](#footnote-ref-0)
2. FHO|UNIARARAS. Luciano Bibiano Gonçalves, 2024, [luciano.bibiano@alunos.fho.edu.br](mailto:luciano.bibiano@alunos.fho.edu.br) [↑](#footnote-ref-1)
3. FHO|UNIARARAS. Camilo César Perucci, 2024, camiloperucci@fho.edu.br [↑](#footnote-ref-2)