UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Guilherme Carneiro de Farias

Análise comparativa de ferramentas de testes automatizados de ponta a ponta em ambientes de aplicações web

Recife

Março de 2024

Guilherme Carneiro de Farias

Análise comparativa de ferramentas de testes automatizados de ponta a ponta em ambientes de aplicações web

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Cleviton Vinicius Fonsêca Monteiro

Recife

Março de 2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

F224a Farias, Guilherme Carneiro de
Análise comparativa de ferramentas de testes automatizados de ponta a ponta em ambientes de aplicações web / Guilherme Carneiro de Farias. - 2024.
69 f. : il.

Orientador: Cleviton Vinicius Fonseca Monteiro.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas da Informação, Recife, 2024.

1. Automação de testes. 2. Cypress. 3. Playwright. 4. Selenium. 5. Testes de ponta a ponta. I. Monteiro, Cleviton Vinicius Fonseca, orient. II. Título

CDD 004

Guilherme Carneiro de Farias

Análise comparativa de ferramentas de testes automatizados de ponta a ponta em ambientes de aplicações web

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado em: 11 de março de 2024.

BANCA EXAMINADORA

Cleviton Vinicius Fonsêca Monteiro
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Gabriel Alves de Albuquerque Junior
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

Recife
Março de 2024

Agradecimentos

Gostaria de expressar minha profunda gratidão a todas as pessoas que desempenharam papéis fundamentais ao longo da minha jornada acadêmica e profissional, tornando possível a realização deste trabalho de conclusão de curso.

Em primeiro lugar, à minha família, em especial aos meus pais, Zoé José de Farias Filho e Valdenizia da Silva Carneiro Farias, e à minha tia Valquíria da Silva Carneiro. Seu apoio incondicional e as condições proporcionadas ao longo de toda a minha vida foram essenciais para que eu alcançasse este momento.

À faculdade e aos professores, meu sincero reconhecimento. Agradeço a todos os docentes que contribuíram para a minha formação, destacando o professor Cleviton Vinicius Fonsêca Monteiro, que me orientou neste trabalho e foi fundamental para o sucesso desta etapa, e a Gabriel Alves de Albuquerque Júnior, por ministrar a disciplina mais significativa para mim, MPOO. Agradeço também às amizades que floresceram durante esse período, especialmente a Augusto Rosário Gomes Júnior e Thamires Lopes das Mercês, e aos momentos compartilhados, que me motivaram a perseverar mesmo nos momentos desafiadores.

Às oportunidades de emprego proporcionadas pela faculdade, agradeço a Saulo Gomes da Silva, que não só me deu minha primeira chance profissional, mas também orientou-me de maneira imensurável, moldando-me como profissional. Agradeço também a Iury Tavares do Monte, que além de ser um colega de trabalho, tornou-se um amigo que me deu força e motivação para concluir este trabalho.

Por último, mas certamente não menos importante, expresso minha gratidão à minha parceira, Natália Macêdo Ribeiro. Desde antes da faculdade, ela tem sido minha fonte de suporte emocional, ouvindo-me, sugerindo ideias e sendo uma presença constante e significativa em minha vida. Suas contribuições vão muito além do que posso expressar aqui.

A todos, meu mais sincero obrigado. Este trabalho não seria possível sem o apoio e contribuições valiosas de cada um de vocês.

Resumo

Num contexto onde o software ocupa um espaço cada vez mais relevante e complexo na sociedade, é de extrema importância viabilizar meios para que ele seja desenvolvido com qualidade. Um desses meios são os testes automatizados e, no cenário atual, observamos o surgimento de uma variedade de ferramentas nesta área, cada uma com suas nuances e funcionalidades únicas. Diante dessa diversidade de opções, esta pesquisa compara as principais ferramentas de automação de testes ponta a ponta em ambientes de aplicações web, visando facilitar a escolha da mais adequada para cada projeto. O referencial teórico inclui conceitos de Qualidade de Software, Teste de Software e Ferramentas de Arquitetura e Automação de Testes. Três ferramentas foram identificadas e avaliadas: Selenium WebDriver, Cypress e Playwright. O método de pesquisa é exploratório e descritivo, combinando abordagens qualitativas e quantitativas. Os resultados indicam que o Playwright apresenta a melhor combinação de recursos para testes automatizados de ponta a ponta em aplicações web.

Palavras-chave: automação de testes; Cypress; Playwright; Selenium; testes de ponta a ponta.

Abstract

In a context where software occupies an increasingly relevant and complex space in society, it is extremely important to enable means for it to be developed with quality. One of these means is automated testing, and in the current scenario, we observe the emergence of a variety of tools in this area, each with its own nuances and unique functionalities. Faced with this diversity of options, this research compares the main end-to-end testing automation tools in web application environments, aiming to facilitate the choice of the most suitable for each project. The theoretical foundation includes concepts of Software Quality, Software Testing, and Architecture and Automation Testing Tools. Three tools were identified and evaluated: Selenium WebDriver, Cypress, and Playwright. The research method is exploratory and descriptive, combining qualitative and quantitative approaches. The results indicate that Playwright presents the best combination of features for end-to-end automated testing in web applications.

Keywords: Cypress; End-to-End Testing; Playwright; Selenium; Test Automation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BDD	*Behavior Driven Development*
CD	*Continuous Deployment*
CDP	*Chrome DevTools Protocol*
CI	*Continuous Integration*
E2E	*End to End*
IDE	*Integrated Development Environments*
POM	*Page Object Model*
RC	*Remote Control*
V&V	Verificação e Validação
W3C	*World Wide Web Consortium*

Sumário

1 Introdução

No contexto atual, o software tornou-se onipresente, permeando todos os setores da sociedade, desde empresas de todos os tamanhos automatizando processos até governos interagindo com cidadãos por meio de sistemas computacionais. Ele integra uma variedade de dispositivos e produtos de engenharia, desempenhando um papel fundamental na renovação de indústrias e serviços tradicionais, como telecomunicações, transporte urbano, segurança e medicina (VALENTE, 2020).

Para Pressman e Maxim (2016), o cenário contemporâneo de desenvolvimento de software é caracterizado pelo número crescente de usuários e pelo aumento da complexidade. Este fator evidencia a necessidade do software possuir uma qualidade elevada, tendo em vista que se o software falhar, pode resultar em impactos negativos significativos. Além disso, conforme a base de usuários aumenta, a demanda por adaptação também cresce, tornando, portanto, a manutenibilidade uma característica vital de software de qualidade.

Nesse cenário onde o software deve possuir qualidade e manutenibilidade, Fowler (2019) destaca, com base no Estudo de Tornhill e Borg (2022), que resolver problemas em um software de baixa qualidade pode levar mais que o dobro do tempo em comparação com outro software de alta qualidade. Adicionalmente, software com baixa qualidade tende a ter uma densidade de defeitos 15 vezes maior, reforçando a importância de investir em qualidade desde as fases iniciais do desenvolvimento.

Desse modo, torna-se imprescindível a busca pelo software de qualidade. Como enfatiza Sommerville (2019), a maneira mais eficaz de produzir um software de qualidade é através do processo de software, definido como “uma sequência de atividades que leva à produção de um software”. Nesse contexto, existem quatro atividades fundamentais comuns a todos os processos de software: Especificação, Desenvolvimento, Validação e Evolução de software. A Especificação do software envolve a definição clara dos requisitos do sistema, o Desenvolvimento se concentra na elaboração do projeto e na programação do software, a Validação garante que o programa atenda às necessidades do cliente, enquanto a Evolução é a etapa de modificação para refletir a mudança de requisitos tanto do cliente quanto do mercado.

Segundo Pressman e Maxim (2016), durante o processo de software podem ocorrer mudanças importantes, necessitando a realização de testes de regressão, isto é, a reexecução do mesmo subconjunto de testes que já foram executados, para assegurar que as alterações não tenham propagado efeitos colaterais indesejados. No entanto, conforme o software cresce, executar novamente esses testes manualmente se

torna impraticável, seja pelo custo orçamentário ou seja pelo tempo que é necessário para realizá-los de novo, sendo imprescindível a necessidade de utilizar ferramentas de testes automatizados.

A respeito da temática dos testes automatizados, Valente (2020), Vocke (2013), Sommerville (2019), Pressman e Maxim (2016) ressaltam a importância dos testes de ponta a ponta, os quais são os mais confiáveis e eficazes por testar todas as partes integradas de um sistema, simulando da forma mais fiel possível o uso do sistema por um usuário real. Para a realização desses testes podem ser utilizadas diversas ferramentas, como Selenium (2024), Cypress (2024), Playwright (2024), entre outras, cada uma com vantagens e desvantagens para testes de ponta a ponta.

Diante de tantas alternativas, é necessário analisar qual ferramenta utilizar para a criação desses testes. Sob essa perspectiva, este trabalho se propõe a analisar as principais ferramentas de testes automatizados de ponta a ponta em ambientes de aplicações web como forma de elucidar suas diferenças e contribuir na análise da utilização das ferramentas em projetos de software.

1.1 Motivação e Justificativa

À medida que a complexidade do software cresce exponencialmente, a importância de ferramentas de testes automatizados se torna vital. Com a constante evolução de tecnologias e arquiteturas, observamos o surgimento de uma variedade de ferramentas na área de testes, cada uma com suas próprias nuances e funcionalidades únicas.

A diversidade desse cenário torna a pesquisa para compreender as diferenças entre essas ferramentas uma tarefa desafiadora e, muitas vezes, demorada. Nesse contexto, este trabalho propõe-se a realizar uma análise aprofundada das principais ferramentas de testes automatizados de ponta a ponta em ambientes de aplicações web. O objetivo é proporcionar uma visão clara das diferenças entre essas ferramentas, contribuindo significativamente para a tomada de decisões informadas por parte dos profissionais de desenvolvimento de software.

Ao elucidar as características e funcionalidades específicas de cada ferramenta, este trabalho busca facilitar o processo de escolha da ferramenta mais adequada às necessidades específicas de cada projeto. Esta análise visa não apenas atender às demandas presentes, mas também propõe-se a fazer reflexões iniciais quanto a alguns aspectos sobre o futuro das ferramentas analisadas.

1.2 Objetivos

Este trabalho tem como objetivo geral criar uma análise comparativa, entre as principais ferramentas de automação de testes para a Web, a fim de elucidar vantagens e desvantagens de cada um. Para a realização deste objetivo, foram definidos os seguintes objetivos específicos:

- Realizar uma revisão da literatura para identificar as principais ferramentas de automação de testes para aplicações web disponíveis no mercado e compreender o funcionamento dos testes automatizados;
- Selecionar as principais ferramentas de automação de testes para a realização da análise comparativa;
- Definir critérios de avaliação e métricas de desempenho para a análise das ferramentas, contemplando aspectos como velocidade de execução, estabilidade, facilidade de uso, escalabilidade e custo;
- Desenvolver suítes de testes para cada uma das ferramentas selecionadas;
- Executar os testes nas ferramentas escolhidas, coletando dados sobre o tempo de execução, a estabilidade e outros fatores relevantes;
- Analisar os resultados obtidos, identificando as vantagens e desvantagens individuais de cada ferramenta em relação as métricas propostas;
- Apresentar as conclusões da análise comparativa, destacando recomendações e informações que possam auxiliar a organização e outros projetos de automação de testes no futuro.

1.3 Empresa e participação do autor

O presente trabalho utiliza o FlowUp como objeto de análise, demonstrando o uso prático da automação de testes ao fornecer casos de testes reais para serem automatizados, destacando a aplicabilidade direta da automação em um ambiente real de mercado.

O sistema desenvolvido pelo FlowUp é uma ferramenta utilizada diariamente pelos funcionários e equipes de seus clientes. Dada a importância desse sistema para as operações diárias, a empresa reconhece a necessidade de assegurar que o software opere de maneira impecável, sem falhas e garantindo uma experiência consistente.

Nesse contexto, a função do autor nesta análise técnica e de viabilidade é orientar o FlowUp na escolha da ferramenta de automação de testes mais adequada. A

seleção não apenas visa atender às demandas específicas de teste do sistema FlowUp, mas também aprimorar a qualidade e a eficiência dos processos de desenvolvimento e entrega da empresa.

2 Referencial teórico

Com o objetivo elucidar o entendimento da discussão proposta pelo trabalho, serão abordados nas seções a seguir assuntos essenciais, sendo esses: Qualidade de software, Testes de software, Arquitetura das ferramentas de automação e Ferramentas de automação de teste.

2.1 Qualidade de software

A definição de qualidade de software é um conceito que pode apresentar variações significativas entre diferentes indivíduos, pois as partes interessadas, têm perspectivas distintas sobre o que realmente constitui qualidade no contexto do desenvolvimento de software. Fowler (2019) aborda essa multiplicidade de perspectivas ao destacar que, para um usuário, a qualidade pode ser percebida na interface do software e na facilidade com que ele realiza suas tarefas, enquanto, muitas vezes, a arquitetura interna subjacente pode passar despercebida.

Assim, Fowler (2019) categoriza os atributos de qualidade de software em duas vertentes principais: qualidades externas e internas.

- **Qualidades externas:** Referem-se a fatores avaliáveis sem a necessidade de analisar o código-fonte, como facilidade de uso, desempenho e a confiabilidade percebida pelos usuários;
- **Qualidades internas:** Referem-se fatores e características relacionadas à implementação do software, aspectos como modularidade, legibilidade do código, manutenibilidade e testabilidade.

De acordo com Pressman e Maxim (2016, p. 414), a qualidade de software pode ser definida como “uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam”. Essa definição pode ser alinhada com a perspectiva de Fowler (2019), uma vez que destaca a importância de fornecer valor mensurável para os usuários, o que pode ser considerado uma qualidade externa, e também fornecer valor mensurável para aqueles que produzem, que pode ser considerado como a categoria de qualidade interna, ambas descritas por Fowler (2019).

Pressman e Maxim (2016) também identifica elementos cruciais para garantir a qualidade de software, entre eles, padrões e testes. Os padrões, como os estabelecidos pela ISO, fornecem uma base sólida para assegurar a conformidade e qualidade

nos processos de desenvolvimento. Os testes, por sua vez, desempenham um papel fundamental na busca pela qualidade, sendo uma atividade de controle de qualidade, com o objetivo de encontrar erros e garantir que o software atenda aos requisitos explícitos e implícitos dos usuários.

Assim, tanto a perspectiva de Fowler (2019) quanto a de Pressman e Maxim (2016) convergem para a compreensão de que a qualidade de software é uma abordagem multifacetada, envolvendo aspectos perceptíveis pelos usuários finais, bem como elementos internos que requerem expertise em Engenharia de Software.

2.1.1 Fatores-Chave para a Qualidade de Software

A busca pela qualidade de software demanda uma análise criteriosa dos fatores que a determinam. Nesse contexto, a norma ISO/IEC 25010 de 2011 (ISO Central Secretary, 2011) emerge como uma referência essencial, definindo 8 atributos fundamentais para avaliação de um software de qualidade. São eles: Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade e Portabilidade, cada um deles com suas subcaracterísticas como vemos na Figura 1.

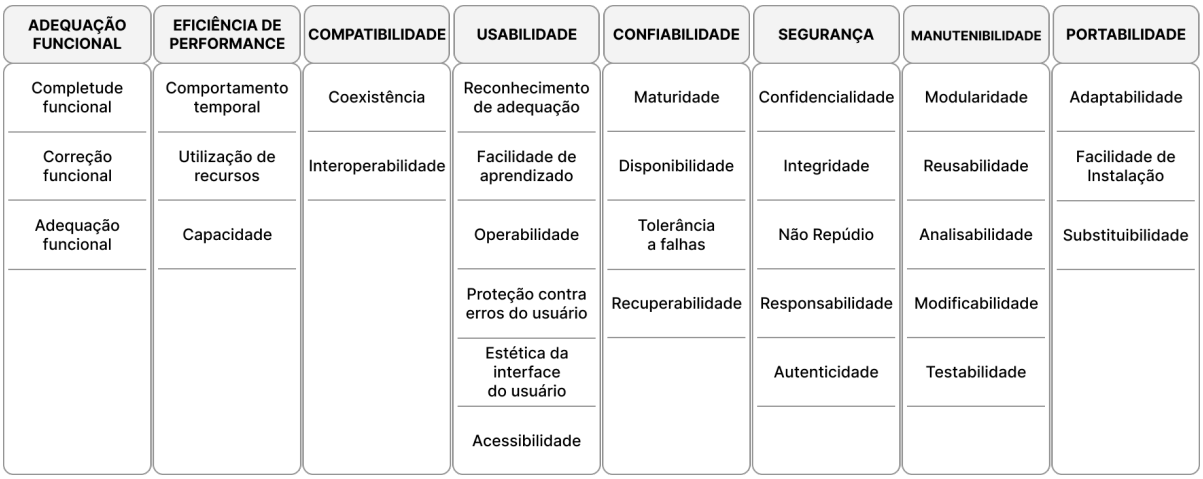

Figura 1 – Fatores-chave para a qualidade de software da ISO/IEC 25010 de 2011

Fonte: elaboração própria.

- **Adequação funcional:** Avalia a capacidade do software em fornecer as funcionalidades necessárias para atender aos requisitos específicos do usuário;
- **Eficiência de performance:** Avalia não apenas o tempo de resposta, mas também como o sistema utiliza recursos e sua capacidade máxima em relação aos requisitos estabelecidos;

- **Compatibilidade:** Avalia a capacidade de um produto, sistema ou componente interagir efetivamente com outros, trocando informações e executando suas funções necessárias, ao compartilhar o mesmo ambiente de hardware ou software;
- **Usabilidade:** Avalia o quão bem um produto ou sistema atende às necessidades dos usuários, promovendo uma experiência positiva com eficácia, eficiência e satisfação em diversos cenários de uso;
- **Confiabilidade:** Avalia o desempenho de um sistema, produto ou componente ao executar funções específicas sob condições determinadas, durante um período específico;
- **Segurança:** Avalia a capacidade de um produto ou sistema em proteger informações e dados, assegurando que apenas pessoas ou outros sistemas autorizados tenham acesso conforme seus tipos e níveis de autorização. Isso se aplica tanto aos dados armazenados quanto aos transmitidos;
- **Manutenibilidade:** Avalia o grau de eficácia e eficiência com que um produto ou sistema pode ser modificado pelos mantenedores pretendidos. Isso inclui correções, melhorias ou adaptações do software a mudanças no ambiente, requisitos e especificações funcionais. A manutenção pode ser realizada por pessoal de suporte especializado, pela equipe de negócios ou operacional, ou pelos próprios usuários finais;
- **Portabilidade:** Avalia o grau de eficácia e eficiência com que um sistema, produto ou componente pode ser transferido de um ambiente operacional ou de uso para outro, incluindo mudanças em hardware, software ou outros aspectos operacionais. Os atributos associados à portabilidade abordam a transferência eficiente entre ambientes, a capacidade de adaptação e a facilidade de instalação e substituição.

2.2 Testes de software

Um dos elementos cruciais para garantir a qualidade de software são os testes de Software como abordado na seção 2.1. Para Pressman e Maxim (2016), o teste de software é um elemento de um tema mais amplo, conhecido como verificação e validação (V & V), que apesar da aparente semelhança, são conceitos que possuem diferenças pertinentes.

A Verificação de software é o processo de conferir se o software cumpre seus requisitos funcionais e não funcionais declarados. Já a validação de software é um processo mais geral, cujo objetivo é assegurar que o software atenda às expectativas do cliente, e vai além da conferência da conformidade com a especificação, para

demonstrar que o software faz o que se espera dele (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2019).

Outra forma que facilita a diferenciação entre estes dois conceitos é a que Barry Boehm (1979), citado por Pressman e Maxim (2016, p. 468), apresenta, através de duas perguntas que ajudam a distinguir e definir cada um:

- **Verificação:** “Estamos criando o produto corretamente?”;
- **Validação:** “Estamos criando o produto certo?”.

No entanto, é crucial compreender as limitações dos testes automatizados, conforme enfatiza Sommerville (2019, p. 204), que destaca que os testes não garantem a ausência total de defeitos, e citando as palavras de Edsger Dijkstra (1972), que declarou que “o teste só consegue mostrar a presença de erros, não a sua ausência”.

Ainda de acordo com o autor, com o teste de software é possível evidenciar se o software atende aos requisitos estabelecidos, ao mesmo tempo em que busca descobrir situações em que o programa pode se comportar de maneira incorreta, indesejável ou fora das especificações.

2.2.1 Abordagens de teste

De acordo com Pressman e Maxim (2016), qualquer produto de engenharia pode ser testado por meio de duas abordagens distintas que se complementam. A primeira abordagem consiste em conhecer a função a qual o produto foi projetado, dando origem ao que é conhecido como teste caixa-preta. Por outro lado, a segunda abordagem envolve o entendimento minucioso do funcionamento interno do produto, caracterizando o teste caixa-branca.

Conforme delineado por Pressman e Maxim (2016), o teste caixa-preta, também chamado de teste comportamental ou teste funcional, focaliza os requisitos funcionais do software. Nesta abordagem, os testes são realizados na interface, com pouca preocupação em relação à estrutura lógica interna do software. Um dos benefícios dessa técnica é que ela não requer o conhecimento da tecnologia empregada ou dos complexos conceitos de implementação aplicados internamente, o que diminui a dificuldade de encontrar profissionais capacitados para modelar os testes. De acordo com o autor, os testes de caixa-preta possibilitam que sejam criados casos de testes com o objetivo encontrar:

- Funções incorretas ou ausentes;
- Erros de interface;

- Erros em estruturas de dados ou acesso a bases de dados externas;
- Erros de comportamento ou de desempenho;
- Erros de inicialização e término.

Conforme delineado por Pressman e Maxim (2016), o teste caixa-branca, também conhecido como teste da caixa-de-vidro ou teste estrutural, são casos de teste que usam a estrutura interna do software. Ou seja, o engenheiro de software possui acesso ao código fonte da aplicação, possibilitando a criação de casos de teste que melhor exerçam o código desenvolvido. De acordo com o autor os testes caixa-branca possibilitam que sejam criados casos testes que:

- Garantam que todos os caminhos independentes de um módulo foram exercitados pelo menos uma vez;
- Exercitem todas as decisões lógicas nos seus estados verdadeiro e falso;
- Executem todos os ciclos em seus limites e dentro de suas fronteiras operacionais;
- Exercitem estruturas de dados internas para assegurar a sua validade.

Analisando estas abordagens, poderia-se concluir que um teste caixa-branca realizado de forma rigorosa resultaria em “programas 100% corretos”. Tudo o que seria preciso fazer seria gerar casos de teste para exercitar a lógica do programa de forma exaustiva. Porém, o autor atenta que o teste exaustivo apresenta certos problemas logísticos, já que o número de caminhos lógicos possíveis pode ser muito grande, mesmo no caso de programas pequenos. Contudo, apesar desse problema, o teste caixa-branca não deve ser considerado impraticável. É possível utilizar esta abordagem selecionando e exercitando um número limitado de caminhos lógicos importantes.

2.2.2 Pirâmide de Testes

A estrutura de pirâmide de testes, introduzida por Cohn (2009). Marco fundamental no contexto de testes de software, ela introduziu a proposta de uma classificação dos testes automatizados em três níveis distintos, delineando a granularidade e o escopo de cada categoria. A representação gráfica desta pirâmide pode ser vista na Figura 2.

Cohn (2009) inicialmente denominou esses níveis como testes de unidade, testes de serviços e testes de interface do usuário. No entanto, Vocke (2013) afirma que, embora a pirâmide seja uma boa diretriz, a nomenclatura original de Cohn pode ser

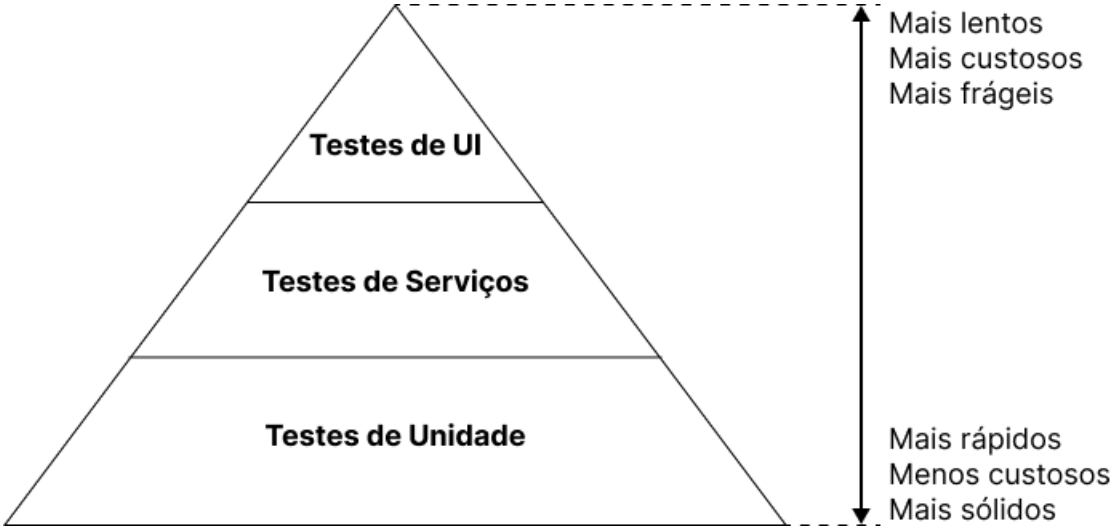

Figura 2 – Pirâmide de teste de Mike Cohn.

Fonte: elaboração própria.

simplista e ele incentiva a adaptação de termos conforme a necessidade da equipe. Em seu livro Valente (2020), utiliza outros termos para se referir aos níveis de testes, sendo eles: Testes de unidade, Testes de integração e Testes de sistema (ou de teste de ponta a ponta).

Os níveis de testes, conforme delineados por Cohn (2009), Vocke (2013), Valente (2020), podem ser categorizados da seguinte forma:

- **Testes de Unidade:** Também chamados de testes unitários, formam a base da pirâmide de testes, focalizando a verificação de pequenas partes do código, geralmente classes isoladas. Esses testes são rápidos, simples de implementar e oferecem feedback imediato ao programador;
- **Testes de Integração:** Também chamados de testes de serviços, estão localizados em um nível intermediário, se concentram em funcionalidades maiores, envolvendo múltiplas classes, pacotes distintos e integrações com bancos de dados e serviços remotos.
- **Testes de Sistema:** Também chamados de Testes de ponta a ponta ou Testes de Interface do Usuário, se encontram no topo da pirâmide, simulando o uso do sistema por um usuário real. Esses testes, mais caros e demorados, verificam a integridade do sistema em seu estado final, garantindo uma validação completa desde a entrada do usuário até a saída desejada (VALENTE, 2020).

Para uma representação visual dos níveis de teste, consulte a Figura 3.

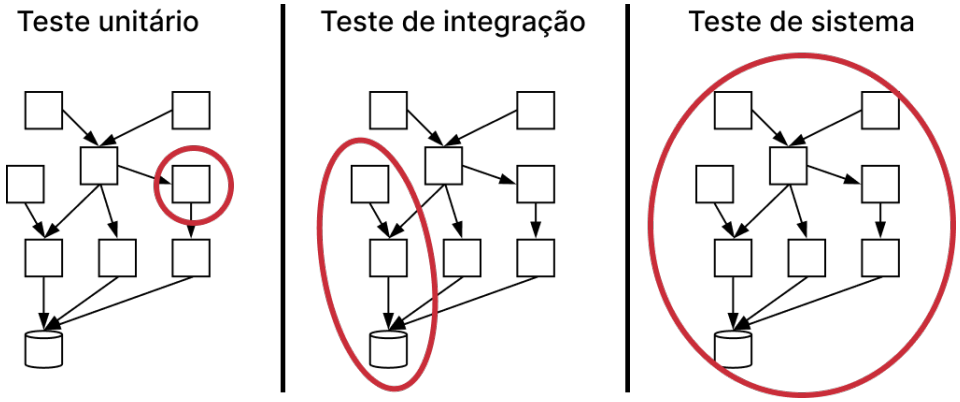

Figura 3 – Escopo dos níveis de teste.

Fonte: adaptado de Valente (2020).

2.3 Arquitetura das ferramentas de automação

Considerando que esta pesquisa lida especificamente com testes automatizados de ponta a ponta, como já mencionado na introdução, esta seção dedica-se a esmiuçar o funcionamento da sua arquitetura.

Podemos distinguir a arquitetura das ferramentas de testes em duas categorias, com base na maneira como automatizam as interações do navegador, especificamente na simulação de ações do usuário (YEEN; SADYM, 2023). São essas:

- **Baixo nível:** Executam comandos remotos que controlam o navegador por meio de protocolos específicos. Exemplos desses tipos de protocolos são o WebDriver e Chrome DevTools Protocol (CDP);
- **Alto nível:** Executam JavaScript diretamente dentro do navegador, possibilitando a simulação de eventos e interações na página. Exemplos desses tipos de ferramentas são o Cypress e o Selenium RC.

2.3.1 Baixo nível

De modo geral, o WebDriver é um protocolo da web compatível com todos os principais navegadores cujos scripts de automação emitem comandos utilizando um protocolo definido por uma especificação W3C, que fornece um conjunto de solicitações HTTP. Esses, por sua vez, definem comandos remotos que interagem com o um servidor WebDriver Proxy, que então se comunica com os navegadores por meio de protocolos internos específicos do navegador (YEEN; SADYM, 2023).

Embora o WebDriver demonstre excelente compatibilidade entre navegadores e APIs projetadas para testes, sua rigidez nas especificações pode resultar em desempenho lento durante a execução. Além disso, ele não oferece suporte a alguns

controles de baixo nível, como rastreamento e interceptação de eventos relacionados à rede.

Já o Chrome DevTools Protocol (CDP) se mostra como uma opção mais rápida comparada ao WebDriver. Originalmente concebido para a depuração e funcionalidades do Chrome DevTools, o CDP teve seu escopo ampliado ao ser incorporado pelo Puppeteer para fins de automação. Ele estabelece comunicação direta com navegadores baseados no Chromium por meio de conexões WebSocket[1], proporcionando um desempenho mais ágil e um controle mais refinado. Esse protocolo possibilita a captura de mensagens do console, a interceptação de solicitações de rede, a simulação de geolocalização, entre outras funcionalidades, demonstrando sua versatilidade e capacidade de oferecer recursos de baixo nível para operações avançadas. No entanto, ele só funciona com navegadores baseados no Chromium (por exemplo, Chrome, Chromium e Edge) e parcialmente no Firefox (YEEN; SADYM, 2023).

É importante destacar que, entre as ferramentas que utilizam o WebDriver, o Selenium WebDriver é a solução mais comumente usada. Isso se deve principalmente à sua longa história, ampla comunidade de usuários e suporte abrangente para várias linguagens de programação, além de oferecer uma variedade de recursos e funcionalidades robustas que melhoram a usabilidade da ferramenta (SELENIUM, 2024).

No que diz respeito ao Chrome DevTools Protocol (CDP), enquanto o Puppeteer se concentra na automação do navegador, o Playwright destaca-se como uma ferramenta voltada para a automação de testes de ponta a ponta. O Playwright oferece recursos mais específicos para essa finalidade, como uma seleção simplificada de elementos, validação do estado dos elementos e relatórios de testes. Esses recursos geralmente exigem implementações ou integrações adicionais no Puppeteer (PUPPETTER, 2024; PLAYWRIGHT, 2024). Além disso, o Playwright foi reconhecido no relatório técnico *ThoughtWorks Technology Radar* com recomendação de adoção, o que reforça sua credibilidade e relevância no cenário atual de desenvolvimento de testes automatizados (THOUGHTWORKS, 2023).

2.3.2 Alto nível

Esta abordagem, destaca-se pela sua capacidade de executar scripts diretamente dentro do navegador, permitindo a simulação de eventos e interações na página de forma mais eficiente. Essa interatividade direta com os elementos da página proporciona, dependendo da forma que implementada, uma experiência de teste mais intuitiva, com feedback em tempo real sobre o comportamento dos testes. Além disso, ao operar dentro do navegador, essas ferramentas conseguem capturar com precisão

[1] Protocolo de comunicação bidirecional em tempo real para aplicações web, permitindo uma conexão persistente entre clientes e servidores.

as interações e comportamentos que ocorrem durante a execução, resultando em uma maior fidelidade com o ambiente real de uso da aplicação. Uma das vantagens dessa abordagem é que ela elimina a necessidade do uso de um WebDriver externo, já que os scripts em JavaScript são interpretados nativamente pelos navegadores modernos, permitindo assim que os testes sejam executados em diferentes navegadores sem a necessidade de modificações significativas nos scripts. A abordagem de Alto Nível, embora seja essencial transpilar[2] a linguagem de programação original para JavaScript, também oferece certa flexibilidade na escolha da linguagem original, desde que haja suporte para a transpilação (YEEN; SADYM, 2023).

Entretanto, é importante reconhecer que, por serem executadas internamente no navegador, essas ferramentas apresentam limitações no controle simultâneo de dois navegadores e podem enfrentar desafios ao lidar com iframes dentro da aplicação e múltiplas abas simultaneamente. Dependendo da complexidade dos scripts, a injeção de JavaScript pode impactar negativamente a performance dos testes, tornando-os mais lentos (CYPRESS, 2024).

Nesse contexto, o Cypress desponta como uma das principais ferramentas que adotam essa abordagem de alto nível, oferecendo uma solução robusta e amigável para a automação de testes. Sua popularidade decorre não apenas de sua capacidade de executar scripts diretamente no navegador, mas também de sua comunidade ativa e suporte abrangente (CYPRESS, 2024).

2.4 Ferramentas de automação de teste

Tendo em mente as arquiteturas de automação de teste abordadas na seção 2.3, este estudo terá como foco as principais ferramentas de cada abordagem. São elas Selenium, Cypress e Playwright.

2.4.1 Selenium

Para compreender o Selenium, é importante entender o contexto da década de 1990, marcada pelo advento dos navegadores web (W3C, 2019). Nesse período, é possível observar que a automação de testes não se estabeleceu como uma prática concreta até a transição para a década de 2000, quando surgiu então o projeto do Selenium (SELENIUM, 2024), incorporando hoje em dia as ferramentas Selenium IDE (do inglês *Integrated Development Environment*), Selenium RC (do inglês *Remote Control*), Selenium WebDriver e Selenium Grid.

[2] Processo de converter código de uma linguagem para outra, preservando funcionalidade e lógica.

Com base nas informações fornecidas pelo Selenium (2024), o Selenium IDE foi concebido como um plugin destinado aos navegadores Firefox, Chrome e Edge. Seu propósito é registrar as interações manuais realizadas no navegador onde o plugin está instalado, gerando scripts de teste que simplificam a automação e possibilitam a reexecução das interações com a aplicação.

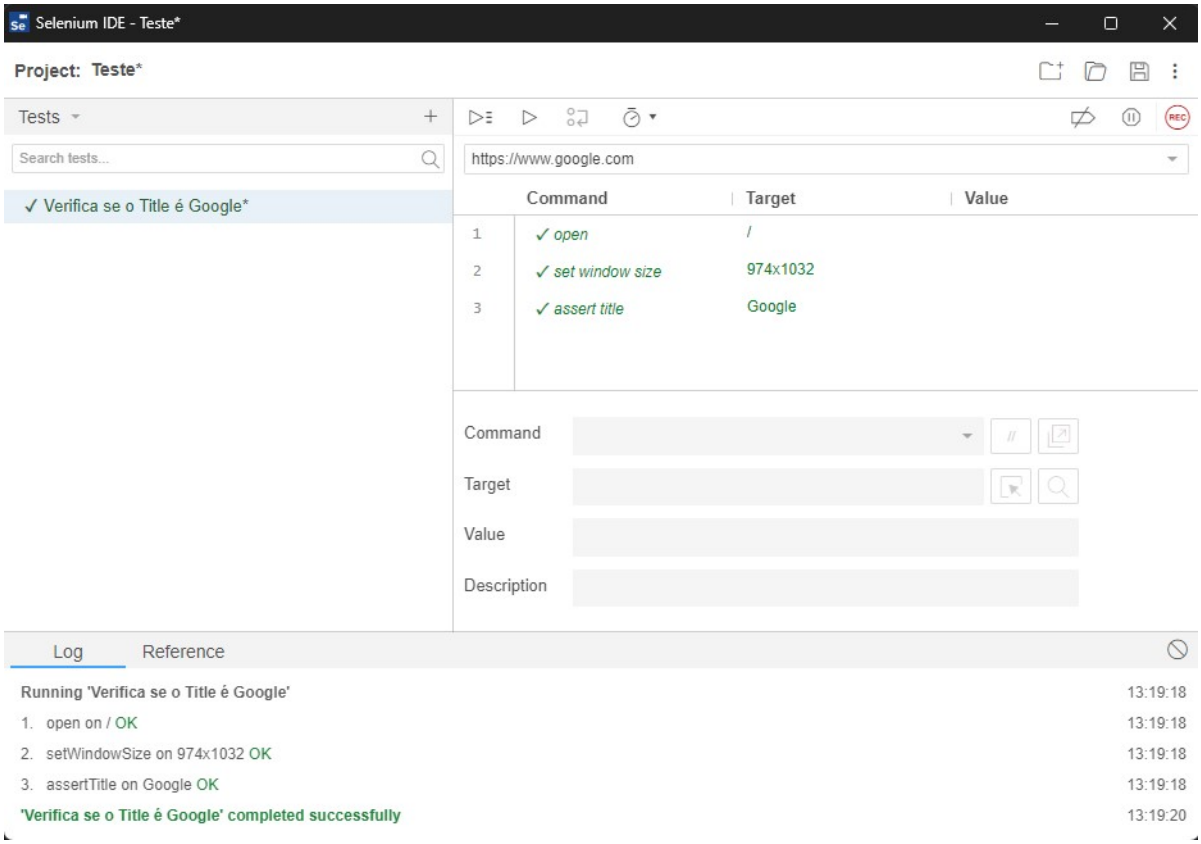

Figura 4 – Interface do plugin Selenium IDE para o Chrome.

Fonte: elaboração própria.

Por meio de sua interface, como ilustrado na Figura 4, o Selenium IDE oferece funcionalidades que permitem a gravação, edição, execução e exportação de casos de teste como scripts para serem utilizados em outras ferramentas do Selenium, como o Selenium WebDriver.

Por mais que o Selenium IDE, seja uma ferramenta que agiliza o desenvolvimento e proporciona uma excelente introdução à sintaxe dos scripts de testes do Selenium, apresenta limitações notáveis, como difícil automação em cenários complexos, testes orientados a dados, falta de detalhamento nos relatórios para análises aprofundadas e suporte limitado de navegadores, restringindo-se a Chrome, Firefox e Edge.

De acordo com as informações disponibilizadas pelo Selenium (2024), o Selenium RC desempenhou um papel central como o principal projeto da plataforma até a fusão do WebDriver com o Selenium. Com essa fusão, o suporte ao Selenium RC foi descontinuado.

O Selenium RC adota uma abordagem de alto nível (seção 2.3.2) e depende de uma arquitetura cliente-servidor, conforme ilustrado na Figura 5. Nessa arquitetura, o cliente estabelece uma conexão com o servidor do Selenium RC, que inicia um navegador com uma URL injetando o JavaScript do Selenium-Core na página. A partir desse ponto, o cliente envia comandos para o servidor do Selenium RC, que os interpreta acionando a execução do JavaScript correspondente para executar esse comando dentro do navegador. Então para executar os comandos o navegador solicita ao Selenium RC o conteúdo do site, este se comunica com o servidor Web, solicitando a página. Quando recebida, a página é enviada ao navegador para execução dos comandos.

Essa arquitetura proporciona uma abordagem eficaz para automatizar cenários de testes mais complexos. Ela mantém também a flexibilidade de escrever testes em diversas linguagens de programação, incluindo Java, Ruby, Python, Perl, PHP e .Net, pois o Selenium RC ficará responsável de enviar o comando em JavaScript equivalente para o navegador pelo Selenium-Core.

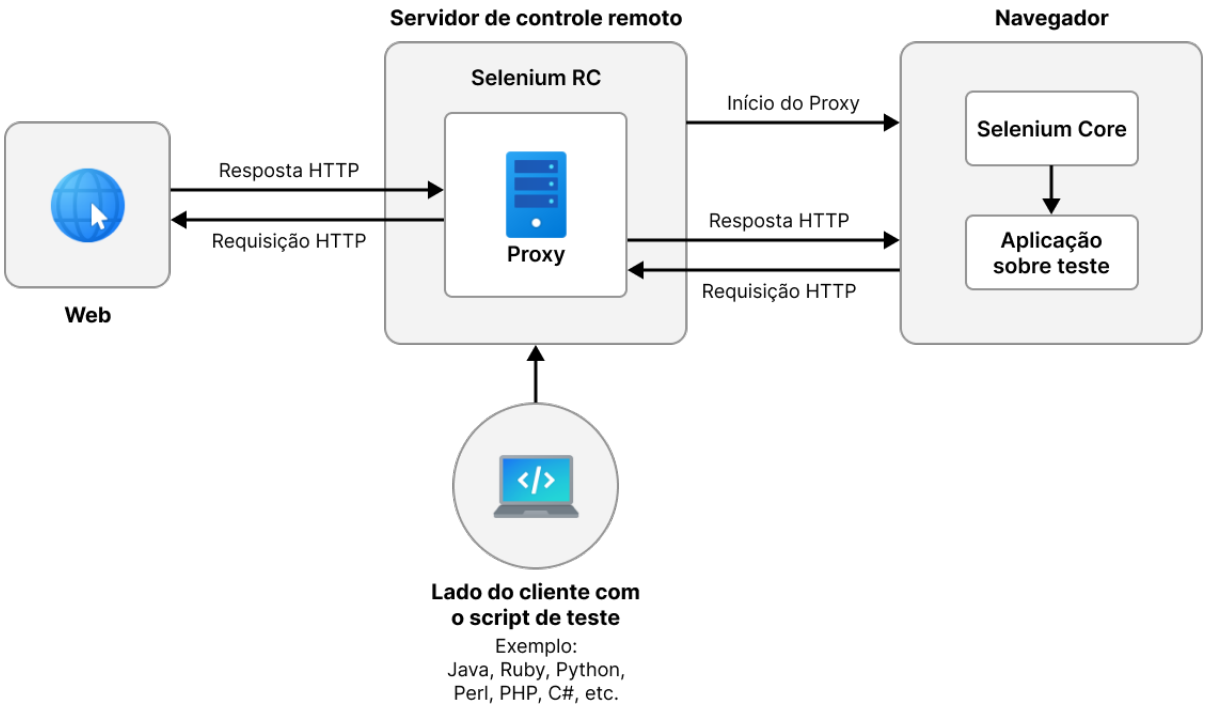

Figura 5 – Diagrama da arquitetura do Selenium RC.

Fonte: elaboração própria.

No entanto, por mais que utilize uma abordagem de alto nível, por se tratar de uma arquitetura de cliente-servidor, o Selenium não possui acesso ao que está acontecendo em tempo real no navegador. Além disso, a necessidade de um servidor separado para a execução dos testes adiciona complexidade à configuração e à manutenção do ambiente de teste. Sendo assim, devido à comunicação com o navegador por meio de um servidor proxy que interpreta o comando e envia para o navegador, a performance do Selenium RC é comprometida negativamente.

Ainda segundo as informações disponibilizadas pelo Selenium (2024), o Selenium WebDriver representa a principal ferramenta de automação de testes do grupo Selenium, operando como uma solução de baixo nível (seção 2.3.1), que pode ser visualizada na Figura 6. O Selenium WebDriver, assim como o Selenium RC, também possui uma grande gama de linguagens de programação para escrever testes, tais como: Java, Python, Ruby, C#, JavaScript entre outras.

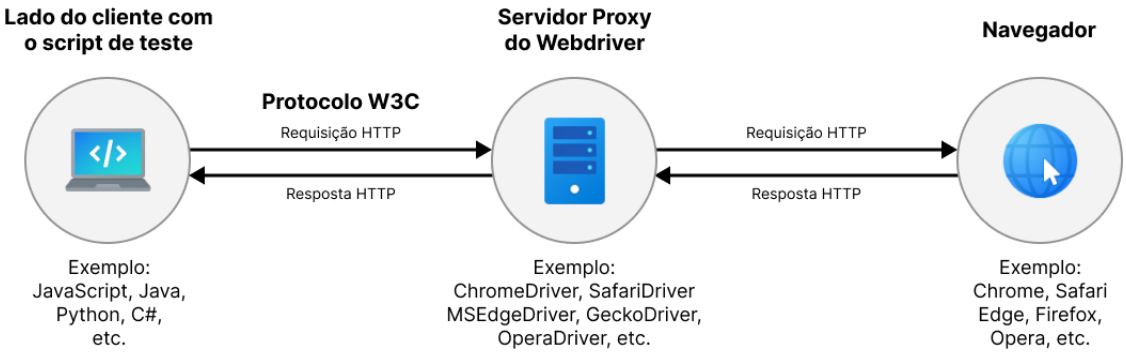

Figura 6 – Diagrama da arquitetura do Selenium WebDriver.

Fonte: elaboração própria.

Ele também é reconhecido por oferecer suporte a uma variedade de navegadores, possibilitando a execução de testes em ambientes diversificados. Contudo, vale ressaltar que para cada navegador é necessário utilizar um driver específico. Isso implica que, para realizar testes em diferentes navegadores, é preciso instalar os drivers correspondentes a cada um deles.

Também com base nas informações fornecidas pelo Selenium (2024), o Selenium Grid otimiza a execução de scripts WebDriver em máquinas remotas, direcionando os comandos enviados pelo cliente para instâncias remotas do navegador. Seu propósito fundamental é facilitar a execução de testes em paralelo em várias máquinas, permitindo testes em diferentes versões de navegador, viabilizando testes de plataformas cruzadas e reduzindo o tempo total de execução do conjunto de testes.

Para atingir esse objetivo, é essencial configurar um servidor (hub) que receba as solicitações do cliente WebDriver, encaminhando os comandos de teste para as unidades remotas (nós). Os nós, por sua vez, são dispositivos remotos equipados com um sistema operacional nativo e um WebDriver remoto. Eles recebem as solicitações do servidor na forma de comandos de teste e os executam usando o WebDriver, como pode ser visto na Figura 7 a seguir.

Ao optar pelo Selenium Grid para testes em paralelo, enfrentamos desafios significativos. A configuração e manutenção do hub e dos nós podem se revelar complexas, demandando recursos como máquinas, hardware, software e uma infraestrutura de rede robusta. A supervisão e solução de problemas tanto no hub quanto nos nós são tarefas cruciais. A sincronização e coordenação dos testes entre os nós apresentam

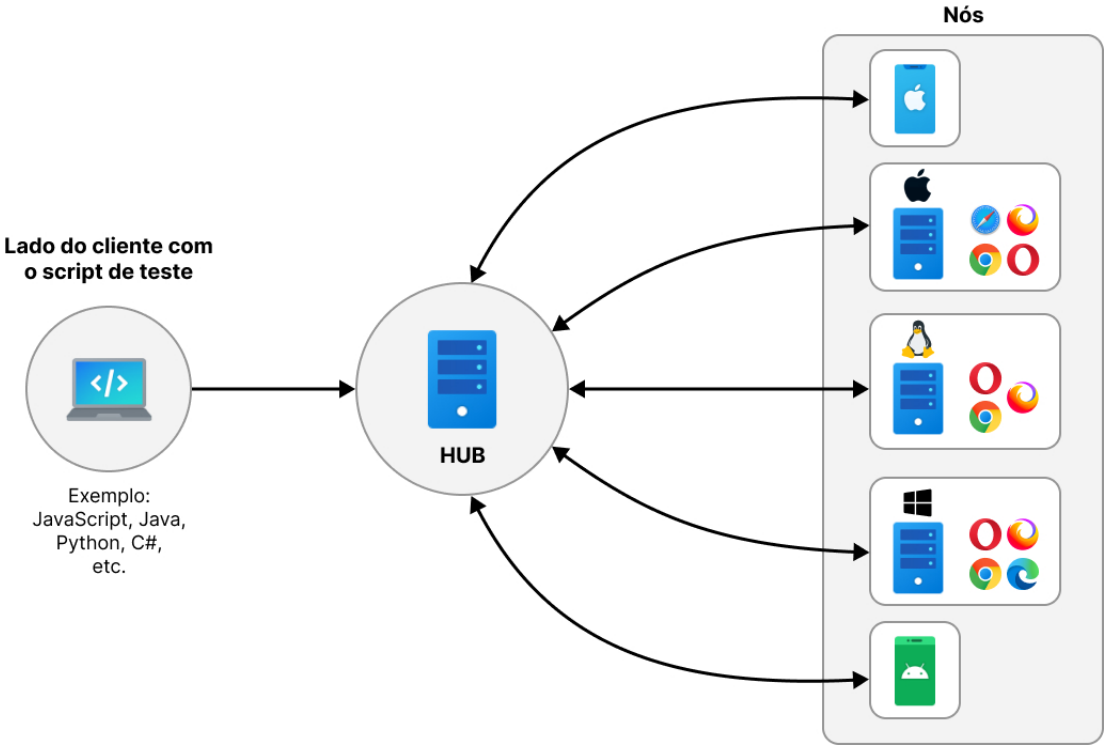

Figura 7 – Diagrama da arquitetura do Selenium Grid.

Fonte: elaboração própria.

dificuldades, requerendo a garantia da independência, confiabilidade e consistência dos testes, assim como a gestão dos dados, resultados e relatórios. Além disso, a exposição dos testes a máquinas externas envolve riscos, necessitando de medidas de segurança contra acesso não autorizado, ataques mal-intencionados e violações de dados.

2.4.2 Cypress

O Cypress é reconhecido como a principal ferramenta de testes de alto nível, como destacado na seção 2.3.2 e sua arquitetura é ilustrada na Figura 8.

Por trás do Cypress está um processo de servidor Node.JS. O Cypress e o Node.js se comunicam utilizando o protocolo Websockets, assim sincronizando e executando tarefas constantemente em nome um do outro. No processo, o Node.js inicia um navegador com um proxy e o configura com dois iframes: um destinado ao Cypress e outro à aplicação em teste. O Cypress, situado em um iframe com o domínio localhost, difere do domínio da aplicação. Para superar essa diferença, um script é injetado no iframe da aplicação, alterando seu domínio para localhost. Esse ajuste possibilita a comunicação entre os iframes, concedendo ao Cypress acesso a todos os objetos da aplicação, incluindo o DOM, a janela do navegador e os seus mecanismos de armazenamento. O Cypress também opera na camada de rede, capturando e modificando

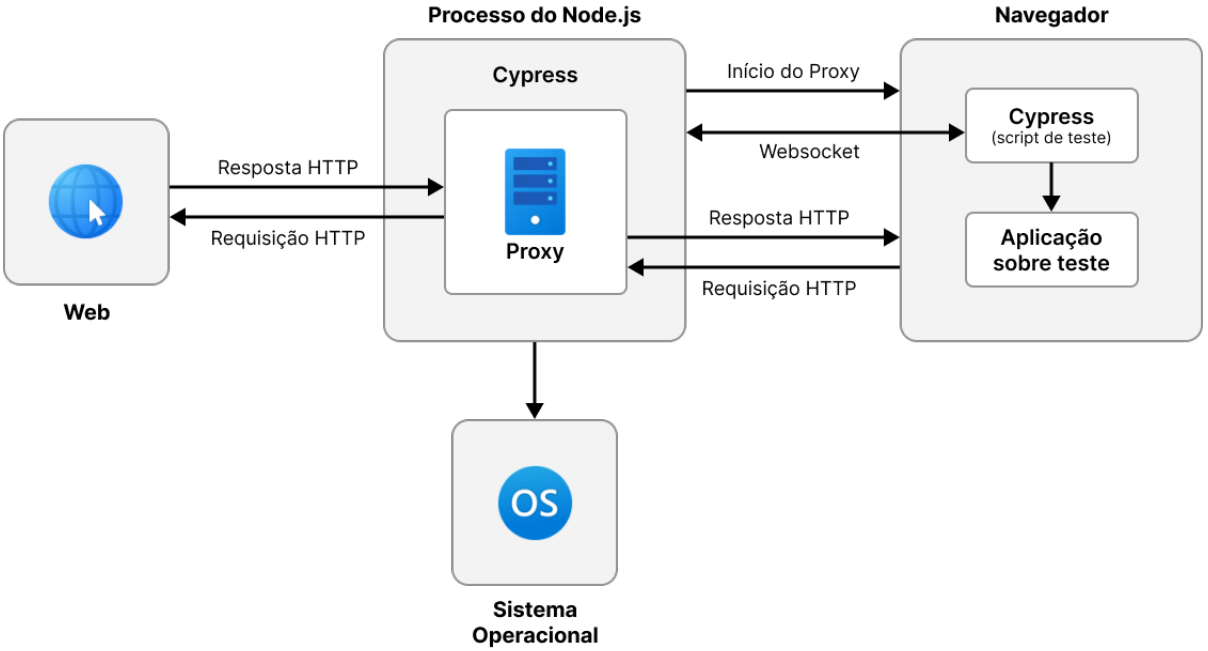

Figura 8 – Diagrama da arquitetura do Cypress.

Fonte: elaboração própria.

solicitações em tempo real conforme necessário (CYPRESS, 2024).

Além do controle do navegador, a interação entre o Cypress e o Node.js é crucial para solicitar privilégios do sistema operacional, permitindo capturas de tela e gravação de vídeos. Esses recursos são essenciais para aprimorar os relatórios gerados pela ferramenta (CYPRESS, 2024).

Como o script de testes é executado diretamente no navegador, o Cypress oferece suporte apenas à escrita de testes em JavaScript e TypeScript (que deve ser transpilado para JavaScript antes da execução dos testes). Como o Cypress é uma ferramenta especializada em testar aplicações enquanto estão em desenvolvimento, a realização de testes em ambientes de homologação ou produção pode resultar em impactos negativos na performance devido a essa natureza (CYPRESS, 2024).

2.4.3 Playwright

Lançado em 2020 e mantido pela Microsoft, o Playwright destaca-se como uma ferramenta de automação de testes com uma arquitetura de baixo nível, utilizando o CDP para comunicação (PLAYWRIGHT, 2024). No entanto, é importante observar, conforme mencionado na seção arquitetura de baixo nível 2.3.1, que o CDP é um protocolo exclusivo para os navegadores baseados no Chromium. Para superar essa limitação e oferecer suporte a uma variedade de navegadores, o Playwright adotou uma abordagem inovadora, implementando sua própria versão do protocolo para garantir compatibilidade com navegadores como Chrome, Chromium, Edge, Opera, Safari, Fi-

refox, entre outros (PLAYWRIGHT, 2024).

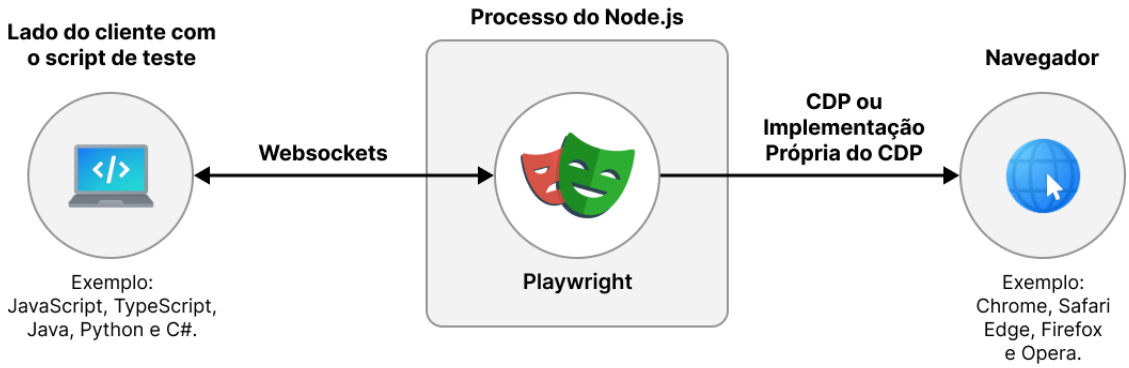

Figura 9 – Diagrama da arquitetura do Playwright.

Fonte: elaboração própria.

A flexibilidade do Playwright vai além do suporte a vários navegadores. A ferramenta possibilita a execução de testes em diferentes sistemas operacionais, incluindo Windows, Linux, macOS e emulação de dispositivos móveis nativamente pelo Google Chrome para Android e do Safari para IOS. Além disso, destaca-se por ser acessível em diversas linguagens de programação, como TypeScript, JavaScript, Python, .Net e Java (PLAYWRIGHT, 2024).

3 Trabalhos relacionados

Durante o processo de revisão da literatura, foram encontrados vários estudos que comparam ferramentas de testes de ponta a ponta em ambientes de aplicações web. Os trabalhos a seguir serviram como base para a execução da análise e desenvolvimento deste trabalho. A contribuição de cada um está descrita a seguir.

Silva (2019) realizou uma análise abrangente entre as ferramentas Selenium e Cypress. As características avaliadas incluem execução em sistemas operacionais, navegadores, configuração de projeto, linguagens de programação e diversos outros critérios. O Selenium destacou-se por sua ampla aplicabilidade em diferentes ambientes, enquanto o Cypress apresentou vantagens em termos de escrita de testes e configuração do projeto. A conclusão da pesquisa ressalta que, embora cada ferramenta tenha suas vantagens, o Cypress oferece mais recursos, facilitando o processo de automação de testes E2E para aplicações web.

Outros autores que compararam as ferramentas Cypress e Selenium são Mobaraya e Ali (2019), onde em sua análise compararam as ferramentas nos aspectos usabilidade, desempenho e diversos outros critérios e concluíram que o Cypress, com o suporte adequado, pode ser considerada uma ferramenta promissora no campo da automação de testes, enquanto destacaram a robustez do Selenium respaldada por sua comunidade estabelecida, devido que está no mercado há muitos anos.

Em uma pesquisa similar, conduzida por Ferreira et al. (2022), foram analisadas as ferramentas Cypress, Selenium WebDriver e Robot Framework em termos de usabilidade, facilidade de instalação e desempenho nos testes do lado do usuário. Para obtenção dos resultados, aplicou-se um formulário eletrônico divulgado em redes sociais, contendo perguntas sobre estas caraterísticas para cada ferramenta. Os resultados destacaram a preferência dos desenvolvedores pelo Cypress devido à sua facilidade de uso e instalação.

Já Gruenbaum (2020) compara as ferramentas Selenium, Cypress, Puppeteer e Playwright abordando diversos critérios, como arquitetura, suporte a navegadores, linguagens de programação e paralelismo. Em sua conclusão, destaca que o Selenium é uma escolha sólida, especialmente quando há a necessidade de utilizar linguagens além do JavaScript, dada sua ampla comunidade e suporte. No contexto JavaScript, recomenda a preferência pelo Playwright em relação ao Cypress, devido à sua arquitetura. Além disso, menciona que o Puppeteer, embora seja uma ferramenta utilizada para automação de teste, é fundamentalmente voltada para automação em geral, o que demanda integração com outras ferramentas para a finalidade de testes.

Metin (2023) investigou a eficácia das ferramentas de teste automatizados em um contexto específico de editores de diagramas. Ele explorou a viabilidade das ferramentas Selenium, Cypress e Playwright para abordar os desafios exclusivos desse contexto. Seus critérios de seleção incluíram suporte de linguagem, navegador, protocolo de automação, paralelização, suporte a Electron, recursos e popularidade. Metin (2023) concluiu que o Playwright era a escolha mais adequada para o contexto, oferecendo uma experiência mais acessível e amigável para testadores.

Na Tabela 14 a seguir podemos analisar uma comparação entre os estudos, no que diz respeito ao resultado dos mesmos e a melhor ferramenta a ser utilizada no desenvolvimento de testes automatizados de ponta a ponta em ambientes de aplicações web.

Autoria	**Ferramentas Comparadas**	**Resultado**	**Melhor ferramenta**
Silva (2019)	Selenium e Cypress	A ferramenta Cypress se sai melhor, por possuir mais recursos, que facilitam o processo de teste. Não houve comparação de desempenho.	Cypress
Mobaraya e Ali (2019)	Selenium e Cypress	Destaca as vantagens percebidas do Cypress, mas não declara explicitamente que uma ferramenta é superior à outra. A escolha entre Selenium e Cypress depende das necessidades específicas do projeto e das preferências da equipe de desenvolvimento.	Variável
Ferreira et al. (2022)	Selenium, Cypress e Robot	Destaca a preferência dos desenvolvedores pelo Cypress devido à sua facilidade de uso e instalação.	Cypress
Gruenbaum (2020)	Selenium, Cypress, Puppeteer e Playwright	Deve-se avaliar os requisitos da aplicação para definir a ferramenta. Houve comparação de desempenho, porém aborda brevemente o tema, afirmando que todos são rápidos, mas não aprofunda a análise com dados concretos.	Variável
Metin (2023)	Selenium, Cypress e Playwright	Concluiu que o playwright é a escolha mais adequada para o contexto de editores de diagramas.	Playwright

Tabela 1 – Comparação dos estudos.

Fonte: elaboração própria.

4 Método

Este estudo adota uma abordagem metodológica fundamentada na pesquisa exploratória e descritiva, utilizando o método comparativo e conduzida sob uma perspectiva integrada qualitativa e quantitativa. Essa escolha metodológica busca proporcionar uma compreensão abrangente das ferramentas: Selenium WebDriver, Cypress e Playwright abordadas na seção 2.4. Para alcançar os objetivos propostos, este trabalho baseia-se em dois procedimentos:

- **A revisão da literatura:** Envolve uma análise de trabalhos científicos, documentação oficial das ferramentas, artigos e livros relevantes;
- **Estudo de caso prático:** Utiliza casos de testes cedidos pela empresa FlowUp, proporcionando uma análise prática complementar da revisão da literatura.

A combinação dessas abordagens visa oferecer uma visão abrangente e aprofundada das ferramentas de testes automatizados em ambientes de aplicações web. Através dessa abordagem híbrida, o trabalho busca não apenas compreender as características das ferramentas, mas também avaliar seu desempenho prático em um contexto empresarial real, contribuindo assim para uma análise robusta e significativa.

4.1 Características analisadas

Nesta seção serão abordadas as características escolhidas para avaliar a qualidade das ferramentas de automação de testes, alinhadas com a seção sobre Fatores-Chave para a Qualidade de Software 2.1.1. A análise de cada característica visa compreender a contribuição de cada ferramenta para os atributos definidos pela ISO/IEC 25010 (ISO Central Secretary, 2011).

As características analisadas foram:

- **Linguagem de programação:** Interfere nos fatores de acessibilidade e usabilidade. Linguagens familiares à equipe de desenvolvimento e teste proporcionam uma curva de aprendizado mais suave, permitindo que os desenvolvedores e testadores se adaptem rapidamente às ferramentas de teste;
- **Suporte a navegadores e dispositivos:** Desempenha um papel importante na garantia da compatibilidade, portabilidade e usabilidade do aplicativo em diferentes cenários de ambientes web;

- **Configuração do projeto de automação:** É importante para os fatores de portabilidade e eficiência do processo de teste automatizado, uma vez que algumas ferramentas apresentam dependências ou integrações em relação a outras tecnologias;
- **Documentação da ferramenta e suporte da comunidade:** Imprescindíveis para a usabilidade, garantindo uma melhor experiência de aprendizagem sobre a instalação, configuração, uso e solução de problemas comuns;
- **Recursos de relatórios e registro de testes:** Interferem na manutenibilidade, contribuindo diretamente na facilidade de análise e modificação dos testes, o que permite os desenvolvedores e testadores gerarem relatórios claros e informativos que destaquem seus resultados;
- **Recursos de ferramenta de teste:** Desempenham um papel significativo na melhoria da Experiência do Desenvolvedor (DX), tornando o processo de criação e manutenibilidade de testes mais eficiente e agradável;
- **Depuração dos testes:** Crucial para o fator de manutenibilidade, pois identifica e corrige erros de forma eficaz;
- **Métricas de velocidade:** Contribui para o atributo de eficiência, fornecendo uma visão da rapidez que o teste é executado;
- **Integração com ambientes de CI/CD:** assegura a execução consistente e eficiente de testes automatizados, trazendo benefícios notáveis em termos de eficiência e economia de tempo, além de fortalecer a manutenibilidade do software.

Uma análise mais detalhada da relação entre as características analisadas e os atributos da ISO/IEC 25010 pode ser encontrada no Apêndice A.

4.1.1 Linguagens de programação

A seleção dessas linguagens foi baseada em uma análise das tendências e preferências tecnológicas atuais. Para tal, recorremos aos resultados do Developer Survey (STACK OVERFLOW, 2023), uma pesquisa que obteve respostas de aproximadamente 90 mil profissionais de tecnologia. De acordo com seus resultados relacionados às tecnologias mais populares, foram separadas as linguagens de programação que ocuparam as 5 primeiras posições em termos de utilização, demonstradas na Tabela 2:

Linguagens	Quantidade de usuário que usam a linguagem
JavaScript	55.711
Python	43.158
TypeScript	34.041
Java	26.757
C#	24.193

Tabela 2 – Linguagens mais populares segundo Stack Overflow Survey.

Fonte: elaboração própria.

É importante ressaltar que, embora essas cinco tenham sido identificadas como as mais populares, algumas ferramentas de testes automatizados podem suportar uma variedade de outras linguagens que podem ser adaptadas às preferências da equipe de desenvolvimento e teste. Sendo assim, para a avaliação deste quesito, além das mais populares, este estudo considerou também outras linguagens suportadas e mantidas pelas ferramentas. Mesmo que sejam menos utilizadas, essas linguagens podem oferecer opções valiosas, promovendo uma maior variedade de escolhas para a equipe.

Portanto, as linguagens a serem avaliadas incluem JavaScript, Python, TypeScript, Java, C#, Kotlin, Ruby. Essa abordagem não apenas leva em conta as preferências específicas da equipe, mas também destaca a versatilidade das ferramentas de teste automatizado, tornando-as compatíveis com diversas linguagens e ambientes de desenvolvimento.

4.1.2 Suporte a navegadores

Com base nos dados de participação no mercado de navegadores em todo o mundo no ano de 2023 (GLOBALSTAT, 2023), foram selecionados os navegadores mais populares. A Figura 10 ilustra essa participação

Diante destes resultados, optamos por focar nossos testes nos navegadores Google Chrome, Apple Safari, e Microsoft Edge. O Google Chrome é a escolha principal devido à sua ampla participação no mercado global. O Safari é selecionado para testes em dispositivos iOS e macOS, enquanto o Microsoft Edge é escolhido para garantir a compatibilidade com sistemas Windows.

Para a avaliação deste quesito, este estudo visa validar se as ferramentas de teste automatizado escolhidas proporcionam um suporte eficaz e eficiente para diferentes navegadores, garantindo a qualidade e a adaptabilidade da aplicação sob teste em diversos cenários de uso. A abordagem estratégica na escolha dos navegadores, conforme mencionado na seção de portabilidade, não apenas atende às expectativas dos usuários finais, mas também fortalece a validação da qualidade de código da aplicação, evidenciando a capacidade da ferramenta em lidar com ambientes distintos de maneira consistente e confiável.

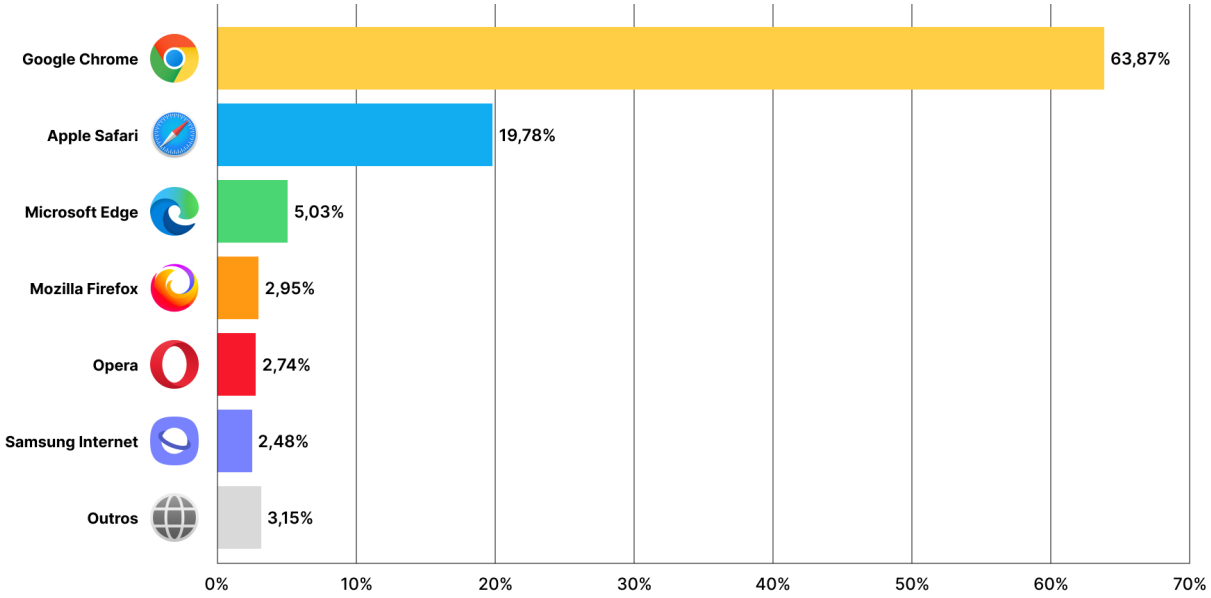

Figura 10 – Participação no mercado de navegadores em todo o mundo no ano de 2023 (GLOBALSTAT, 2023).

Fonte: elaboração própria.

4.1.3 Configuração do projeto de automação

Um dos primeiros desafios da configuração do projeto de automação consiste em avaliar a facilidade com que é possível criar um ambiente de teste funcional usando a ferramenta escolhida. Isso engloba desde a instalação até a configuração inicial e o estabelecimento de um ambiente de teste básico. Configurações complexas e deficientemente documentadas podem ser particularmente desconcertantes para desenvolvedores que estão se familiarizando com a ferramenta e suas tecnologias.

A personalização das configurações é fundamental na adaptação às necessidades específicas de um projeto de automação. É importante avaliar como a ferramenta possibilita a personalização de variáveis de configuração e a definição de comportamentos esperados. Algumas ferramentas ainda oferecem a capacidade de incorporar plugins ou extensões para ampliar suas funcionalidades, um fator que pode ser relevante durante a configuração.

Em última análise, a configuração do projeto de automação deve ser uma etapa que não consuma muito tempo e que seja eficaz. Uma ferramenta eficiente simplifica essa fase, permitindo que o desenvolvedor concentre seus esforços no desenvolvimento de testes automatizados de alta qualidade.

As ferramentas de automação serão comparadas de maneira relativa, analisando a complexidade envolvida na configuração, a dependência de tecnologias adicionais para compor o projeto e a capacidade de personalização para atender às necessidades específicas de cada projeto. Essa análise relativa proporcionará uma compre-

ensão mais completa das vantagens e desvantagens de cada ferramenta no contexto da configuração do projeto de automação.

4.1.4 Documentação da ferramenta e suporte da comunidade

Uma documentação eficaz deve oferecer informações claras sobre a instalação, configuração, uso e solução de problemas comuns. É importante que seja completa, minuciosa e bem estruturada para facilitar e solidificar a experiência de aprendizagem por parte da equipe. Isso engloba a inclusão de exemplos práticos e uma cobertura completa de todos os recursos da ferramenta, apresentados de maneira organizada. A documentação também deve abordar padrões e práticas recomendadas, garantindo que os desenvolvedores sigam diretrizes que otimizem o uso da ferramenta. Não menos importante, a documentação pode sugerir outras fontes úteis de informações, como artigos externos, ferramentas de integração, cursos ou vídeos instrutivos.

Outro elemento valioso na experiência do desenvolvedor é a participação ativa da comunidade, na qual se reflete em fóruns de discussão, como o Stack Overflow, e no envolvimento em projetos de código aberto relacionados à ferramenta. Uma comunidade ativa permite que os usuários compartilhem suas experiências, colaborem na resolução de problemas e contribuam para o aprimoramento contínuo da ferramenta.

A avaliação desta pesquisa utiliza a documentação oficial de cada ferramenta, juntamente com dados provenientes da plataforma Stack Overflow, para avaliar os seguintes aspectos:

- Clareza da documentação;
- Padrões e Práticas Recomendadas;
- Integração com outras ferramentas;
- Fontes adicionais de conhecimento;
- Participação da Comunidade.

4.1.5 Recursos de relatórios e registro de testes

Um dos principais critérios ao avaliar ferramentas de automação de testes é a robustez de seus recursos de relatórios. Isso inclui informações sobre testes bem-sucedidos e falhados, tempo de execução, registros de erros detalhados e métricas de cobertura de teste.

As ferramentas de automação de testes geralmente oferecem uma gama de recursos nesse aspecto, proporcionando diversos artefatos que facilitam a análise de

erros. Dado que as falhas podem ocorrer por uma variedade de motivos, é importante que os relatórios detalhem minuciosamente as informações sobre os erros encontrados.

Um relatório eficaz de ferramentas de testes de ponta-a-ponta deve incluir informações sobre os resultados dos testes após a execução, a identificação das causas das falhas nos testes, capturas de tela da interface após a execução dos testes e gravações em vídeo que demonstram a execução completa dos testes. A avaliação das características dessas ferramentas será realizada de forma comparativa, com ênfase nos recursos oferecidos pelos relatórios de erros de cada uma delas.

4.1.6 Recursos de ferramenta de teste

Ao avaliar ferramentas de automação de testes, é fundamental considerar os recursos oferecidos por cada uma delas, uma vez que interferem significativamente na Experiência do Desenvolvedor (DX). Esta característica de avaliação abrange diversos recursos essenciais que impactam significativamente a DX ao criar testes automatizados. Durante a análise das ferramentas de automação de testes, foram considerados os seguintes atributos:

- **Interface de Execução de Testes:** Ela facilita a criação, execução e organização de casos de teste, permite a seleção de configurações de execução e proporciona um acompanhamento eficaz dos resultados. Além disso, uma interface bem projetada acelera a identificação de falhas e a torna mais eficiente;
- **Recarregamento rápido:** Este recurso é um aliado essencial no desenvolvimento de testes ágeis, pois contribui para a eficiência no desempenho da escrita de testes, permitindo que os desenvolvedores vejam imediatamente o impacto das alterações no código. Essa capacidade de atualizar automaticamente os testes em tempo real economiza tempo e aumenta a produtividade, garantindo que as mudanças no código sejam refletidas instantaneamente;
- **Depuração de viagem no tempo:** é uma poderosa ferramenta de análise de testes, já que permite a execução em diferentes pontos da aplicação, possibilitando a análise de comportamentos passados e futuros. Isso é particularmente útil para depurar problemas complexos e entender o fluxo de execução dos testes em detalhes;
- **Automação da Espera pela Disponibilidade de Elementos:** Este recurso elimina a necessidade de adicionar pausas ou temporizações manuais nos testes, tornando-os mais confiáveis e eficientes. Isso é especialmente importante em testes de aplicações web, onde a espera por elementos é comum;

- **Seletor de Elementos Simplificado:** A seleção de elementos pode ser difícil, por isso algumas ferramentas de testes automatizados possuem um Seletor simplificado, facilitando a captura dos identificadores dos elementos visuais, que são importantes para que seja possível interagir com os mesmos através dos comandos de teste;
- **Simulação de Requisições Web:** Por sistemas complexos poderem ter integrações com serviços de terceiros como APIs, a simulação de requisições web se torna um ponto importantíssimo, permitindo que os testes sejam executados de forma consistente, mesmo quando os recursos externos não estão disponíveis;
- **Lida com Múltiplas Abas e navegadores ao mesmo tempo:** importante para projetos mais complexos, este recurso possibilita lidar com testes que envolvam mais de uma página (sejam abas ou navegadores), sendo essencial para simular cenários reais de uso onde os usuários interagem com várias partes de um aplicativo web ao mesmo tempo. A habilidade de lidar com concorrência e múltiplas guias é fundamental para testes robustos e eficazes;
- **Execução Paralela:** Para aprimorar a experiência de desenvolvimento, ferramentas de automação de testes que oferecem suporte para a execução paralela podem reduzir o tempo total de execução, além disso permite que diversos casos de teste sejam executados em diferentes configurações de ambiente ao mesmo tempo;
- **Geração Visual de Testes:** Ajuda na escrita de testes de uma maneira visual, gravando interações do usuário no aplicativo em teste. Isso não apenas economiza tempo na criação manual de testes, mas também é útil em projetos de grande escala, onde a interação do usuário pode ser complexa e diversificada, e a criação manual de casos de teste seria demorada e propensa a erros;
- **Reaproveitamento de Código:** é importante para seguir as boas práticas de programação, evitando a repetição de código nos testes, tornando a manutenção mais fácil e economizando tempo no desenvolvimento de novos casos de teste.

4.1.7 Depuração dos testes

Durante a criação e execução de testes automatizados, diversos tipos de erros podem ocorrer. Isso inclui erros de sintaxe, erros de lógica, erros de elemento não encontrado, erros de tempo de espera excedido, erros de configuração, problemas na massa de dados e muito mais. Sendo assim, a depuração de testes é crucial para identificar e corrigir esses erros de forma eficaz.

Uma ferramenta de automação de testes deve proporcionar uma interface de depuração intuitiva que permita aos desenvolvedores e testadores identificarem rapidamente os erros. Isso envolve recursos como realce de código para destacar áreas problemáticas, visualização da pilha de chamadas para rastrear a origem dos erros e mensagens de erro descritivas para uma compreensão clara dos problemas identificados.

Mais do que identificar os erros, uma ferramenta eficaz de automação de testes pode ir além e oferecer sugestões de solução. Isso engloba recomendações sobre como abordar e corrigir o erro, sugestões de aprimoramento de código e possíveis ajustes nas ações de teste, a fim de evitar recorrências de problemas similares no futuro. Outro aspecto relevante é a fácil integração com as IDEs (Integrated Development Environments) usadas pela equipe. Isso simplifica o processo de depuração, permitindo que os desenvolvedores acessem as ferramentas de depuração diretamente de sua IDE de escolha.

Nesta avaliação, será considerado se a ferramenta de automação de testes oferece recursos robustos para depuração e se a integração com as IDEs utilizadas pela equipe é possível e eficaz, o que contribui para um ambiente de desenvolvimento de testes mais eficiente com maior manutenibilidade.

4.1.8 Métricas de velocidade

As métricas de velocidade no desenvolvimento de software podem assumir várias formas, dependendo dos objetivos específicos do projeto e das áreas a serem avaliadas. Uma categoria importante de métricas de velocidade é o tempo de execução de testes, que fornece informações críticas sobre o desempenho e a eficiência dos processos de teste de software. Por exemplo, se um teste específico está levando muito tempo para ser executado, a equipe pode decidir se deve priorizá-lo de maneira diferente. Dentro do contexto das métricas de tempo de execução de testes, existem algumas métricas como:

- **Média de Tempo de Execução:** fornece uma visão geral do tempo médio que um conjunto de testes leva para ser concluído. Isso ajuda a equipe a entender o tempo necessário para testar uma determinada funcionalidade ou componente;
- **Desvio Padrão do Tempo de Execução:** é uma medida de dispersão que indica a variabilidade dos tempos de execução. Um alto desvio padrão pode indicar inconsistências nos processos de teste;
- **Coeficiente de Variação (CV):** é calculado como a razão entre o desvio padrão e a média dos tempos de execução. Mostrando a variabilidade dos resultados

em relação à média;

- **Percentil 95 (P95):** O valor mais alto restante quando os 5% superiores de um conjunto de dados coletados numericamente classificados são descartados. Interessante para entender como poderia ser um ponto de dados não extremo, mas ainda alto.

Para avaliar essas métricas, foram seguidas as Diretrizes Gerais estabelecidas na seção 4.2. A partir desses resultados, serão consideradas métricas como a média de tempo de execução, seu desvio padrão, coeficiente de variação e percentil 95.

4.1.9 Integração com ambientes de CI/CD

Ao avaliar as ferramentas de automação de testes, um dos critérios importantes é determinar se a ferramenta possui a capacidade de integração perfeita com ambientes de integração contínua e entrega contínua (CI/CD). Essa integração tem o objetivo de garantir que os testes automatizados sejam executados de maneira consistente e eficiente. O objetivo principal é verificar se as alterações no código fonte introduziram erros, permitindo a identificação e correção de problemas de forma proativa, antes que eles impactem o ambiente de produção.

A integração eficaz com ambientes de CI/CD oferece uma série de benefícios notáveis, particularmente em termos de eficiência e economia de tempo. Ao incorporar a automação de testes na esteira de CI/CD, as equipes podem significativamente reduzir o tempo gasto em tarefas manuais e garantir que o software seja entregue de forma mais rápida e consistente.

Uma integração sólida contribui para estabelecer uma prática de integração, entrega e implantação mais confiável e eficiente, assegurando que todas as mudanças no código sejam verificadas e validadas automaticamente. O resultado final é uma entrega mais rápida e de maior qualidade para o cliente, proporcionando um ambiente de desenvolvimento ágil e eficaz. A avaliação neste aspecto se concentra na habilidade da ferramenta de automação de testes de se integrar harmoniosamente à prática de CI/CD.

4.2 Diretrizes gerais do estudo de caso

Nesta seção serão abordadas as diretrizes gerais para garantir a consistência, comparabilidade e confiabilidade dos resultados obtidos no estudo de caso.

4.2.1 Cenário de teste

A empresa disponibilizou um arquivo confidencial no formato *Behavior Driven Development* (BDD), contendo um cenário com um conjunto de casos de teste, descritos na Tabela 3, para uma funcionalidade de login. Este cenário de teste abrange tanto o fluxo ideal quanto os cenários de erro, proporcionando uma cobertura abrangente das possíveis interações do usuário com a funcionalidade. Para esse cenário, os script foram implementados conforme as especificações detalhadas no arquivo BDD e foram coletados dados a partir de 100 execuções sequenciais bem-sucedidas do mesmo script.

Descrição do caso de teste
Login - Email e senha inválidos
Login - Email inválido
Login - Senha inválida
Login com sucesso
Login de um usuário com perfil Colaborador
Login de um usuário com perfil Gerente
Login de um usuário com perfil Comercial - Acesso ao Cash
Login de um usuário com perfil Comercial - Acesso ao Task
Login de um usuário com perfil Assistente Administrativo - Acesso ao Cash
Login de um usuário com perfil Assistente Administrativo - Acesso ao Task

Tabela 3 – Casos de teste do cenário de Login do FlowUp.

Fonte: elaboração própria.

4.2.2 Benchmark

No contexto dos testes de métricas de velocidade abordados na seção 4.1.8, as seguintes diretrizes foram adotadas para assegurar a consistência nas condições de avaliação

- **Paridade de recursos:** Cada teste foi executado sequencialmente na mesma máquina enquanto ela estava “em repouso”, ou seja, nenhuma carga de trabalho pesada estava ocorrendo em segundo plano durante o benchmark nem interferindo potencialmente nas medições;
- **Execução simples:** Os scripts foram executados conforme indicado na documentação, com configurações mínimas adicionais. Essa abordagem simplificada buscou preservar a integridade e a fidelidade dos resultados, minimizando alterações não essenciais nas configurações padrão;
- **Scripts comparáveis:** As diferenças nos scripts são mínimas. Ainda assim, algumas instruções tiveram que ser adicionadas, removidas ou ajustadas para alcançar uma execução estável;

- **Versões mais recentes:** Todos os testes foram conduzidos utilizando as versões mais recentes das ferramentas disponíveis no momento da publicação deste trabalho. Isso assegurou que estávamos avaliando as últimas melhorias e otimizações implementadas pelas ferramentas;
- **Mesmo navegador:** Todos os scripts foram executados no ambiente Chromium Headless[1], garantindo uniformidade nas condições do navegador para todos os testes realizados.

4.2.3 Configuração técnica

Todos os testes foram realizados em um desktop, com as seguintes especificações:

- **Sistema Operacional:** Windows 11 PRO com WSL(Windows Subsystem for Linux) utilizando a Distro Ubuntu 22.04;
- **Processador:** Intel Core i7 14700K:
  - **Frequência máxima:** 5.6 GHz;
  - **Frequência base dos P-Cores (cores de performance):** 3.4 GHz;
  - **Frequência base dos E-Cores (cores de eficiência):** 2.5 GHz;
  - **Cache:** 33 MB;
  - **Cache L2:** 28MB;
- **Memória:** 32 GB DDR5 com frequência 6200 MHz;

4.2.4 Bibliotecas utilizadas

Para atender às diretrizes estabelecidas na seção de benchmark 4.2.2, desenvolvemos os scripts utilizando JavaScript. A escolha dessa linguagem foi respaldada pelo fato de ser a mais amplamente reconhecida, conforme indicado pela pesquisa Stack Overflow Survey. Além disso, destaca-se sua compatibilidade com todas as ferramentas em análise: Selenium WebDriver, Cypress e Playwright. Para viabilizar o uso de JavaScript, optamos pelo ambiente Node.js na versão 20.11.1, que inclui o gerenciador de pacotes npm na versão 10.2.4. Essa decisão proporciona um ambiente de execução eficaz para o JavaScript fora do navegador, promovendo consistência e facilidade de desenvolvimento.

No que diz respeito aos requisitos de configuração mínima, adotamos as seguintes bibliotecas e suas respectivas versões para cada ferramenta:

[1] Versão sem interface gráfica do navegador.

- **Selenium WebDriver:**
  - **chromedriver:** 122.0.3;
  - **mocha:** 10.0.3;
  - **selenium-webdriver:** 4.18.1;
- **Cypress:**
  - **cypress:** 13.6.6;
- **Playwright:**
  - **@playwright/test:** 1.41.2;

5 Resultados e discussões

Nesta seção são realizados os métodos descritos na seção 4, a fim de validar as ferramentas selecionadas de acordo com os critérios das características apresentadas na seção 4.1. Para tal, foram utilizadas como fonte de informação majoritária as documentações oficiais das ferramentas Selenium (2024), Cypress (2024) e Playwright (2024). Em seguida, serão apresentados os resultados do estudo de caso, cuja análise se utilizou das diretrizes já explicadas na seção 4.2.

5.1 Linguagens de programação

Ao analisarmos a Tabela 4, é possível observar que o Selenium WebDriver possui uma vantagem significativa em relação às outras ferramentas, oferecendo suporte a uma variedade maior de linguagens de programação para a construção de testes automatizados.

	Selenium WebDriver	**Cypress**	**Playwright**
JavaScript	SIM	SIM	SIM
TypeScript	SIM	SIM	SIM
Python	SIM	NÃO	SIM
C#	SIM	NÃO	SIM
Java	SIM	NÃO	SIM
Kotlin	SIM	NÃO	NÃO
Ruby	SIM	NÃO	NÃO

Tabela 4 – Comparação entre as linguagens de programação de escrita dos testes.

Fonte: elaboração própria.

É importante ressaltar que, embora a Tabela forneça uma visão inicial das linguagens oficialmente suportadas e mantidas pelas ferramentas, o Selenium WebDriver vai além, permitindo a incorporação de outras linguagens por meio de pacotes mantidos pela comunidade. Esses pacotes permitem a escrita de testes em Go, Haskell, Perl, PHP, R e Dart.

O Playwright também oferece um excelente suporte para diversas linguagens. Por outro lado, o Cypress se apresenta com uma quantidade bem restrita de linguagens, e isso é devido a sua arquitetura de alto nível, que executa os scripts de teste diretamente no navegador, que compreende apenas JavaScript. Portanto, o Cypress restringe-se à escrita de testes em JavaScript e TypeScript, sendo necessário transpilar este último para JavaScript antes da execução.

5.2 Suporte a navegadores

Ao examinarmos a Tabela 5, fica evidente que todas as três ferramentas analisadas oferecem um sólido suporte aos principais navegadores em análise.

	Selenium WebDriver	**Cypress**	**Playwright**
Chrome	SIM	SIM	SIM
Safari	SIM	SIM	SIM
Edge	SIM	SIM	SIM
Firefox	SIM	SIM	SIM

Tabela 5 – Comparação de uso entre os principais navegadores.

Fonte: elaboração própria.

Assim, conclui-se que o suporte abrangente a navegadores é uma característica sólida e consistente nas ferramentas de automação de testes em análise, garantindo a eficácia dos testes em ambientes diversificados.

5.3 Configuração do projeto de automação

A respeito desta seção, o Selenium WebDriver exige configuração inicial mais complexa, incluindo a instalação de outras ferramentas como Test Runners (gerenciadores de testes), ferramenta de verificação e drivers específicos para navegadores. Além disso, a documentação nesta parte carece de exemplos, fazendo com que seja necessário a busca de outros recursos para obter informações sobre estes componentes fundamentais para a efetiva utilização da ferramenta em testes automatizados. Devido à sua dependência de outras ferramentas, a personalização acaba sendo mais voltada para a ferramenta integrada do que para o Selenium em si. Apesar dessa lacuna documental, o Selenium possui uma página dedicada ao ecossistema, destacando diversos frameworks e linguagens que a ferramenta suporta, muitos dos quais já incorporam as ferramentas necessárias para a execução dos testes, facilitando a configuração inicial.

Por outro lado, o Cypress oferece uma configuração inicial facilitada, uma vez que já está integrado com as ferramentas essenciais para testes, demandando menos configuração e permitindo a execução rápida de testes. A ferramenta utiliza, por padrão, os navegadores instalados no sistema operacional, embora seja necessário baixar e instalar navegadores adicionais caso seja necessário testar em um navegador não presente na máquina. A documentação abrangente do Cypress explica cada opção de configuração de forma detalhada, mostrando para que serve cada ajuste no arquivo de configuração. Essa abordagem simplifica a personalização, possibilitando modificações diretas no arquivo ou através da interface gráfica da ferramenta em tempo

real. Adicionalmente, o Cypress oferece uma página dedicada para sua integração com outras ferramentas, facilitando a resolução de casos de teste mais complexos.

Já o Playwright possui uma configuração inicial extremamente ágil, onde diferentemente do Cypress, além de instalar a da ferramenta, instala os navegadores e as dependências que a ferramenta possui do sistema operacional da máquina utilizada para testar. Assim como o Cypress, o Playwright apresenta uma documentação detalhada e um arquivo de personalização que compreende todas as configurações disponíveis na ferramenta. No que diz respeito à integração com outras ferramentas, a documentação do Playwright destaca que a ferramenta já está integrada a diversas funcionalidades, como comandos que facilitam a seleção de elementos no navegador, relatórios que são utilizados para exibir um resumo após a execução dos testes, e ainda permite integrações personalizadas de forma simples e bem documentada.

Considerando essas informações, também destacadas na Tabela 6, onde a preferência recai sobre soluções com menor necessidade de instalação de ferramentas, o Playwright emerge como a ferramenta que oferece a melhor experiência para um desenvolvimento ágil, proporcionando uma instalação completa e eficiente de todos os elementos necessários para seu funcionamento. Além disso, a documentação abrangente e a facilidade de integração contribuem para uma experiência de automação robusta e eficaz.

	Selenium WebDriver	**Cypress**	**Playwright**
Depende da instalação de drivers	SIM	NÃO	NÃO
Depende da instalação de ferramentas de execução de teste	SIM	NÃO	NÃO
Depende da instalação de ferramentas de validação de elementos	SIM	NÃO	NÃO
Depende da instalação de navegadores	SIM	SIM	NÃO
Depende da instalação de dependências do sistema operacional	SIM	SIM	NÃO

Tabela 6 – Comparação da configuração do projeto de automação das ferramentas.

Fonte: elaboração própria.

5.4 Documentação da ferramenta e suporte da comunidade

5.4.1 Clareza da documentação

Quando se trata da clareza da documentação, é importante notar que esta avaliação é subjetiva e reflete a opinião do autor. No que diz respeito à documentação do Selenium WebDriver, observa-se que ela destaca uma visão das funcionalidades de automação, oferecendo exemplos em algumas linguagens. No entanto, carece de informações para integração com outras ferramentas, principalmente quando se trata

de automação de testes. Por outro lado, tanto o Cypress quanto o Playwright apresentam documentações com mais informações em todas as etapas do processo, desde a instalação e configuração até a resolução de problemas comuns. Ambas as ferramentas destacam-se por apresentar exemplos práticos que ilustram cada comando. Além disso, a documentação do Playwright abrange múltiplas linguagens de programação.

5.4.2 Padrões e Práticas Recomendadas

O Selenium possui em sua documentação uma seção específica que aborda diretrizes e recomendações. No entanto, as orientações são mais voltadas para boas práticas de teste de software do que boas práticas de uso da ferramenta. Se diferenciando do Selenium WebDriver, tanto o Cypress quanto o Playwright, possuem seções para diversos exemplos de boas práticas elucidando o que fazer e o que evitar, demonstrando os impactos de seguir ou não seguir as práticas recomendadas.

5.4.3 Integração

O Selenium dedica uma seção ao ecossistema da ferramenta, porém, quando se trata de integração com ferramentas que agregam recursos de escrita de teste, a documentação carece de informações específicas e, em alguns casos, não oferece orientações claras. O que leva a busca por essas informações em outras fontes.

Em contrapartida, o Cypress apresenta uma documentação abrangente sobre integrações, fornecendo exemplos práticos e orientações claras, sendo útil em projetos que possam necessitar de integração com outras ferramentas.

Quanto ao Playwright, embora apresente algumas lacunas em relação à integração com outras ferramentas, destaca-se por suas funcionalidades nativas, reduzindo a necessidade de adicionar e configurar recursos adicionais. No entanto, caso seja necessário incorporar novas ferramentas para suprir as funcionalidades nativas, é importante estar ciente da necessidade de buscar essas informações em outras fontes.

5.4.4 Fontes adicionais de conhecimento

Além da documentação oficial, as ferramentas de automação oferecem uma variedade de fontes adicionais de conhecimento para auxiliar os desenvolvedores em diferentes aspectos do uso da ferramenta e na resolução de dúvidas, como podemos ver na Tabela 7.

	Selenium WebDriver	**Cypress**	**Playwright**
Github	SIM	SIM	SIM
Youtube	SIM	SIM	SIM
Comunidade oficial	SIM	SIM	SIM
Perfil no Twitter	SIM	SIM	SIM
Blog oficial	SIM	SIM	SIM
Curso gratuito oficial	NÃO	SIM	NÃO
Suporte personalizado pago	NÃO	SIM	NÃO

Tabela 7 – Comparação das fontes adicionais de conhecimento das ferramentas.

Fonte: elaboração própria.

Os GitHub de cada ferramenta são plataformas abertas que possibilitam aos desenvolvedores relatar bugs, solicitar novas funcionalidades e contribuir para o aprimoramento contínuo da ferramenta. Paralelamente os canais no YouTube das ferramentas oferecem uma ampla gama de recursos, incluindo vídeos sobre funcionalidades, palestras em conferências, transmissões ao vivo e listas de vídeos recomendados feitos por terceiros. Esses canais são valiosos para aprofundar o conhecimento sobre as ferramentas e suas práticas recomendadas.

Outra opção são as comunidades, que servem como espaços interativos onde desenvolvedores podem se conectar, trocar experiências e obter respostas para suas dúvidas. Tanto a comunidade quanto os perfis no Twitter fornecem atualizações regulares sobre as últimas notícias e atualizações das ferramentas.

As três ferramentas mantêm blogs oficiais, que oferecem conteúdos mais detalhados e informativos sobre o uso e as melhores práticas das ferramentas.

Destacando-se entre as ferramentas, o Cypress oferece cursos gratuitos, desenvolvidos pela equipe da ferramenta, e dedicados a mostrar testes no mundo real. Estes cursos utilizam vídeos, documentação em texto, questões e exemplos de diversos cenários de testes, proporcionando uma experiência educacional abrangente. Além disso, para aqueles que buscam assistência mais detalhada, o Cypress oferece suporte personalizado pago, garantindo resolução ágil de dúvidas e problemas específicos.

5.4.5 Participação da Comunidade

Ao explorar as comunidades em torno das ferramentas de automação, além do suporte oferecido pelas bibliotecas, é valioso analisar dados provenientes do STACK OVERFLOW (2024a), uma plataforma amplamente utilizada pelos desenvolvedores para compartilhar conhecimentos e buscar soluções para desafios específicos. Abaixo, na Tabela 8, são apresentadas métricas que revelam a participação da comunidade, relacionadas aos marcadores no Stack Overflow: “selenium-webdriver” para o Selenium

WebDriver, “cypress” para o Cypress e “playwright” para o Playwright.

	selenium-webdriver	cypress	playwright
Total de perguntas sem respostas votadas ou aceitas	64.467	3.438	1.393
Total de Perguntas	157.615	9.853	2.802

Tabela 8 – Comparação das fontes adicionais de conhecimento das ferramentas.

Fonte: elaboração própria.

O Selenium destaca-se com a maior quantidade absoluta de perguntas, é um reflexo de sua ampla adoção e uso extensivo na comunidade de desenvolvimento. Por outro lado, o Cypress, embora apresente uma quantidade menor de perguntas em comparação ao Selenium, mantém uma posição intermediária entre as três ferramentas. Entretanto, ao analisar o gráfico a seguir sobre a porcentagem de perguntas no Stack Overflow por mês na Figura 11, observamos um declínio no número de perguntas com o marcador “cypress” desde o final de 2023. Esse declínio pode indicar uma possível estabilização no uso da ferramenta ou a resolução eficaz de problemas comuns ao longo do tempo ou também a queda no uso da ferramenta.

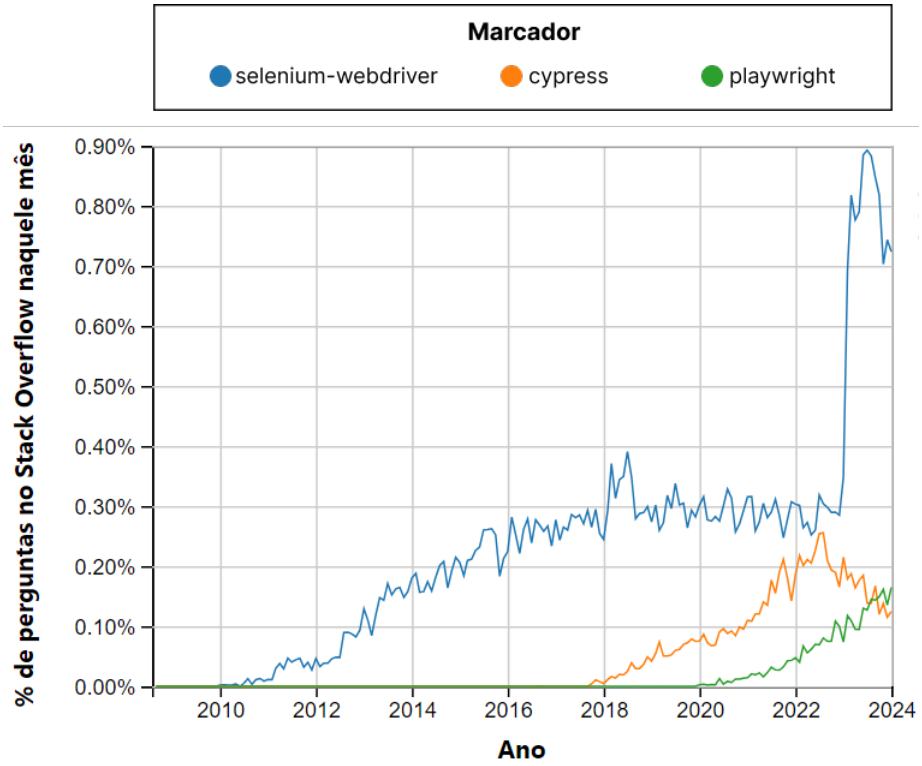

Figura 11 – Variação de Perguntas no Stack Overflow dos marcadores selenium-webdriver, cypress e playwright.

Fonte: traduzida de STACK OVERFLOW (2024b).

Já o Playwright, inicialmente com a menor quantidade de perguntas, apresenta uma trajetória ascendente, este é um comportamento esperado, devido a ser uma fer-

ramenta nova. Desde o final de 2023, observa-se um aumento nas perguntas relacionadas ao Playwright, indicando um crescimento na adoção e interesse da comunidade em explorar e compreender essa ferramenta.

Portanto, considerando tanto a documentação da ferramenta quanto o suporte da comunidade, ocorre um empate entre as 3 ferramentas. O Selenium destaca-se por ter uma comunidade ativa, o que facilita a resolução de problemas, mesmo diante de uma documentação menos robusta. Por outro lado, o Cypress oferece a melhor documentação das três ferramentas, apesar de não possuir uma comunidade tão grande quanto a do Selenium WebDriver. Por fim, o Playwright possui uma documentação detalhada e direta, e apesar de ter uma comunidade menor em comparação com as outras duas ferramentas, está em ascensão, o que pode indicar um crescente interesse e adoção por parte dos desenvolvedores.

5.5 Recursos de relatórios e registro de testes

O Selenium WebDriver, por si só, oferece recursos para registrar resultados de testes, incluindo informações sobre testes bem-sucedidos, falhados e tempo de execução. No entanto, a profundidade da informação e o formato exato dos relatórios dependem da configuração feita pelos desenvolvedores, muitas vezes exigindo integração com outras ferramentas de teste para organizar os resultados. Isso inclui métricas de cobertura de teste e informações detalhadas de validação e verificação para obter uma visão abrangente e facilitar a identificação e correção de problemas.

Em contrapartida como podemos ver na Tabela 9, tanto o Cypress quanto o Playwright apresentam similaridades nos recursos de relatórios e registros de testes. Ambas as ferramentas por padrão, já oferecem detalhes abrangentes dos resultados dos testes, destacando aspectos como tempo de execução, testes completados e falhas. Além disso, ambas podem incorporar métricas de cobertura de teste nos relatórios, gerar capturas de tela após a execução dos testes e suportar gravações em vídeo, elementos essenciais para compreender a dinâmica da aplicação durante os testes.

	Selenium WebDriver	**Cypress**	**Playwright**
Informações de sucesso, falha e tempo de execução	SIM	SIM	SIM
Geração de relatórios	NÃO	SIM	SIM
Capturas de tela	NÃO	SIM	SIM
Gravação em vídeo	NÃO	SIM	SIM

Tabela 9 – Comparação dos recursos de relatórios e registro de testes das ferramentas.

Fonte: elaboração própria.

No que diz respeito à facilidade de análise e modificação, tanto o Cypress quanto o Playwright se sobressaem, proporcionando interfaces amigáveis e documentação detalhada que simplificam a compreensão e personalização dos relatórios gerados.

5.6 Recursos de ferramenta de teste

5.6.1 Interface de Execução de Testes

O Selenium WebDriver carece de uma interface gráfica dedicada para a execução de testes, o que pode ser interpretado como uma limitação em relação ao critério de Usabilidade. Embora possua a capacidade de abrir o navegador durante a execução, a ausência de uma interface específica pode dificultar a reprodução e o acompanhamento detalhado dos testes, afetando a eficiência e a experiência do desenvolvedor. Para suprir essa lacuna pode ser utilizada outra ferramenta como o Selenium IDE que funciona apenas em alguns navegadores.

Em contrapartida, tanto o Cypress quanto o Playwright destacam-se ao oferecer interfaces gráficas dedicadas nativamente, como evidenciado nas Figuras 12 e 13, respectivamente. Essa abordagem proporciona uma experiência mais intuitiva para desenvolvedores e testadores. Essas interfaces não apenas facilitam a execução dos testes, mas também contribuem para uma organização mais eficiente, endereçando aspectos essenciais desta seção.

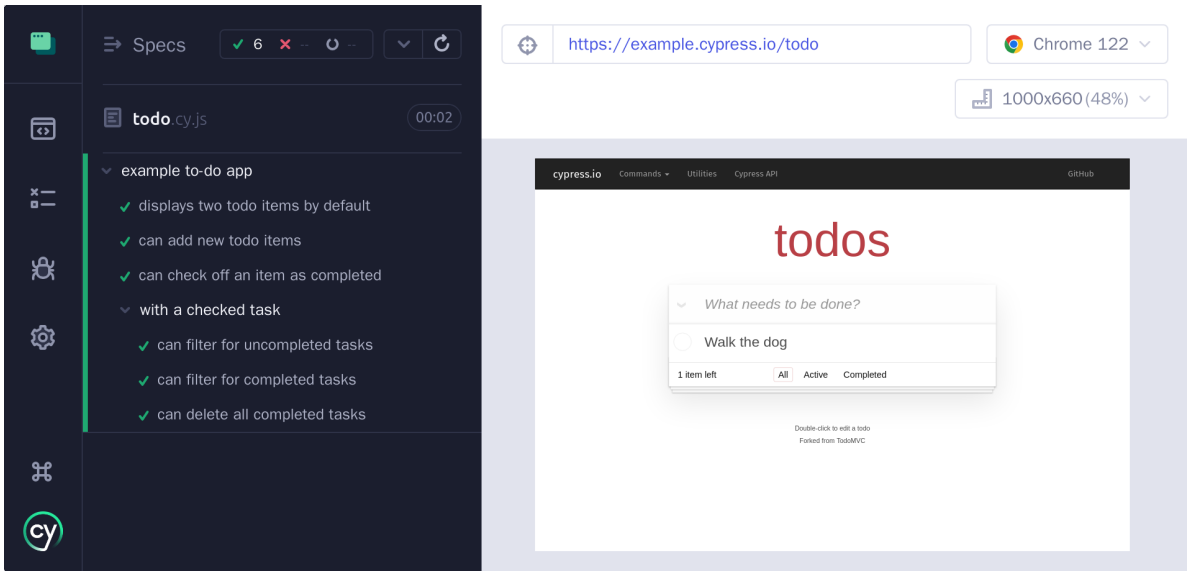

Figura 12 – Interface visual do Cypress.

Fonte: elaboração própria.

Ao considerar a escolha entre Cypress e Playwright, em relação à interface de execução de testes, ambas as ferramentas oferecem vantagens em comparação com

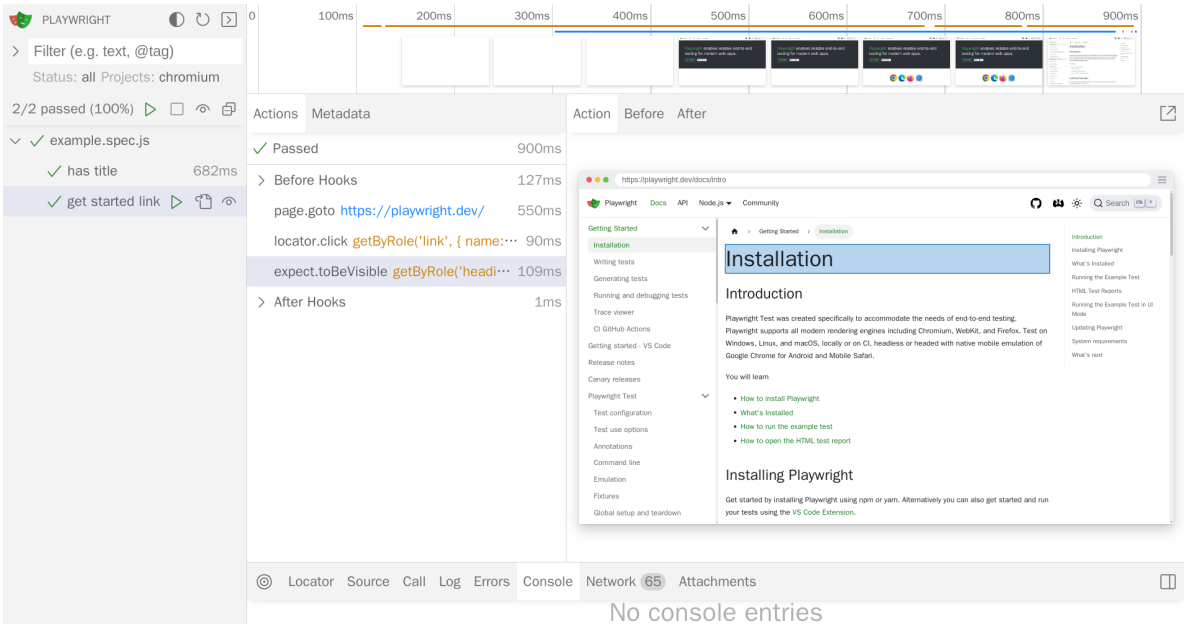

Figura 13 – Interface visual do Playwright.

Fonte: elaboração própria.

o Selenium WebDriver. Contudo, a decisão final dependerá da harmonização desses recursos com as necessidades específicas do projeto.

5.6.2 Recarregamento rápido

O Selenium WebDriver apresenta uma lacuna notável ao não oferecer um recurso de recarregamento rápido integrado, e a ausência de informações sobre esse recurso na documentação da ferramenta reflete uma limitação no quesito Manutenibilidade.

Por outro lado, tanto o Cypress quanto o Playwright se destacam ao fornecer suporte para o recurso. Essa funcionalidade, alinhada ao princípio de “Eficiência no Desenvolvimento” (Seção 2.1.1), permite aos desenvolvedores visualizarem instantaneamente o impacto das alterações no código, acelerando o processo de escrita de testes.

Neste contexto, ocorre um empate entre Cypress e Playwright em relação ao recarregamento rápido, tendo em vista que ambas as ferramentas oferecem essa funcionalidade, proporcionando uma experiência de desenvolvimento mais ágil e eficiente.

5.6.3 Depuração de viagem no tempo

No caso do Selenium WebDriver, a ausência de um recurso específico para depuração de viagem no tempo, aliada à falta de informações na documentação, sugere uma limitação em diversos quesitos abordados na seção Fatores-chave para qualidade

de código 2.1.1. Por outro lado, tanto o Cypress quanto o Playwright empatam neste quesito, pois se destacam ao oferecer esse recurso o que contribui para uma análise detalhada e eficiente do fluxo de execução dos testes, promovendo a qualidade e a compreensão dos resultados.

5.6.4 Automação da Espera pela Disponibilidade de Elementos

O Selenium WebDriver não apresenta uma funcionalidade nativa de espera automática por elementos, o que implica a necessidade de configurações adicionais para automatizar esse aspecto. Isso pode impactar a eficiência do processo de escrita de testes, especialmente em cenários dinâmicos.

Por sua vez, tanto o Cypress quanto o Playwright destacam-se ao oferecer mecanismos integrados de espera automática. Além da espera automática, também disponibilizam comandos de espera mais específicos para casos particulares de teste, ampliando sua flexibilidade em situações que demandam abordagens mais personalizadas.

Em suma, há um empate entre Cypress e Playwright nesse aspecto, pois ambas as ferramentas oferecem automação de espera pela disponibilidade de elementos. Essa característica contribui para a confiabilidade e eficiência dos testes automatizados, promovendo a qualidade do software.

5.6.5 Seletor de Elementos Simplificado

O Selenium WebDriver, demanda seletores mais complexos, não oferecendo uma ferramenta específica para facilitar a seleção de elementos. Esse processo no Selenium envolve interações diretas no navegador, exigindo conhecimento avançado e experiência por parte dos desenvolvedores de testes.

Em contrapartida, tanto o Cypress quanto o Playwright proporcionam uma funcionalidade dedicada para simplificar a seleção de elementos pela interface gráfica dos testes. Além de que ambas as ferramentas oferecem comandos que tornam a seleção mais intuitiva e semântica, possibilitando, por exemplo, a busca por elementos com base em seu papel específico e até mesmo por expressões regulares, simplificando significativamente o processo. Essa abordagem contribui com a usabilidade, eficiência e manutenção no processo de escrita de testes automatizados, havendo assim um empate nesta seção entre as ferramentas Cypress e Playwright.

5.6.6 Simulação de Requisições Web

Por padrão, o Selenium WebDriver não oferece um mecanismo nativo para a interceptação de requisições web, e a ausência de informações sobre esse recurso

em sua documentação sugere a necessidade de soluções adicionais para simular interações com a web.

Já o Cypress e o Playwright destacam-se ao proporcionar recursos integrados para a simulação de requisições web. Essas ferramentas oferecem mecanismos que permitem aos desenvolvedores de testes simular interações com serviços externos de maneira controlada e previsível. Nesse sentido, podemos considerar um empate entre Cypress e Playwright, tendo em vista que ambas possuem essa funcionalidade, contribuindo para a Adequação Funcional e Confiabilidade.

5.6.7 Lida com Múltiplas Abas e navegadores ao mesmo tempo

Tanto o Selenium quanto o Playwright destacam-se por permitir a execução de testes em múltiplas abas e navegadores simultaneamente, proporcionando flexibilidade e versatilidade na simulação de cenários complexos de interação com o usuário.

Porém, o Cypress, devido à sua arquitetura específica, não permite essa mesma execução das outras 2 ferramentas, e isso acaba limitando a capacidade de simular cenários que envolvem interações em várias partes de uma aplicação web ao mesmo tempo.

Considerando os critérios de Facilidade de uso e Eficiência, o Selenium e o Playwright empatam nesta categoria.

5.6.8 Execução Paralela

Ao analisar os recursos relacionados à “Execução Paralela” no contexto da escrita de testes, observamos diferenças significativas entre o Selenium WebDriver, o Cypress e o Playwright, cada um apresentando abordagens distintas para atender a essa demanda.

Por padrão, o Selenium WebDriver não oferece funcionalidades nativas de paralelismo. No entanto, é possível implementar a execução paralela usando o Selenium Grid ou outros frameworks externos, o que acaba demandando um esforço adicional de configuração e gerenciamento. Por outro lado, o Cypress possibilita a execução paralela por meio da ferramenta Cypress Cloud. Esta opção, embora apresente um plano gratuito, impõe limitações na quantidade de usuários por equipe e na quantidade de execuções dos testes por mês.

No que diz respeito ao Playwright, ele destaca-se por oferecer suporte nativo para a execução paralela, pois executa testes em paralelo por padrão, utilizando vários processos de trabalho simultaneamente. Além disso, ele permite configurar esse tipo de execução ocorra no mesmo arquivo, executando diversos casos de teste de um

mesmo cenário, proporcionando flexibilidade e eficiência .

Considerando esses aspectos, o Playwright emerge como o vencedor nesta categoria, oferecendo uma solução pronta e nativa para a execução paralela, sem a necessidade de instalação de ferramentas adicionais. Além disso, é uma opção totalmente gratuita, proporcionando uma experiência de escrita de testes mais eficiente e econômica para equipes de desenvolvimento.

5.6.9 Geração Visual de Testes

O Selenium WebDriver não oferece nativamente um recurso para a geração visual de testes. No entanto, pode-se contornar essa limitação utilizando ferramentas adicionais como o Selenium IDE, que possibilita a criação e exportação de scripts de teste em seu ambiente.

Já o Cypress não possui funcionalidades específicas para geração visual de testes como padrão, mas há outras alternativas, como a extensão do navegador Google Chrome denominada “Cypress Chrome Recorder” e o Cypress Studio, que embora sejam experimentais até a versão atual no momento da escrita do projeto (v3.4.1), ambas oferecem os recursos de captura das interações do usuário e de exportação como scripts de teste do Cypress.

Por outro lado, o Playwright destaca-se ao incluir nativamente a funcionalidade de captura de interações do usuário e geração automática de scripts de teste. O Playwright facilita o processo ao permitir a interação com o site desejado em uma janela do navegador e gerar scripts prontos para uso.

Considerando essas características, o Playwright vence nesta categoria. Sua capacidade de gerar testes prontos para uso sem a necessidade de ferramentas adicionais ou configurações adicionais proporciona uma experiência de escrita de testes mais eficaz e direta.

5.6.10 Reaproveitamento de Código

O Selenium WebDriver requer a adoção de práticas específicas, como o Page Object Model (Modelo de Objeto de Página), criado por Martin Fowler em 2013 (FOWLER, 2013). Este modelo permite o reaproveitamento de código por meio da organização de elementos e funcionalidades em objetos de página.

Tanto o Cypress quanto o Playwright destacam-se ao oferecer suporte nativo para o reaproveitamento eficiente de código, pois ambos permitem a criação de comandos personalizados, comportamentos de espera e validações que podem ser reutilizados em vários testes, sem a necessidade de criar um modelo de objeto de página.

Essa flexibilidade contribui para uma escrita de testes mais concisa e modular.

Portanto, há um empate do Cypress e Playwright. Ambas as ferramentas proporcionam recursos robustos para o reaproveitamento de código, garantindo uma abordagem eficaz na escrita de testes automatizados.

Considerando todos os recursos de ferramentas de teste abordados nesta seção e destacados na Tabela 10, de modo geral, a melhor ferramenta nestes quesitos é o Playwright, sendo o único que possui todos os atributos por padrão sem a necessidade de adicionar novas ferramentas, ou de configurações mais robustas.

	Selenium WebDriver	**Cypress**	**Playwright**
Interface de Execução de Testes	PARCIAL	SIM	SIM
Recarregamento rápido	NÃO	SIM	SIM
Depuração de viagem no tempo	NÃO	SIM	SIM
Automação da Espera pela Disponibilidade de Elementos	NÃO	SIM	SIM
Seletor de Elementos Simplificado	NÃO	SIM	SIM
Simulação de Requisições Web	NÃO	SIM	SIM
Lida com Múltiplas Abas e navegadores ao mesmo tempo	SIM	NÃO	SIM
Execução Paralela	PARCIAL	PARCIAL	SIM
Geração Visual de Testes	PARCIAL	PARCIAL	SIM
Reaproveitamento de Código	SIM	SIM	SIM

Tabela 10 – Comparação entre os recursos das ferramentas.

Fonte: elaboração própria.

5.7 Depuração dos testes

A depuração feita pelo Selenium WebDriver possui dependência da IDE e da linguagem de programação utilizada, não apresentando opções nativas de depuração em sua documentação, exigindo configurações e ferramentas complementares para esse fim.

No caso do Cypress, a depuração é facilitada através de sua interface gráfica de testes. Isso proporciona um ambiente intuitivo para depurar testes, tanto pela interface quanto pelo console do navegador. O Cypress ainda se destaca ao incorporar a funcionalidade de depuração de viagem no tempo, permitindo a análise detalhada do teste em cada etapa da execução .

O Playwright oferece uma gama diversificada de opções nativas para depuração e recomenda o uso do editor de código Visual Studio Code, uma escolha popular entre desenvolvedores (STACK OVERFLOW, 2023), cuja extensão desenvolvida pela equipe da Microsoft chamada “*Playwright Test for VSCode*” facilita a depuração direta, exibindo mensagens de erro, permitindo a definição de pontos de interrupção e a navegação nos testes.

Outra opção que o Playwright oferece é a de depuração pela interface gráfica dos testes que permite que percorra os testes, adicione ou edite seletores de elementos em tempo real. Existem ainda mais duas maneiras que o Playwright permite a depuração, sendo elas depuração pelo console do navegador em que os testes estão sendo executados e por último através da depuração de viagem no tempo que permite analisar o teste em cada etapa da execução.

Considerando os resultados na Tabela 11, observa-se que o Playwright possui vantagem em relação as opções oferecidas, incluindo a recomendação do Visual Studio Code com a extensão dedicada para depuração, a ferramenta se destaca como a escolha superior nesta categoria, proporcionando uma experiência abrangente e eficiente na depuração de testes automatizados.

	Selenium WebDriver	**Cypress**	**Playwright**
Interface gráfica da ferramenta	NÃO	SIM	SIM
Navegador	NÃO	SIM	SIM
Depuração de viagem no tempo	NÃO	SIM	SIM
IDE	NÃO	NÃO	SIM

Tabela 11 – Comparação entre os recursos das ferramentas.

Fonte: elaboração própria.

5.8 Métricas de velocidade

Nesta seção, são apresentados os resultados obtidos do estudo de caso analisando as métricas de velocidade para as ferramentas Selenium WebDriver, Cypress e Playwright. Essas métricas foram calculadas com base em 100 execuções de um cenário de teste repetidos para cada ferramenta, conforme descrito na seção de Diretrizes gerais 4.2. Importante destacar que todos os testes foram inicialmente executados em série. Porém, pelo Playwright possuir a funcionalidade de paralelismo nativamente e ativada por padrão como abordado na seção 5.6.8, os testes também foram realizados utilizando esta funcionalidade para uma comparação de desempenho.

Ao observar a média de tempo de execução na Tabela 12 abaixo, o Playwright destaca-se por possuir vantagem em relação às outras ferramentas tanto ao utilizar a funcionalidade do paralelismo quanto sem utilizar. Essa performance média do Playwright é atribuída à sua arquitetura, onde a comunicação entre o lado do cliente e o navegador ocorre por meio de websockets, otimizando seu desempenho, como discutido na seção de arquitetura das ferramentas de teste 2.3.

O Selenium WebDriver por sua vez, teve a segunda melhor performance entre as ferramentas, ficando atrás do Playwright sem paralelismo em aproximadamente 14% e com paralelismo em aproximadamente 501%. Esta performance também está

relacionada a sua arquitetura, tendo em vista que cada comando deve ir e voltar do lado do cliente Selenium para o Servidor Proxy para driver do navegador. Adjunto a isso, por não possuir a funcionalidade de espera automática exige que para cada clique, seleção de elementos, obtenção da URL seja necessário esperar por eles estarem disponíveis para uso, o que aumenta seu tempo de execução.

Já o Cypress teve o pior desempenho entre as ferramentas, ficando atrás do Playwright sem paralelismo em aproximadamente em aproximadamente 20% e com paralelismo em aproximadamente 531%. Assim como as outras ferramentas, esta performance também ocorre devido à sua arquitetura, onde embora o Cypress possa ser utilizado para testar sites em produção, seu foco é testar aplicativos localmente à medida que ele é desenvolvido.

	Selenium WebDriver	**Cypress**	**Playwright sem paralelismo**	**Playwright com paralelismo**
Média (s)	42,14	44,26	36,89	7,01
Desvio Padrão (s)	2,07	2,41	2,39	1,37
Coeficiente de Variação	4,91%	5,44%	6,47%	19,58%
Percentil 95 (s)	45,65	47,91	41,66	9,70

Tabela 12 – Métricas de velocidade.

Fonte: elaboração própria.

A Figura 14 a seguir, destaca a a diferença desses valores de forma visual. Assim, evidenciando ainda mais que a ferramenta Playwright destaca-se não apenas ao utilizar o paralelismo, mas também ao apresentar uma melhor performance em execuções em série em relação às outras ferramentas.

5.9 Integração com ambientes de CI/CD

No contexto da integração com ambientes de CI/CD conforme os resultados destacados na Tabela 13, as três ferramentas apresentam a capacidade de serem integradas efetivamente. Todas oferecem a opção de execução de testes sem depender de uma interface visual do navegador, permitindo sua incorporação em *pipelines*[1] de teste.

	Selenium WebDriver	**Cypress**	**Playwright**
Suporta integração	SIM	SIM	SIM
Documentação da integração	NÃO	SIM	SIM

Tabela 13 – Comparação do suporte e documentação da integração com ambientes de CI/CD das ferramentas.

Fonte: elaboração própria.

[1] Processo automático que guia o código da integração à entrega.

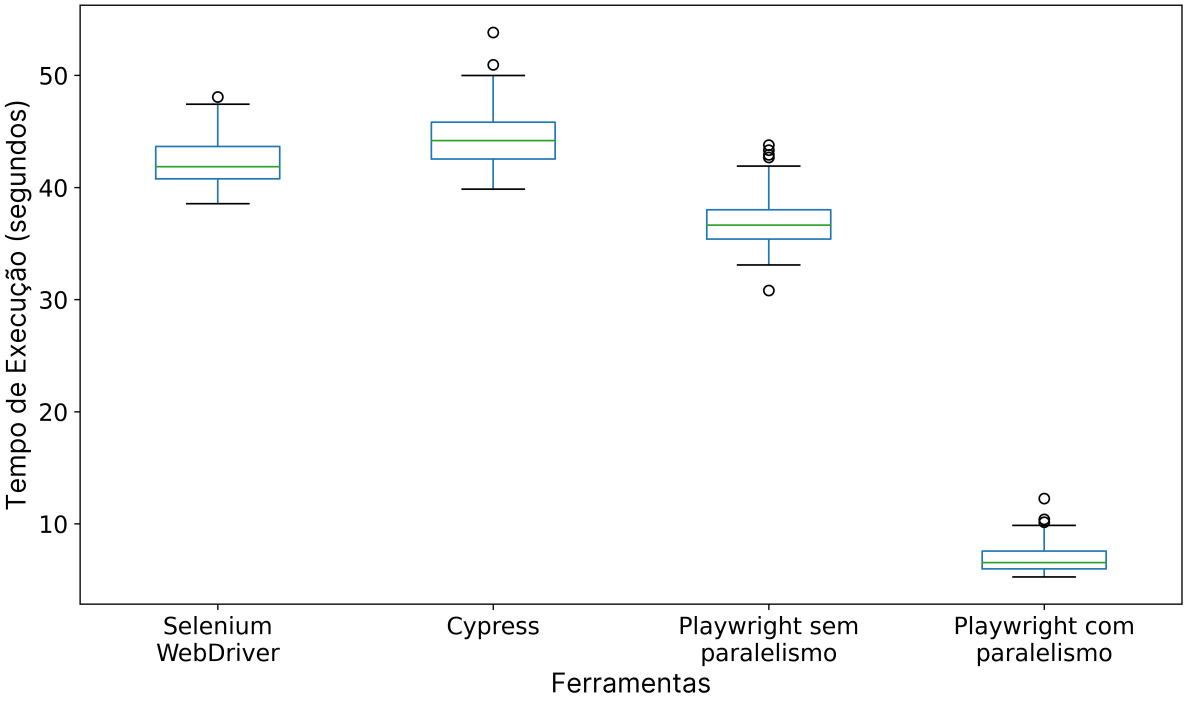

Figura 14 – Gráfico de caixa de Comparação de Tempos de Execução entre Ferramentas de Automação de Teste.

Fonte: elaboração própria.

Entretanto, observa-se que a documentação do Selenium carece de informações detalhadas nesse aspecto. Em contraste, as documentações do Cypress e Playwright fornecerem informações abrangentes sobre a implementação, incluindo exemplos práticos e integrações documentadas com diversos ambientes de integração contínua, como Docker, Jenkins, GitHub Actions, Bitbucket, AWS CodeBuild, entre outros. Essa diferenciação nas documentações influencia a facilidade de implementação e a eficácia da integração com práticas de CI/CD.

5.10 Discussão dos resultados

Com base nos resultados obtidos no trabalho, resumidos no Apêndice B, é evidente que cada ferramenta apresenta distinções significativas. O Selenium WebDriver, por exemplo, se estabeleceu como uma ferramenta robusta, contando com uma comunidade extensa. A ferramenta oferece suporte a diversas linguagens de programação e serve como base de implementação para diversos frameworks. No entanto, vale salientar que para a utilização do Selenium WebDriver são necessárias configurações e integrações adicionais para aprimorar a usabilidade da ferramenta. Esse desafio pode ser superado graças à presença de uma comunidade vasta, pronta para fornecer suporte e compartilhar conhecimentos, mesmo diante da falta de algumas informações na documentação oficial.

O Cypress oferece uma configuração inicial fácil, uma documentação abrangente e recursos adicionais, como cursos gratuitos, que facilitam o aprendizado da equipe de desenvolvimento. Além disso, a ferramenta conta com diversos recursos já integrados para aprimorar a usabilidade, e para equipes que necessitam de funcionalidades mais específicas, apresenta uma extensa lista de integrações, proporcionando uma experiência ainda mais fluida. No entanto, é importante notar que, devido à sua arquitetura, o Cypress não suporta a realização de testes em múltiplas abas. Além disso, ele está limitado às linguagens de programação JavaScript e TypeScript, o que pode representar um obstáculo em projetos que demandam a utilização de outras linguagens. Com isso se torna essencial avaliar cuidadosamente os requisitos do projeto antes de optar pelo uso do Cypress.

Já o Playwright possui uma amálgama das características das duas ferramentas, possuindo um suporte maior em relação ao Cypress de linguagens de programação suportadas. Além disso, possui uma configuração adicional mais simplificada, alinhada a uma documentação, mesmo que menor que a do Cypress, muito completa e direta. Por padrão oferece diversos recursos que melhoram a usabilidade da ferramenta, possuindo o diferencial de executar os testes em paralelo sem necessidade de muita configuração ou pagamento de um serviço em nuvem para obter potencial máximo dessa funcionalidade. No entanto, possui uma comunidade ainda em ascensão, o que dificulta a resolução de problemas não documentados.

Portanto, a escolha entre ferramentas depende de alguns fatores como suporte das linguagens de programação, tendo em vista que as ferramentas possuem um número limitado de linguagens que dão suporte. Além disso, outro fator crucial é a necessidade de funcionalidades pela ferramenta, como recursos de relatório, depuração e configuração. De modo geral o Playwright mesmo possuindo uma comunidade ainda em ascensão, possui uma boa combinação entre as características analisadas, se destacando entre as outras ferramentas.

6 Conclusão e trabalhos futuros

Nesta pesquisa foi realizado uma análise comparativa entre as principais ferramentas de automação de testes para web, com o objetivo de elucidar as características e funcionalidades específicas de cada ferramenta, para facilitar o processo de escolha da ferramenta mais adequada às necessidades específicas de cada projeto.

O trabalho passou, primeiramente, por capítulos de referencial teórico e trabalhos relacionados para definir conceitos importantes para a compreensão do contexto em que o projeto está inserido e para a compreensão das atividades realizadas durante a parte prática. A partir desses conceitos, foram estabelecidos e descritos os métodos utilizados na parte prática. Por fim, para prática foi realizado o desenvolvimento de uma revisão de literatura aplicada em um estudo de caso prático realizado para a empresa FlowUp.

Com isso, foi identificado neste trabalho que no contexto de testes automatizados de ponta a ponta para aplicações web, por mais que seja necessário analisar os requisitos necessários de cada projeto para definir a melhor ferramenta, o Playwright possui a melhor combinação de recursos que facilitam o processo de teste. Apesar de sua comunidade ainda estar em ascensão, fornece recursos bem mais atrativos que compensam seus pontos fracos.

Fazendo a comparação desta pesquisa com os trabalhos relacionados 3, é possível notar diferenças em suas conclusões. Enquanto Silva (2019) e Mobaraya e Ali (2019) focaram principalmente a comparação entre Selenium e Cypress, e Ferreira et al. (2022) incluiu também o Robot, esta pesquisa direcionou o escopo para o Selenium, Cypress e Playwright. Portanto, as conclusões sobre qual ferramenta é a melhor também variam, dado o escopo diferente de ferramentas abordadas. Em relação à análise de Gruenbaum (2020), as diferenças encontram-se nos recursos das ferramentas que foram escolhidos para serem avaliados. A seleção dele levou à conclusão de que a melhor escolha de ferramenta depende dos requisitos do projeto. A presente pesquisa analisa outros recursos que permitiram fazer uma comparação mais robusta, sendo possível assim concluir que o Playwright possui mais vantagens em relação às outras ferramentas comparadas no contexto de testes automatizados de ponta a ponta em ambientes de aplicações web. Além disso, é interessante observar que, embora este trabalho e o trabalho de Metin (2023) se concentrem em contextos diferentes, ambos concluem que o Playwright é a melhor ferramenta, o que pode indicar a versatilidade e eficácia da ferramenta.

No entanto é importante elucidar que essa pesquisa é um trabalho limitado pela

quantidade de ferramentas testadas, já que seria possível utilizar outras ferramentas que desempenham funções semelhantes às utilizadas, de forma a compará-las passando pelo mesmo processo e pelos mesmos critérios de comparação. Além disso, também é limitado pela pequena quantidade de casos de testes executados. Com uma maior quantidade de ferramentas e de cenários de teste, permitiria explorar ainda mais as capacidades das ferramentas de teste utilizadas. Vale ressaltar que o conhecimento e experiência da equipe são fatores que podem ter influência significativa na escolha da ferramenta mais adequada. Como este trabalho se baseia em uma ponderação equitativa dos critérios de avaliação, alterar essa ponderação pode resultar em uma ferramenta vencedora diferente.

Com isso em mente, sugere-se para trabalhos futuros a aplicação dos métodos desta pesquisa em outras ferramentas utilizando os casos de uso específicos do projeto a ser testado, o que pode possibilitar realizar cenários mais reais para a necessidade do projeto. Por fim, para enriquecer a comparação, é viável incorporar critérios subjetivos e estabelecer pesos diferentes para cada critério, permitindo a participação de diversos desenvolvedores no processo de elaboração de testes para uma aplicação. Essa abordagem possibilita a coleta de opiniões e experiências, podendo ser conduzida por meio de formulários ou outras estratégias interativas.

Referências

COHN, M. *Succeeding with Agile: Software Development Using Scrum*. 1. ed. [S.l.]: Addison-Wesley Professional, 2009. Citado 2 vezes nas páginas 21 e 22.

CYPRESS. *How Cypress Works*. 2024. <https://www.cypress.io/how-it-works/>. (Online; Acessado em 26/02/2024). Citado 4 vezes nas páginas 14, 25, 30 e 46.

FERREIRA, A. C. da S. et al. Selenium, robot e cypress: Um estudo comparativo entre ferramentas de automaÇÃo de teste. *Revista de Tecnologia da Informação da Faculdade Lourenço Filho*, v. 5, n. 5, 2022. Citado 3 vezes nas páginas 32, 33 e 63.

FOWLER, M. *Page Object*. 2013. <https://martinfowler.com/bliki/PageObject.html>. (Online; Acessado em 26/02/2024). Citado na página 57.

FOWLER, M. *Is High Quality Software Worth the Cost?* 2019. <https://martinfowler.com/articles/is-quality-worth-cost.html>. (Online; Acessado em 26/02/2024). Citado 3 vezes nas páginas 13, 17 e 18.

GLOBALSTAT. *Statcounter Global Stats - Browser Market Share Worldwide*. 2023. <https://gs.statcounter.com/browser-market-share/#monthly-202208-202308-bar>. (Online; Acessado em 26/02/2024). Citado 3 vezes nas páginas 7, 36 e 37.

GRUENBAUM, B. *Puppeteer, Selenium, Playwright, Cypress – how to choose?* 2020. Citado 3 vezes nas páginas 32, 33 e 63.

ISO Central Secretary. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Geneva, CH, 2011. Disponível em: <https://www.iso.org/standard/35733.html>. Citado 2 vezes nas páginas 18 e 34.

METIN, H. *Testing of GLSP-based Web Modeling Tools*. Tese (Doutorado) — Wien, 2023. Citado 2 vezes nas páginas 33 e 63.

MOBARAYA, F.; ALI, S. Technical analysis of selenium and cypress as functional automation framework for modern web application testing. In: . [S.l.: s.n.], 2019. Citado 3 vezes nas páginas 32, 33 e 63.

PLAYWRIGHT. *Fast and reliable end-to-end testing for modern web apps | Playwright*. 2024. <https://playwright.dev/>. (Online; Acessado em 26/02/2024). Citado 5 vezes nas páginas 14, 24, 30, 31 e 46.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. [S.l.]: McGraw Hill Brasil, 2016. Citado 7 vezes nas páginas 13, 14, 17, 18, 19, 20 e 21.

PUPPETTER. *Puppetter | Docs*. 2024. <https://pptr.dev/>. (Online; Acessado em 26/02/2024). Citado na página 24.

SELENIUM. *The Selenium Browser Automation Project*. 2024. <https://www.selenium.dev/>. (Online; Acessado em 26/02/2024). Citado 6 vezes nas páginas 14, 24, 25, 26, 28 e 46.

SILVA, C. G. G. de M. Estudo comparativo de ferramentas de testes de ponta a ponta automatizados em sistemas web. 2019. Citado 3 vezes nas páginas 32, 33 e 63.

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. [S.l.]: Pearson Education do Brasil, 2019. Citado 3 vezes nas páginas 13, 14 e 20.

STACK OVERFLOW. *2023 Developer Survey*. 2023. <https://survey.stackoverflow.co/2023/#most-popular-technologies-language>. (Online; Acessado em 26/02/2024). Citado 2 vezes nas páginas 35 e 58.

STACK OVERFLOW. *Recently Active 'selenium-webdriver+or+cypress+or+playwright' Questions - Stack Overflow*. 2024. <https://stackoverflow.com/questions/tagged/selenium-webdriver+or+cypress+or+playwright>. (Online; Acessado em 26/02/2024). Citado na página 50.

STACK OVERFLOW. *Stack Overflow Trends*. 2024. <https://insights.stackoverflow.com/trends?tags=selenium-webdriver,cypress,playwright>. (Online; Acessado em 26/02/2024). Citado na página 51.

THOUGHTWORKS. *Technology Radar – An opinionated guide to today's technology landscape*. 2023. <https://www.thoughtworks.com/radar>. (Online; Acessado em 26/02/2024). Citado na página 24.

TORNHILL, A.; BORG, M. *Code Red: The Business Impact of Code Quality – A Quantitative Study of 39 Proprietary Production Codebases*. 2022. Citado na página 13.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Editora: Independente, 2020. Citado 4 vezes nas páginas 13, 14, 22 e 23.

VOCKE, H. *Page Object*. 2013. <https://martinfowler.com/bliki/PageObject.html>. (Online; Acessado em 26/02/2024). Citado 3 vezes nas páginas 14, 21 e 22.

W3C. *The history of the Web*. 2019. <https://www.w3.org/wiki/The_history_of_the_Web>. (Online; Acessado em 26/02/2024). Citado na página 25.

YEEN, J.; SADYM, M. *A look back in time: the evolution of test automation*. 2023. <https://developer.chrome.com/blog/test-automation-evolution/>. (Online; Acessado em 26/02/2024). Citado 3 vezes nas páginas 23, 24 e 25.

Apêndices

APÊNDICE A – RELAÇÃO DAS CARACTERÍSTICAS ANALISADAS COM OS ATRIBUTOS DA ISO/IEC 25010:2011

Características analisadas	**Atributos da ISO/IEC 25010:2011**
Linguagens de programação	Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Segurança, Manutenibilidade, Portabilidade
Suporte a navegadores	Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Segurança, Manutenibilidade, Portabilidade
Configuração do projeto de automação	Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade, Portabilidade
Documentação da ferramenta e suporte da comunidade	Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade, Portabilidade
Recursos de relatórios e registro de testes	Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade, Portabilidade
Recursos de ferramenta de teste	Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade, Portabilidade
Depuração dos testes	Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade
Métricas de velocidade	Eficiência de performance
Integração com ambientes de CI/CD	Adequação funcional, Eficiência de performance, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Manutenibilidade, Portabilidade

Tabela 14 – Relação das características analisadas com os atributos da ISO/IEC 25010 de 2011.

Fonte: elaboração própria.

APÊNDICE B – RESUMO DOS RESULTADOS

Características	**Selenium WebDriver**	**Cypress**	**Playwright**
Linguagens de programação	Javascript, Typescript, Python, C#, Java, Kotlin e Ruby	Javascript e Typescript	Javascript, Typescript, Python, C# e Java
Suporte a navegadores	Chrome, Safari, Edge e Firefox	Chrome, Safari, Edge e Firefox	Chrome, Safari, Edge e Firefox
Configuração do projeto de automação	Complexa, requer instalação de outras ferramentas e drivers	Simplificada, Requer configuração mínima	Simplificada, Requer configuração mínima
Clareza da documentação	Boa documentação. Carece de informações na área de testes	Excelente documentação, robusta e com muitos exemplos	Excelente documentação, robusta e com muitos exemplos
Padrões e Práticas Recomendadas	Boas práticas de teste de software, não focadas nas boas práticas específicas da ferramenta	Excelentes práticas específicas da ferramenta. Numerosos exemplos elucidando o que fazer e o que evitar	Excelentes práticas específicas da ferramenta. Numerosos exemplos elucidando o que fazer e o que evitar
Integração	Dedica uma seção ao ecossistema, mas carece de informações específicas, levando a busca por informações em outras fontes	Documentação abrangente sobre integrações, fornecendo exemplos práticos e orientações explícitas	Possui algumas lacunas na integração com outras ferramentas, porém possui funcionalidades nativas que reduzem a necessidade de recursos adicionais
Fontes adicionais de conhecimento	Github, Youtube, Comunidade Oficial e Perfil no Twitter	Github, Youtube, Comunidade Oficial, Perfil no Twitter, Blog Oficial, Curso gratuito oficial e suporte personalizado pago	Github, Youtube, Comunidade Oficial, Perfil no Twitter e Blog Oficial
Participação da Comunidade	Comunidade muito forte e madura	Comunidade grande e ativa	Comunidade crescente e ativa

Características	**Selenium WebDriver**	**Cypress**	**Playwright**
Recursos de relatórios e registro de testes	Básicos, depende da configuração e integração com outras ferramentas	Possui ferramentas de depuração integradas que inclui métricas de cobertura de teste, capturas de tela, gravações de vídeo	Possui ferramentas de depuração integradas que inclui métricas de cobertura de teste, capturas de tela, gravações de vídeo
Interface de Execução de Testes	Não possui	Possui	Possui
Recarregamento rápido	Não possui	Possui	Possui
Depuração de viagem no tempo	Não possui	Possui	Possui
Automação da Espera pela Disponibilidade de Elementos	Não possui	Possui	Possui
Seletor de Elementos Simplificado	Não possui	Possui	Possui
Simulação de Requisições Web	Não	Sim	Sim
Lida com Múltiplas Abas e navegadores ao mesmo tempo	Sim	Não	Sim
Execução Paralela	Requer ferramenta de terceiros	Requer o Cypress Cloud que possui plano pago	Suporte integrado para paralelização
Geração Visual de Testes	Com Selenium IDE	Com Extensão do navegador ou experimental com Cypress Studio	Suporte integrado para funcionalidade
Reaproveitamento de Código	Depende de padrões como o POM	Suporta extensibilidade das funcionalidades	Suporta extensibilidade das funcionalidades
Depuração dos testes	Depende da IDE	Pela Interface Gráfica, depuração de viagem no tempo e console do navegador	Pela Interface Gráfica, depuração de viagem no tempo, console do navegador e pelo Visual Studio Code
Métricas de velocidade	Bom	Bom	Excelente, principalmente com paralelismo
Suporte a integração com ambientes de CI/CD	Sim, porém sem muita documentação oficial	Sim, bem documentado com exemplos	Sim, bem documentado com exemplos

Tabela 15 – Visão geral dos resultados de cada ferramenta.
Fonte: elaboração própria.