Universidade Federal do Rio Grande do Norte
Departamento de Engenharia de Computação e Automação
Curso de Engenharia de Computação

Estudo Comparativo de Ferramentas de Testes de Ponta a Ponta Automatizados em Sistemas Web

Carlos Gabriel Gomes de Melo Silva

Natal-RN
dezembro de 2019

Carlos Gabriel Gomes de Melo Silva

Estudo Comparativo de Ferramentas de Testes de Ponta a Ponta Automatizados em Sistemas Web

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Engenharia de Computação e Automação – DCA

Curso de Engenharia de Computação

Orientador: Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui

Natal - RN

dezembro de 2019

Dedico este trabalho a toda minha família, amigos e professores.

Agradecimentos

Agredeço a todos da minha família que contribuíram para minha formação, sendo ela a base para minhas conquistas. Em especial a minha mãe Albanir pela educação e por não me deixar desistir dos meus sonhos. Agradeço aos meus avós por tudo que fizeram por mim e acima de tudo deixo este trabalho com uma dedicação especial para eles.

Quero agradecer em especial as pessoas que foram fundamentais para essa conquista e que me deram condições de concluir esse estágio na minha vida, meu agradecimento especial para minhas amigas Maria Gelza, Dayana Amaro e Sula, que fazem parte da minha família do coração. Também agradeço a minha comadre Cristina Ferreira e toda a família pela a ajuda, bem como gostaria de deixar meus agradecimentos aos meus tios Almir, Nazaré, Adelia e Ricardo pelo apoio. Agradeço a minha companheira Layrane que me apoiou muito durante a construção deste trabalho.

Agradeço aos meus amigos pela paciência e por me apoiar, deixo aqui alguns nomes de pessoas que foram muito importantes para mim durante a realização do curso. Agradeço aos amigos Philipy, Mateus, Daniel, André, Miguel, Álvaro, Gabriel Signoretti, Felipe Fernandes, Solano, Hemerson Lucas, Camilly, Washington, Mara, Rafael, Ranny, Dido, Samuel, João e Sânzio. Faltam espaços nesta página para agradecer a todos aqueles que foram importantes para mim, por isso fica o meu registro de agradecimento a todos que fazem parte da minha vida.

Faço um agradecimento em especial ao professor Aquiles Burlamaqui, por não desistir de mim e acreditar que eu conseguiria chegar ao final desta jornada, também agradeço por todo o conhecimento e oportunidade profissional que me foi concedida.

“To me, legacy code is simply code without tests.”

Michael Feathers.

Estudo Comparativo de Ferramentas de Testes de Ponta a Ponta Automatizados em Sistemas Web

Autor: Carlos Gabriel Gomes de Melo Silva
Orientador: Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui

Resumo

Esse trabalho consiste no estudo das ferramentas de automação de testes Cypress e Selenium, bem como também tem por objetivo avaliar através de um estudo documental suas aplicações, e evidenciar as vantagens e desvantagens na utilização de cada ferramenta. A motivação do trabalho é demonstrar a evolução das ferramentas de testes de software em nível de teste de ponta a ponta, avaliando a recente solução Cypress, de forma que seja apresentado quais são os benefícios reais de fazer uso desta ferramenta em relação ao Selenium, que por sua vez é uma ferramenta mais consolidada no meio da automação de testes. Neste trabalho será abordado os problemas existentes na automação de testes ponta a ponta, e os recursos disponíveis nas ferramentas. Dessa forma, objetiva-se justificar a utilização do Cypress na melhoria do processo de teste de software.

Palavras-chave: Cypress, Testes ponta a ponta, Automação de Testes, Selenium.

Comparative Study of Automated End-to-End Testing Tools in Web Systems

Author: Carlos Gabriel Gomes de Melo Silva
Supervisor: Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui

Abstract

This work consists in the study of the test automation tools Cypress and Selenium, besides that to evaluate through a documentary study their applications, and show the advantages and disadvantages in the use of each tool. The motivation of the work is to demonstrate the evolution of end-to-end test-level software testing tools by evaluating the recent Cypress solution so that the real benefits of using this tool over Selenium are presented. This in turn is a more consolidated tool in the midst of test automation. This paper will address the issues that exist in end-to-end test automation, and the features available in the tools. Thus, the objective is to justify the use of Cypress to improve the software testing process.

Keywords: Cypress, End-to-End Testing, Test Automation, Selenium.

Lista de ilustrações

Figura 1 – Representação do teste estrutural (caixa-branca). 20
Figura 2 – Representação do teste funcional (caixa-preta). 21
Figura 3 – Representação da pirâmide de testes de Mike Cohn. 24
Figura 4 – Diferenças entre testes E2E e UI. 26
Figura 5 – Representação de uma pirâmide de testes com testes E2E. 27
Figura 6 – Interface do *plugin* Selenium IDE para o Chrome. 30
Figura 7 – Diagrama da arquitetura do Selenium RC. 31
Figura 8 – Diagrama da arquitetura do Selenium WebDriver. 32
Figura 9 – Diagrama da arquitetura do Cypress. 34
Figura 10 – Diagrama representando o CI/CD. 35
Figura 11 – Relatório de divisão de mercado entre os navegadores em 2019. 38
Figura 12 – Configuração de um projeto de testes com Cypress. 46
Figura 13 – Interface visual do Cypress sendo executado. 48
Figura 14 – Opção de depuração do Cypress no Google Chrome. 51

Lista de tabelas

Tabela 1 – Comparação de uso entre os principais OS. 44
Tabela 2 – Comparação de uso entre os principais navegadores. 45
Tabela 3 – Comparação entre as configurações das ferramentas. 46
Tabela 4 – Comparação entre as linguagens de programação de escrita dos testes. 47
Tabela 5 – Comparação entre os recursos de escrita. 48
Tabela 6 – Comparação entre as formas de depuração. 50
Tabela 7 – Comparação entre as arquiteturas das ferramentas. 51
Tabela 8 – Comparação entre os relatórios de execução. 52
Tabela 9 – Comparação entre as documentação das ferramentas. 53
Tabela 10 – Comparação entre o Uso em CI/CD. 53

Lista de Abreviaturas

API *Application Programming Interface*

AUT *Application Under Test*

CD *Continuous Delivery* ou *Deployment*

CI *Continuous Integration*

E2E *End-To-End*

IDE *Integrated Development Environment*

JSON *JavaScript Object Notation*

OS *Operating System*

RC *Remote Control*

UI *User Interface*

V&V Verificação e Validação

XP *Extreme Programming*

Sumário

Introdução 12

1 INTRODUÇÃO 12
1.1 Motivação 14
1.2 Objetivos 14
1.2.1 Objetivo Geral 15
1.2.2 Objetivo Específicos 15
1.3 Metodologia 15
1.4 Estrutura do Trabalho 15

Capítulo 2 16

2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 Qualidade de software 17
2.1.1 Fatores de Qualidade 18
2.2 Testes de software 19
2.2.1 Técnicas de Testes 19
2.2.1.1 Teste Estrutural 20
2.2.1.2 Teste Funcional 21
2.2.2 Níveis de Teste de software 22
2.2.3 Testes Automatizados 23
2.2.4 Testes de Ponta a Ponta 25
2.2.5 Verificação e Validação 27
2.3 Ferramentas de Testes Automatizados 28
2.3.1 Selenium 29
2.3.1.1 Selenium IDE 29
2.3.1.2 Selenium RC 30
2.3.1.3 Selenium WebDriver 32
2.3.1.4 Selenium Grid 33
2.3.2 Cypress 33
2.3.3 Integração, Entrega e Implantação Contínua 34

Capítulo 3 35

3 METODOLOGIA E CARACTERÍSTICAS ANALISADAS 36
3.1 Metologia Adotada 36
3.2 Características Analisadas 37

3.2.1 Execução em Sistemas Operacionais . . . 37
3.2.2 Execução em Navegadores . . . 37
3.2.3 Configuração do Projeto de Automação . . . 38
3.2.4 Linguagens de Programação . . . 39
3.2.5 Recursos de Escrita de Testes . . . 39
3.2.6 Depuração dos Testes . . . 40
3.2.7 Arquitetura das Ferramentas . . . 41
3.2.8 Relatório de Execução . . . 42
3.2.9 Documentação . . . 42
3.2.10 Possibilidade de Uso em CI/CD . . . 43

Capítulo 4 . . . 43

4 RESULTADOS E DISCUSSÕES . . . 44
4.1 Comparações . . . 44
4.1.1 Execução em Sistemas Operacionais . . . 44
4.1.2 Execução em Navegadores . . . 45
4.1.3 Configuração do Projeto de Automação . . . 45
4.1.4 Linguagens de Programação . . . 47
4.1.5 Recursos de Escrita de Testes . . . 47
4.1.5.1 Interface da Execução dos Testes . . . 48
4.1.5.2 Seletor de Elementos Via Interface . . . 49
4.1.5.3 *Hot Reload* . . . 49
4.1.5.4 *Time Travel* . . . 49
4.1.5.5 Espera Automática . . . 49
4.1.6 Depuração dos Testes . . . 50
4.1.7 Arquitetura das Ferramentas . . . 51
4.1.8 Relatório de Execução . . . 52
4.1.9 Documentação . . . 52
4.1.10 Possibilidade de Uso em CI/CD . . . 53

Capítulo 5 . . . 53

5 CONSIDERAÇÕES FINAIS . . . 54

REFERÊNCIAS . . . 55

1 Introdução

Nos dias atuais os softwares assumem um importante papel na nossa sociedade, sendo responsáveis por automatizar difíceis processos e dando suporte a diversas organizações dos mais variados segmentos. Segundo (PRESSMAN; MAXIM, 2016) os indivíduos, negócios e governos estão dependendo, de forma crescente, de software para tomada de decisões, de forma que uma falha no software pode provocar desde pequenos inconvenientes até grande catástrofes.

Cada vez mais os softwares desempenham uma responsabilidade maior na sociedade, servindo como alicerce para negócios ou sendo a fonte de negócios. Atualmente existe uma busca incessante pela automatização de processos cada vez mais complexos, que torna os softwares mais complexos também. (RIOS; MOREIRA, 2006) descreveu em sua obra que os importantes objetivos das organizações são reduzir os custos de desenvolvimento e aumentar a qualidade dos produtos finais. Porém atingir esses objetivos é tremendamente difícil pelo aumento da complexidade dos softwares. Como uma alternativa de solução para a diminuição de custos e melhoria da qualidade do produto de software busca-se melhorias no processo de produção de software.

Como definido por (PRESSMAN; MAXIM, 2016), um software de qualidade entrega um valor mensurável para aqueles que o produzem e para aqueles que o utilizam. Com essa definição em mente podemos entender que um software de qualidade atende as necessidades dos clientes e também são confiáveis, para isso é necessário pensar sempre na qualidade do software.

É inquestionável que os softwares produzidos precisam atender às exigências dos seus usuários finais, bem como precisam também atender à critérios de qualidade específicos. Um dos processos definidos no desenvolvimento de software com o propósito de melhorar a sua qualidade é a prática de testes de software. Com esta prática conseguimos medir e avaliar se o software produzido atende aos requisitos impostos sobre ele durante a sua concepção e desenvolvimento. Segundo a visão de (BARTIÉ, 2002), os softwares mal testados são a causa de prejuízos enormes para as organizações, por este motivo é importante ter um processo de teste maduro, para que assim seja diminuído os riscos de entregas de software com qualidade baixa.

Em sua obra (SOMMERVILLE, 2011), descreve quatro atividades básicas do processo de software, são elas a especificação, projeto e implementação, validação e evolução. Ainda nessa obra a atividade de validação de software é descrita sucintamente como sendo a responsável por mostrar que o software se adequa as especificações e atende as necessidades dos clientes. A evolução é também descrita sucintamente como sendo a

responsável por dar continuidade do software, através de manutenção do mesmo para que se adeque a novas necessidades dos clientes ou mudanças no negócio envolvido.

Neste contexto em que se torna necessário validar primeiramente o software para que ele atenda as expectativas do cliente e também é necessário que mantenha padrões de qualidade, após mudanças no mesmo. Vemos que é necessário cada vez mais a existência de teste de software, pois a cobertura de teste de software dá mais segurança para entregar qualidade no software e também torna o processo de mudança menos drástico durante sua evolução.

Durante o desenvolvimento, validação e evolução do software podem ocorrer diversas mudanças, algumas podem ser bastante impactantes em todo sistema, de forma que testes precisam ser executados novamente em pontos do sistema que já haviam sido validados e fazer isso de forma manual é impraticável em alguns casos, pelo custo orçamentário e de tempo que a prática de realizar novamente todos os teste possui. Dentro desse cenário surge a automação de testes como uma forma de minimizar o custo da atividade de teste do projeto.

Comumente durante as fases de desenvolvimento e validação, são realizados alguns tipos de testes que a longo prazo trazem muitos benefícios ao projeto, durante a fase de desenvolvimento são aplicados os testes unitários e de integração, e durante a fase de validação são aplicados os testes de ponta a ponta (também conhecido como testes *end-to-end* ou E2E). Todos esses tipos de testes podem ser automatizados, de forma que seja reduzido bastante o custo de execução e permita que seja realizado a sua reexecução com custos baixos, dessa forma se torna viável a aplicação de testes de regressão. Como dito por (SOMMERVILLE, 2011), os testes de regressão envolvem a execução de testes que tenham sido executados com sucesso anteriormente. Com essa prática verifica se mudanças no código não introduziram novos erros, e se o código novo interage conforme o esperado com o código existente, sendo por esse motivo muito caro executar testes de regressão e geralmente impraticável quando um sistema é testado manualmente, sem nenhum tipo de automação de testes.

Os teste de ponta a ponta, fornecem uma confiabilidade maior do funcionamento geral do sistema, já que tem como objetivo imitar o comportamento de um cliente no sistema. São executados geralmente em um ambiente controlado similar ao que será utilizado por usuários finais do sistema. Nesse tipo de teste, utilizamos ferramentas de automação de testes que nos permitam a execução dos testes pela interface do sistema, simulando a utilização de um usuário final.

Existem diversas ferramentas de automação para testes de ponta a ponta, a mais conhecida e bastante utilizada para esse fim, é o Selenium que possui integração com diversas linguagens de programação e possui diversas opções de bibliotecas disponíveis para auxiliar na construção de testes automatizados, sendo essa uma das primeiras soluções

que apareceram no mercado para automação de teste nesse nível. Mas atualmente tem aparecido outras opções de ferramentas, como o Cypress, que fornece muitas vantagens para testes de ponta a ponta.

Diante disso, é necessário analisar qual ferramenta utilizar para a criação de testes, tendo em mente as suas vantagens e desvantagens. Sob essa perspectiva o trabalho tende a analisar as duas principais ferramentas, Selenium e a recente ferramenta Cypress, como forma de elucidar suas diferenças e contribuir na análise da utilização das ferramentas em projetos de software.

1.1 Motivação

Com o crescimento da quantidade de ferramentas de automação de testes e evolução das mesmas, está cada vez mais difícil de se avaliar com clareza quais ferramentas devem ser utilizadas em projetos de software. Cada ferramenta tem uma nuance diferente e servem melhores a alguns propósitos do que a outros. Sendo assim é necessário muita pesquisa comparativa para que seja visto qual ferramenta se encaixa melhor em determinados contextos.

Com a evolução das ferramentas no decorrer dos anos tornou-se ainda mais evidente as divergências entre si, que torna o processo de escolha ainda mais criterioso, uma das mudanças drásticas que houve nesse processo de evolução foi na arquitetura dessas ferramentas de automação, que trouxeram diversas vantagens em sua utilização.

Neste trabalho será comparado ferramentas de testes de ponta a ponta de diferentes gerações, para que se entenda quais são os benefícios adquiridos com a evolução das mesmas. Assim espera-se contribuir para a análise das ferramentas disponíveis para automação de testes de ponta a ponta.

1.2 Objetivos

Este trabalho tem como objetivo analisar e comparar as ferramentas de automação de testes de ponta a ponta, mostrando suas diferenças e vantagens de utilização. As ferramentas escolhidas são de gerações diferentes e se propõem a resolver o mesmo problema de formas diferentes. De forma que esse trabalho também é dedicado a mostrar a evolução das ferramentas, e evidenciar os novos recursos disponíveis nas ferramentas de testes da nova geração, com foco no estudo da ferramenta Cypress, comparando com ferramentas mais antigas como o Selenium.

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é mostrar a evolução das ferramentas de testes ponta a ponta, bem como elucidar as vantagens e desvatagens de se utilizar ferramentas mais recentes, como Cypress para a realização de testes automatizados nesse nível.

1.2.2 Objetivo Específicos

Os objetivos específicos que derivam desse objetivo geral são:

- Descrever o processo de testes e como a automação de testes funciona;
- Definir as características de ferramentas de automação de testes de ponta a ponta.
- Descrever as características de cada ferramenta de teste, para que por fim seja feita uma comparação entre as ferramentas.

1.3 Metodologia

Para a realização deste trabalho será considerado uma metodologia de pesquisa exploratória e descritiva, sob o método hipotético-dedutivo, levando em consideração uma abordagem qualitativa. Para este trabalho foi realizado uma pesquisa com base em procedimentos bibliográficos e documentais.

Com isso serão abordados outros trabalhos científicos, documentação oficial das ferramentas, artigos e livros. Para que como resultado se obtenha uma comparação entre as duas principais ferramentas propostas para estudo que é o Selenium e o Cypress, elucidando vantagens e desvantagens de cada uma.

Contribuindo assim para futuros trabalhos que venham utilizar as ferramentas estudadas, bem como contribuir para que seja feito escolhas mais assertivas no uso das ferramentas de automação de testes apresentadas.

1.4 Estrutura do Trabalho

No Capítulo 2 serão abordados os fundamentos téoricos básicos para um melhor entendimento do tema do trabalho proposto, sendo vistos conceitos de qualidade de software, testes de software e sua importância no processo de melhoria da qualidade, as técnicas de testes e os principais níveis de testes. Neste capítulo também serão abordados os conceitos de automação de testes, verificação e validação e uma leve introdução às práticas de integração, entrega e implantação contínua.

Já no Capítulo 3 será visto a metodologia e características que serão analisadas em cada ferramenta, tendo o objetivo de explicar como cada característica observada é importante para a comparação entre as ferramentas que será realizada no fim deste trabalho. No Capítulo 4 é feito as comparações entre as ferramentas e também é realizado uma breve discussão sobre cada resultado apresentado.

No Capítulo 5 temos a conclusão, onde é realizado uma síntese de toda a discussão que foi realizada durante este trabalho, neste último capítulo é dado um veredito sobre a utilização das ferramentas baseado nas dicussões feitas nos capítulos anteriores.

2 Fundamentação Teórica

Neste capítulo são abordados um conjunto de assuntos essenciais, para o melhor entendimento da discussão proposta pelo trabalho. Tendo como objetivo elucidar os seguintes tópicos: qualidade de software, testes de software, testes de ponta a ponta e testes automatizados, bem como apresentar as ferramentas de automação de testes. Será conceituado de forma breve as práticas de integração, entrega e implantação contínua.

2.1 Qualidade de software

A qualidade de software pode ser entendida de várias formas, e discutir suas várias definições levaria a um longo processo de discussão acerca da definição que é mais completa, pois se trata de um conceito muito subjetivo onde sua avaliação depende do ponto de vista do interessado na qualidade do software. Como o próprio (SOMMERVILLE, 2011, p. 457) atribui em sua obra "a avaliação da qualidade de software é um processo subjetivo, em que a equipe de gerenciamento de qualidade precisa usar seu julgamento para decidir se foi alcançado um nível aceitável de qualidade".

Uma boa definição da qualidade de software mais simplista foi dada por (PRESSMAN; MAXIM, 2016, p. 360) em sua obra.

> "No sentido mais geral, 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."

Seguindo esta definição acima, podemos crer que para que um software seja considerado de qualidade é preciso que ele entregue valor ao cliente que utiliza o software de maneira que cumpra com seu propósito e também seja confiável.

O software é produzido de acordo com as especificações, que são definidas no início do projeto, juntamente com as partes interessadas. Para (SOMMERVILLE, 2011), a especificação de software é o processo de compreensão e definição dos requisitos do produto de software, bem como a identificação das restrições de operação e de desenvolvimento do software. De modo que o software seja desenvolvido com o propósito de atender as necessidades iniciais dos clientes.

Segundo a visão de (BARTIÉ, 2002) a qualidade de software está atrelada à quantidade de requisitos que foram devidamente testados e estão em conformidade com o especificado. Sob essa perspectiva vemos o quanto os testes de software são importantes

para a prover uma boa qualidade ao software e alinhar suas características de forma a atender melhor às necessidades dos clientes.

2.1.1 Fatores de Qualidade

No esforço de melhorar a qualidade do software, primeiramente é necessário identificar os seus atributos de qualidade fundamentais. A norma técnica 25010 (ISO/IEC, 2011) define oito atributos de qualidade primários para o produto de software, são eles: compatibilidade, usabilidade, segurança, confiabilidade, manutenibilidade, performance e eficiência, adequação funcional, portabilidade. Segundo a norma temos as seguintes definições para cada atributo:

- Compatibilidade: Este atributo é dividido entre interoperabilidade e coexistência. O conceito de interoperabilidade está atrelado a troca de informações e utilização dos dados ou recursos de outras aplicações. O conceito de coexistência está por sua vez mais atrelado ao funcionamento da aplicação com outras em um mesmo ambiente, ou seja, compartilhando recursos mesmo que seja com aplicações externas à aplicação, no qual não se possui controle.

- Usabilidade: É um atributo associado a qualidade em uso do produto. Este atributo pode ser medido por algumas sub-características relevantes: reconhecimento da sua adequação às necessidades do cliente (conformidade com o propósito do produto), capacidade de aprendizagem que o sistema proporciona, operacionalidade, proteção contra erros do usuário, estética da interface do usuário e acessibilidade.

- Segurança: Se resume aos mecanismos internos para controlar o acesso, garantindo a integridade e confidencialidade dos dados dos dispositivos.

- Confiabilidade: Este atributo está profundamente atrelado à disponibilidade e capacidade de recuperação em casos de falha.

- Manutenibilidade: Pode ser entendido como o grau de eficácia ou eficiência com que o produto tem de ser modificado por parte de seus mantenedores de interesse, isso inclui a equipe comercial, operacional ou usuários finais. Esse é um aspecto mais difícil de ser medido diretamente, envolve a análise da aplicação de um conjunto de boas práticas da engenharia de software, incluindo os testes automatizados desde níveis mais baixos (testes sobre as menores partes testáveis do sistema), até testes de alto nível que envolve a integração de todos os componentes de software.

- Performance eficiente: Este atributo trata do tempo de resposta da aplicação, utilização de recursos e capacidade. Geralmente os software não são desenvolvidos em ambiente ou plataforma em que o mesmo será utilizado pelo seus usuários finais, que dificulta mensurar esse atributo antes da entrega.

- Adequação funcional: Este atributo pode ser entendido como uma visão inicial da provável eficácia do produto, a norma em questão a divide em três sub-características que são a completude funcional, adequabilidade funcional e corretude funcional. A completude funcional é uma estimativa se o software está completo com todas as funcionalidades necessárias, a adequabilidade funcional é uma estimativa se o produto está adequado às necessidades do cliente, por sua vez a corretude funcional estima se as respostas que o produto provê estão realmente corretas com o esperado.

- Portabilidade: É um atributo relacionado à capacidade que o software tem de se adaptar e funcionar em diferentes ambientes ou plataformas.

É importante a definição destes atributos definidos pela norma 25010 (ISO/IEC, 2011), para guiar o processo de melhoria da qualidade do produto de software, visto que é necessário se atentar para esses atributos no processo de verificação e validação, para que assim o software tenha uma melhoria efetiva da sua qualidade.

2.2 Testes de software

Para discutir sobre os testes de software é importante primeiro conceituá-lo. Segundo um conceito trazido por (PRESSMAN; MAXIM, 2016), "O teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente". O conceito se estende um pouco mais em outros trabalhos, mas de forma resumida, é um conjunto de atividades do processo de desenvolvimento de software, que requer planejamento e execução sistemática.

Os testes de software possuem objetivos bem específicos, pela visão de (SOMMERVILLE, 2011) os testes possuem dois objetivos, que é demonstrar aos desenvolvedores e clientes que o software atende aos requisitos e descobrir situações que o mesmo se comporta de forma incorreta, indesejável ou diferente das especificações.

2.2.1 Técnicas de Testes

As técnicas de testes são divididas entre testes funcionais e testes estruturais. Na visão de (BARTIÉ, 2002) as duas técnicas não são exclusivas entre si, e sim complementares, podemos então obter um software de maior qualidade ao aplicar as duas técnicas em conjunto.

Segundo a definição dada por (GONTIJO, 2019) as técnicas de testes de software tem como principal objetivo definir como o teste deve ser elaborado. Nesse contexto as técnicas norteiam a elaboração de testes, definindo de que maneira os mesmos serão planejados.

2.2.1.1 Teste Estrutural

Os testes estruturais ou testes de caixa-branca, tem como base a estrutura lógica interna do software para derivar casos de testes. De acordo com (PRESSMAN; MAXIM, 2016) nessa técnica os engenheiros de software podem criar casos de testes que melhor exercitem o código desenvolvido, pois nesse caso existe a possibilidade de acesso ao código-fonte da aplicação.

Ainda de acordo com o mesmo autor, os testes funcionais possibilitam que sejam criados casos testes com os seguintes objetivos:

- Garantir que todos os caminhos independentes de um módulo tenham sido executados pelo menos uma vez.
- Exercitar todas as decisões lógicas nos seus estados verdadeiro e falso.
- Executar todos os ciclos em seus limites e fronteiras operacionais.
- Exercitar as estruturas de dados internas para garantir sua validade.

A Figura 1 exemplifica o teste estrutural de acordo com a visão de um engenheiro de software, que para criar os casos de teste o mesmo possui a percepção da estrutura lógica interna e também tem conhecimento dos conjuntos de entrada e saída.

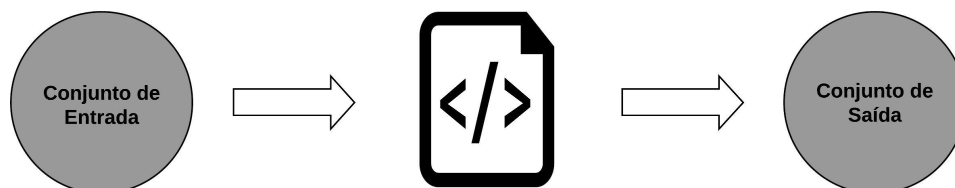

Figura 1 – Representação do teste estrutural (caixa-branca).

Fonte: Próprio Autor

Segundo (BARTIÉ, 2002) essa técnica de teste é conhecida por sua alta detecção, porém também é conhecida por ser mais difícil de se implementar. Como bem pautado pelo autor essa técnica requer um bom conhecimento da estrutura interna do sistema e um trabalho de análise da mesma para construção de casos de testes, o custo de se implementar essa técnica é bem mais alto, porém é marcada pela sua eficiência em encontrar erros pela boa construção de casos de testes.

2.2.1.2 Teste Funcional

Os testes funcionais ou testes de caixa-preta, diferentemente do teste estrutural, nessa técnica o engenheiro de software não tem conhecimento da estrutura lógica interna do sistema. De acordo com (PRESSMAN; MAXIM, 2016) essa técnica tem como o foco os requisitos funcionais do software para desenvolver os casos de testes, para isso são considerados somente os conjuntos de entrada e o conjunto de saída esperado do sistema. O objetivo desse teste é validar o aspecto funcional do software, tendo como base os requisitos funcionais do sistema.

Ainda de acordo com o autor, os testes funcionais possibilitam que sejam criados casos de testes como objetivo encontrar:

- Funções incorretas ou ausentes.
- Erros na interface.
- Erros em estruturas de dados ou acesso a bases de dados externas.
- Problemas de desempenho ou comportamento.
- Erros de inicialização ou término.

A Figura 2 exemplifica o teste funcional de acordo com a visão de um engenheiro de software, que no processo de criação dos casos de teste o mesmo não possui o conhecimento da estrutura interna do sistema. Para ele o software é visto como uma caixa-preta e para planejar os casos de testes somente é necessário ter o conhecimento dos conjuntos de estrada e saída.

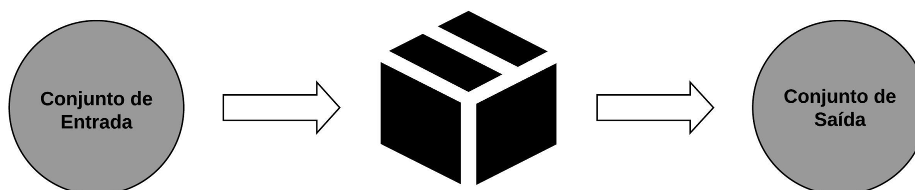

Figura 2 – Representação do teste funcional (caixa-preta).

Fonte: Próprio Autor

Segundo (BARTIÉ, 2002) o teste funcional, apenas valida se o sistema produz resultados esperados, não sendo necessário validar a sua estrutura interna. Com isso um dos benefícios dessa técnica é que ela não requer o conhecimento da tecnologia empregada ou dos complexos conceitos de implementação aplicados internamente, que diminui a dificuldade de encontrar profissionais capacitados para modelar os testes.

2.2.2 Níveis de Teste de software

Como bem conceituado por (NETO, 2007) o teste de software deve ocorrer em diversos níveis, sendo os mesmos contruídos em paralelo ao processo de desenvolvimento, para um melhor aproveitamento dos testes.

Podemos conceituar os níveis de testes de acordo com a visão de (PRESSMAN; MAXIM, 2016) e (SOMMERVILLE, 2011), os níveis podem ser definidos como:

- Teste de unidade: Também conhecido como teste unitário, tem foco em testar a menor unidade do sistema, que seria o componente ou módulo do sistema, o objetivo nesse nível é validar a lógica interna de processamento e estruturas de dados dentro dos limites do componente. Testes nesse nível podem ser conduzidos em paralelo para diversos componentes.

- Teste de integração: Os testes de integração visam testar a interação entre componentes do sistema através de suas interfaces. No desenvolvimento de software podem ocorrer problemas na comunicação entre componentes do sistema, dados podem ser perdidos através da interface, um componente pode ter um efeito inesperado ou adverso sobre outro, as subfunções quando combinadas podem não produzir a função principal, entre outros problemas. Os testes nesse nível, portanto, visam garantir que existe uma boa integração entre os componentes do sistema.

- Teste de sistema: A finalidade principal desse teste é exercitar o sistema como um todo. Nesse nível os componentes do sistema agora são integrados, formando um sistema completo. Assim os testes visam verificar se a integração dos componentes foi feita de forma adequada. Também é nesse nível que se verifica se o software atende aos seus requisitos, bem como é verificado seus atributos de qualidade. Nesse nível testes podem ser feitos de forma manual ou automatizada.

- Teste de regressão: Toda vez que um módulo novo é acrescentado ou o software é alterado, é necessário reexecutar os testes antes feitos para garantir que nenhum efeito indesejado foi introduzido no sistema. Em nível de sistema, para softwares de grande porte, é impraticável realizar todos testes de forma manual. Para esses casos é escolhido um subconjunto de testes para serem reexecutados. Nessa perspectiva os testes automatizados possibilitam que testes de regressão sejam mais eficientes, pelo baixo custo gerado para que esses testes sejam reexecutados.

Sendo assim, vemos que para a maioria dos níveis de teste é necessário que seja implementado uma certa automação dos testes que possibilite a execução dos mesmos. No caso dos testes de unidade e integração é necessário que sejam criados testes automatizados para viabilizar sua aplicação no sistema, diferente do teste de sistema que pode ser

executado de forma manual, mas que para sistemas de grande porte é importante que se tenha testes automatizados nesse nível para que seja praticável testes de regressão.

2.2.3 Testes Automatizados

De acordo com (SOMMERVILLE, 2011), o processo de testes em geral envolve uma mistura entre testes manuais e automatizados. Sendo que no teste manual é necessário que um testador execute o programa com um conjunto de dados específico e compara o seus resultados com suas expectativas. Por fim reporta as discrepâncias para os desenvolvedores do sistema. O teste automatizado por sua vez são codificados em um programa que é executado toda vez que o sistema é testado, essa forma é geralmente mais rápida que os testes manuais, principalmente quando envolve testes de regressão.

Os testes automatizados dão uma maior segurança e confiabilidade, principalmente nos casos em que se deseja evoluí-lo ou refatorá-lo, pois evitam que alterações no código causem efeitos indesejados. Na visão de (FOWLER, 2018) ter uma efetiva abordagem de teste, dá a equipe de desenvolvimento mais agilidade e confiança. A automação de testes permite que mudanças no código-fonte sejam feitas, sem preocupações em olhar a aplicação, por necessidade de verificá-la de forma manual.

Ainda nessa perspectiva os testes de software de forma automatizada garantem uma maior agilidade na entrega do produto, isso está alinhado com as práticas de métodos ágeis como o *Extreme Programming* (XP). Segundo (SOMMERVILLE, 2011) dentre as práticas do XP está a de desenvolvimento *test-first*, que consiste na utilização da escrita de testes automatizados para uma funcionalidade antes da mesma ser implementada, isso garante que toda funcionalidade criada terá pelo menos um teste fazendo a cobertura da mesma. Sendo assim teremos um código melhor testado com uma maior cobertura.

Os testes automatizados podem ser executados em diferentes níveis de teste, sendo difícil em muitos casos implementar todos em uma quantidade para um sistema de grande porte. Como dito por (COHN, 2009) testes automatizados podem ser considerados como testes caros e também podem demorar meses ou anos, para serem completamente implementados depois do código do sistema ser escrito. Isso acontece provavelmente, pois está sendo feito a automação de testes em quantidade e em níveis errados. O próprio definiu uma boa abordagem para implementação de testes em sistemas, chamado de pirâmide de testes.

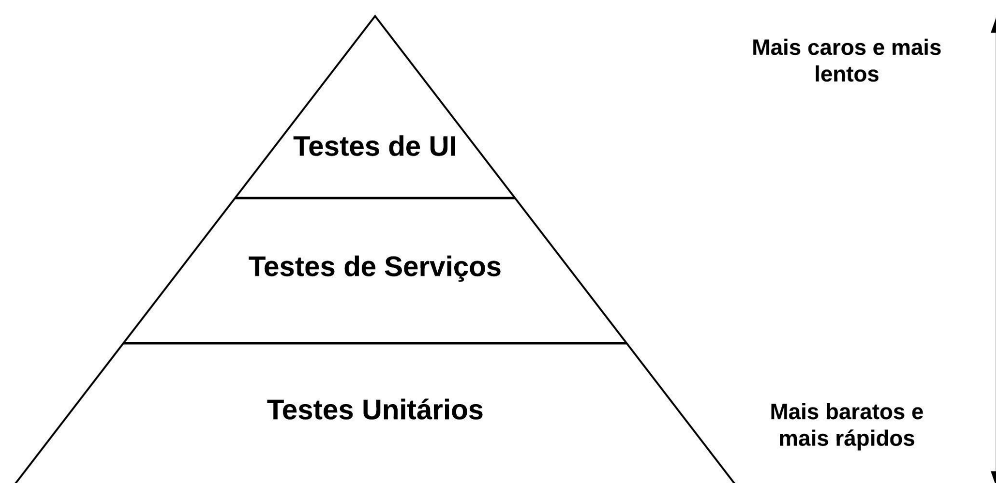

Figura 3 – Representação da pirâmide de testes de Mike Cohn.

Fonte: Próprio Autor

No primeiro esboço da pirâmide de testes representado pela Figura 3 são descritos os testes unitários, testes de serviços e testes de interface de usuário ou UI (do inglês *User Interface*). Os testes unitários já foram conceituados anteriormente, já os outros dois ainda não foram conceituados. Seguindo a visão de (COHN, 2009) e (FOWLER, 2018) podemos dizer que o propósito de cada um desses nível:

- Testes de UI: Os testes de interface, tem como objetivo exercitar o sistema pela interface do usuário, verificando se a mesma está se comportando como o esperado. Isso quer dizer que é necessário que através da interface, campos de entrada sejam preenchidos, ações sejam efetuadas e logo após é necessário comparar os resultados obtidos com o esperado e apresentado para o usuário.
- Teste de Serviço: Um serviço é algo que a aplicação faz em reposta a alguma entrada ou conjunto de entradas. O teste no nível de serviço tem como propósito testar os serviços da aplicação separadamente de sua interface.

Nesse modelo da pirâmide de testes dado por (COHN, 2009) é defendido que os testes unitários devem ser a base dos testes, logo devem representar a maior porção dos testes implementados por serem os tipos de testes mais baratos e mais rápidos de serem executados. Seguidos pelos testes de serviços que já são um pouco mais lentos e mais caros de serem implementados, portanto devem se encontrar em uma porção menor que os testes unitários. Por fim, por serem testes que geralmente são mais caros e mais lentos, os testes de UI devem ser encontrados em menores porções que os anteriores.

2.2.4 Testes de Ponta a Ponta

As aplicações modernas estão ficando cada vez mais complexas, como já mencionado anteriormente, segundo o autor (RIOS; MOREIRA, 2006) os processos que são automatizados por meio de softwares estão ficando mais complexos, que torna os mesmos também mais complexos. Com isso as técnicas e critérios de testes precisam passar por uma reformulação, para acompanhar esse aumento da complexidade dos softwares.

Uma abordagem atual para testar os sistemas modernos é o teste de ponta a ponta, ou como também são conhecidos testes E2E (acrônimo da expressão em inglês End-To-End). Segundo a visão de (CLERISSI et al., 2017) os testes E2E são uma abordagem relevantes para melhorar a qualidade dos sistemas web complexos, uma vez que exercita o sistema como um todo.

Na visão de (FOWLER, 2018) os testes E2E são parecidos com os testes de UI. De acordo com o autor os testes E2E de fato geralmente conduzem os testes pela interface do usuário, porém o contrário não é verdadeiro. Dependendo da tecnologia utilizada para se desenvolver a interface do sistema, os testes de interface podem ser realizados através de testes unitários.

Com esses conceitos podemos ver que os testes de E2E testam o sistema por completo, sendo necessário que toda infraestrutura e serviços que a aplicação necessite esteja disponível para a realização dos testes. O sistema então é testado de ponta a ponta, desde sua interface até sua integração com o banco de dados, serviços ou servidores. Esse conceito está representado na Figura 4. Importante conceituar que os testes E2E podem ser executados em qualquer dispositivo computacional: *desktop*, *notebook*, dispositivos móveis e etc.

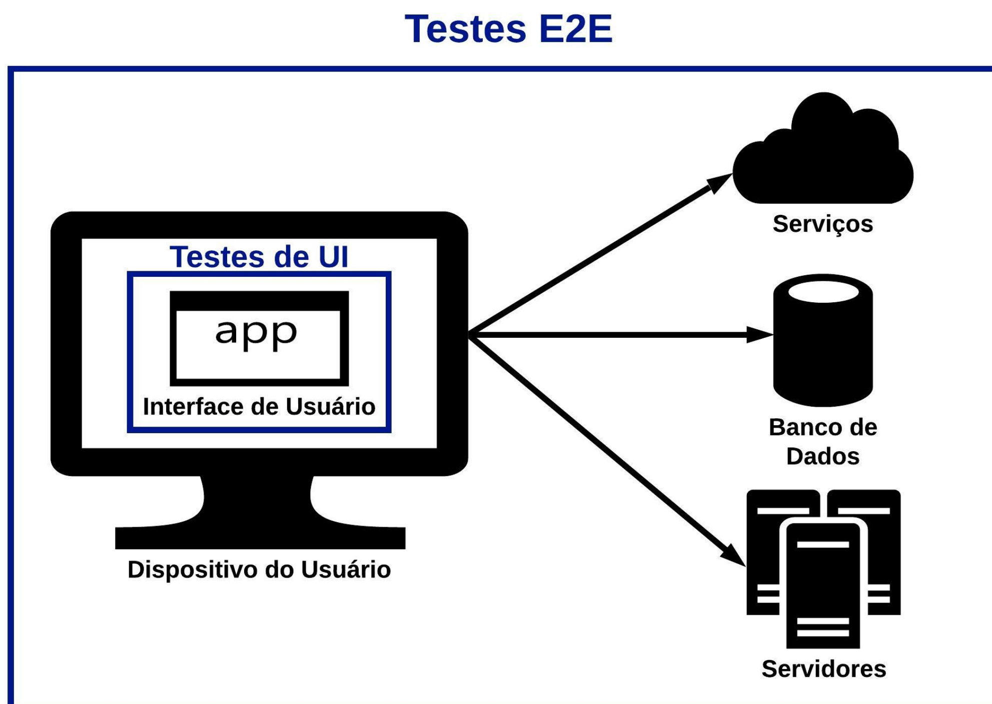

Figura 4 – Diferenças entre testes E2E e UI.

Fonte: Próprio Autor

Com isso a própria pirâmide de testes de Mike Cohn já passou por alterações desde que foi definida pela primeira vez, para se adequar a novas necessidades do desenvolvimento de software. O importante é que o conceito se mantenha para que a automação de testes não se torne um pesadelo, como (FOWLER, 2018) menciona em sua obra.

> “Atenha-se à forma da pirâmide para criar um conjunto de testes saudável, rápido e sustentável: escreva vários testes de unidade pequenos e rápidos. Escreva alguns testes de granularidade maior e poucos testes de alto nível que testam seu aplicativo de ponta a ponta.”

Ainda em sua obra (FOWLER, 2018) diz que não é necessário se manter apegado aos nomes das camadas da pirâmide de Mike Cohn, visto que dado a deficiência das mesmas é totalmente aceitável que elas sejam mudadas, desde que a nova definição se mantenha consistente na base de código da aplicação e nas discussões da equipe.

Em seu artigo (WACKER, 2015) define novas camadas para a pirâmide de testes, onde foi feito a substituição dos testes de serviços pelos testes de integração e também foi substituído os testes de UI pelo testes E2E. A pirâmide de testes definida no seu artigo, pode ser representada pela Figura 5.

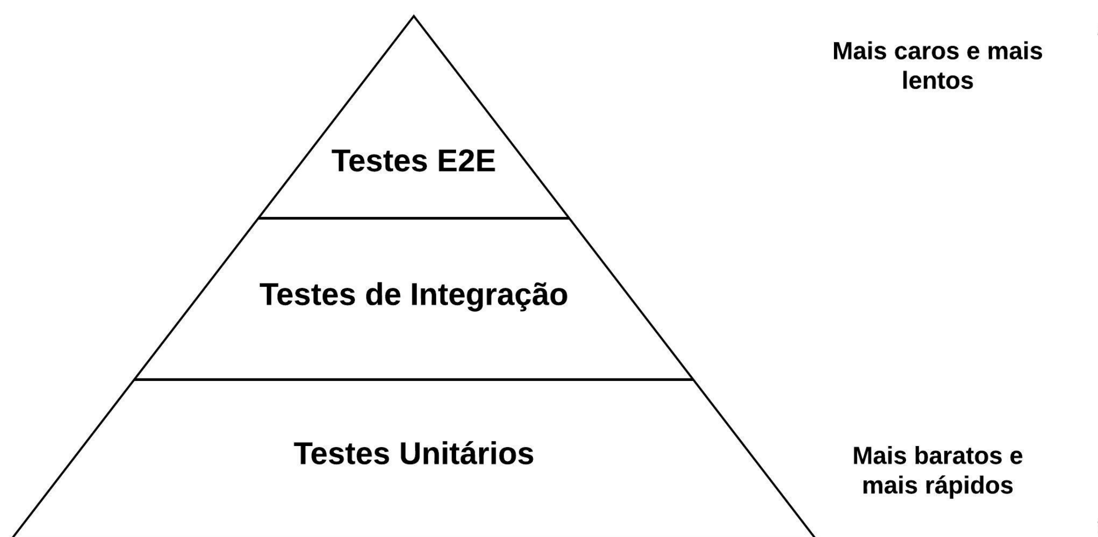

Figura 5 – Representação de uma pirâmide de testes com testes E2E.

Fonte: Próprio Autor

2.2.5 Verificação e Validação

A verificação e validação, ou também como é conhecido V&V, é um importante conceito dentro dos processos de software. Segundo a visão de (SOMMERVILLE, 2011, p. 27) o V&V tem a seguinte definção "verificação e validação (V&V), tem a intenção de mostrar que um software se adequa a suas especificações ao mesmo tempo que satisfaz as especificações do cliente do sistema.".

Para Barry Boehm (1979), citado por (SOMMERVILLE, 2011, p. 145), a validação se preocupa em responder a pergunta "estamos construindo o produto certo?", enquanto a verificação se preocupa em responder "estamos construindo o produto da maneira certa?". Os dois conceitos são bem parecidos, mas a distinção é que enquanto a validação se preocupa se o software atende as necessidades do cliente, a verificação se preocupa se o mesmo atende aos seus requisitos.

Ainda na concepção de (SOMMERVILLE, 2011) podemos dar as seguintes definições para os processos dentro de V&V:

- Verificação: Tem o objetivo de checar se o software atende aos seus requisitos funcionais e não funcionais.
- Validação: Tem o objetivo de garantir que o software produzido corresponde as necessidades do cliente. A validação é essencial, pois nem sempre as especificações de requisitos refletem os desejos dos clientes.

2.3 Ferramentas de Testes Automatizados

Segundo (BARTIÉ, 2002) a automação de testes é o emprego de ferramentas de testes que simulem usuários ou atividades humanas de forma que não seja necessário a execução de testes manuais com a intervenção humana. Vemos que por essa definição um dos pontos chaves da automação é a ferramenta de testes.

De acordo com (VELOSO et al., 2010), a escolha de uma ferramenta de testes inadequada pode ocasionar em desperdícios de tempo. Podemos interpretar seguindo essa visão que é imprecindível a escolha de uma boa ferramenta que atenda as necessidades do projeto, para que não haja desperdícios.

Existem muitas ferramentas disponíveis atualmente, cada uma com suas próprias peculiaridades, portanto é fundamental avaliar as vantagens e desvantagens de cada ferramenta.

Em termos de arquitetura podemos separar as ferramentas de testes em nível E2E de aplicações web em dois tipos, que ficará mais evidente ao ser abordado as diferenças arquiteturais, mas sucintamente podemos definir os dois tipos como sendo:

- As ferramentas que utilizam uma API (Application Programming Interface) para automatizar navegadores e assim tornar possível a escrita de testes. Um exemplo desse tipo de ferramenta é o Selenium.
- As ferramentas que não utilizam uma API para se comunicar com os navegadores, ao invés disso executam os comandos de automação dentro do navegador juntamente com a aplicação web. Um exemplo desse tipo de ferramenta é o Cypress.

O Selenium é uma ferramenta bastante popular, que segundo o estudo realizado por (VELOSO et al., 2010) é uma das ferramentas mais adequadas para a realização de testes funcionais automatizados. Muitas outras ferramentas são baseadas no Selenium, ou seja, são construídas sobre o Selenium, como mostrou os estudos de (FOWLER, 2018), só provando a grande popularidade da ferramenta.

Uma alternativa ao Selenium é a ferramenta Cypress. A mesma foi indicada para adoção no relatório técnico *ThoughtWorks Technology Radar* realizado por ThoughtWorks (2019). Nele o Cypress é caracterizado como uma ferramenta "pós-Selenium" que tem dado boas respostas na sua utilização pelos colaboradores da empresa.

O estudo será portanto focado nessas duas ferramentas, visto sua relevância dentro dos dois tipos de ferramentas de automação de testes em nível E2E de aplicações web.

2.3.1 Selenium

De acordo com informações do SeleniumHQ (2019), o Selenium possui um conjunto de ferramentas de testes, para suporte da automação de teste, cada uma focada em atender uma necessidade específica. As ferramentas que incorporam o Selenium são: Selenium IDE (do inglês *Integrated Development Environment*), Selenium RC (do inglês *Remote Control*), Selenium WebDriver e Selenium Grid.

Ainda segundo informações do site oficial da ferramenta, o projeto do Selenium surgiu em 2004, quando Jason Huggins estava realizando os testes de uma aplicação interna da ThoughtWorks, como solução para um processo de teste manual repetitivo. Então como solução desenvolveu uma biblioteca na linguagem de programação JavaScript, que permitia interagir com a aplicação e reexecutar os testes em vários navegadores. Essa biblioteca mais tarde se transformou no núcleo das ferramentas Selenium RC e Selenium IDE.

O Selenium causou grande impacto na época, pois nenhuma outra ferramenta permitia a automatização do navegador, com livre escolha na linguagem de programação para se escrever os testes automatizados.

Mais tarde no ano de 2006 um engenheiro da Google, James Sterwart começou a trabalhar em um projeto chamado WebDriver, que tinha como objetivo resolver as limitações do Selenium na época. No ano de 2008 esse projeto foi adicionado ao Selenium, criando uma nova ferramenta que substituiria o Selenium RC, sendo nomeado de Selenium WebDriver.

2.3.1.1 Selenium IDE

Segundo informações do SeleniumHQ (2019), o Selenium IDE foi desenvolvido como um *plugin* para os navegadores Firefox e Chrome. Essa ferramenta tem o propósito de gravar interações manuais feitas no navegador em que o seu *plugin* tiver instalado e assim gerar *scripts* de testes, facilitando a automação e também permitindo a reexecução das interações com a aplicação.

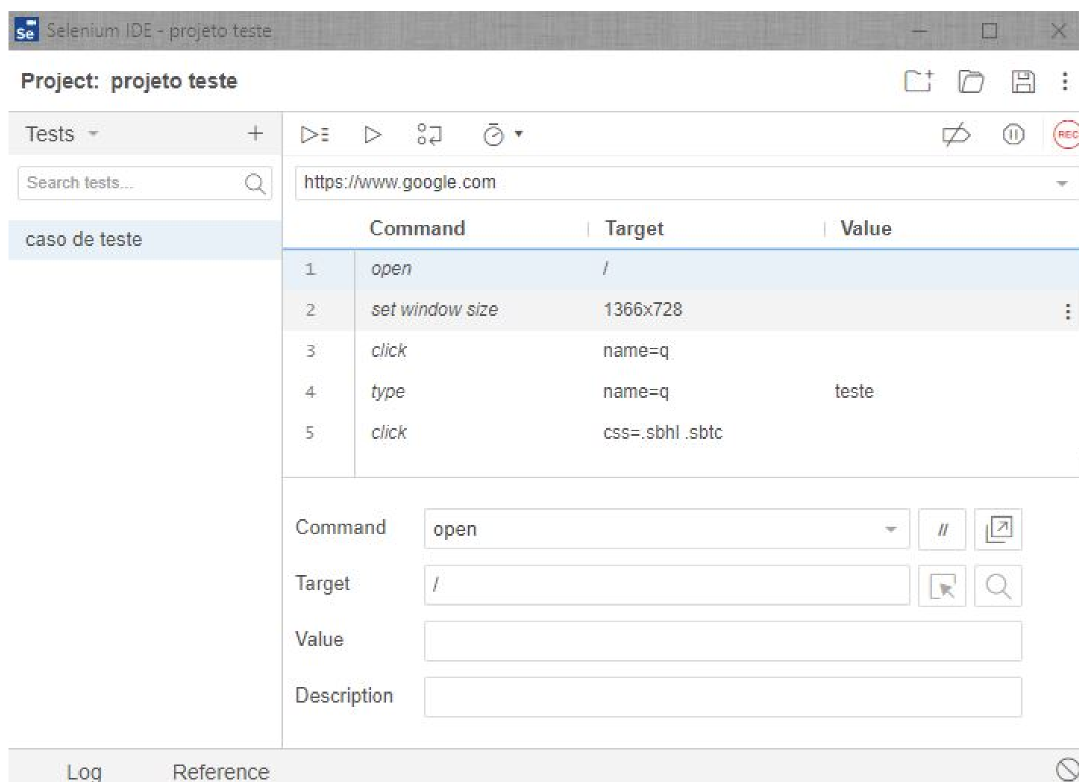

Figura 6 – Interface do *plugin* Selenium IDE para o Chrome.

Fonte: Próprio Autor

Através da sua interface exibida na Figura 6, o Selenium IDE permite que casos de testes sejam gravados, editados, executados e exportados como *scripts* de testes para uso nas outras ferramentas do Selenium como o WebDriver.

O Selenium IDE não foi projetado para realizar testes com asserções ou contruir todos os casos de testes, pois o mesmo não fornece iteiração ou instruções condicionais para os *scripts* de teste. O mesmo é definido simplesmente como uma ferramenta de prototipagem rápida. Para criação de testes robustos é recomendado utilizar outras soluções do Selenium.

2.3.1.2 Selenium RC

Por informações obtidas através do SeleniumHQ (2019), o Selenium RC como dito anteriormente, foi feito com base na biblioteca JavaScript desenvolvida por Jason Huggins. Com essa ferramenta é possível automatizar casos de testes escritos em variadas linguagens de programação.

A ferramenta tem dois principais componentes, o primeiro componente é o Selenium Server (Servidor do Selenium RC) e a *client libraries* (bibliotecas do cliente). O Selenium Server é o servidor responsável por inicializar e finalizar o navegador, bem como interpretar e executar os comandos do Selenium, também atua na interceptação da aplicação sobre teste (Application Under Test ou AUT). Já a biblioteca do cliente fornece uma interface entre o programa de automação de testes e o Selenium Server.

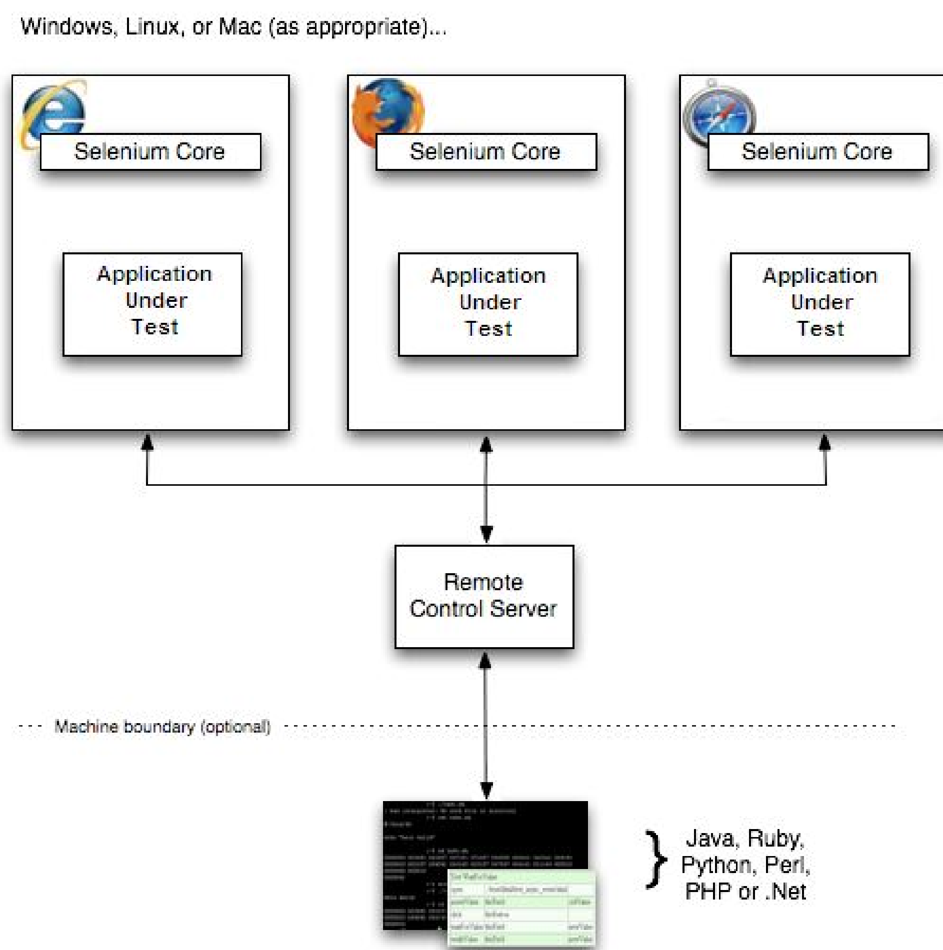

Figura 7 – Diagrama da arquitetura do Selenium RC.

Fonte: SeleniumHQ (2019)

A arquitetura do Selenium RC é representada pela Figura 7, as bibliotecas do cliente enviam comandos do Selenium para o Selenium Server, que por sua vez repassa os comandos para o Selenium Core (que é uma biblioteca de comandos em JavaScript), que é responsável por executar os comandos dentro do navegador e assim automatiza ações sobre da AUT.

As limitações do Selenium RC, são principalmente a deficiência em fornercer uma solução totalmente baseada no paradigma da orientação a objetos, lentidão na execução

dos comandos dentro do navegador, não suporte a automação em dispositivos móveis, não suporte a execução dos testes em um navegador sem interface visual.

O Selenium RC não é mais evoluído, está apenas sendo mantido, pois foi substituído por uma ferramenta ainda mais poderosa que é o Selenium WebDriver. Portanto não é recomendado o seu uso em novos projetos de automação de testes.

2.3.1.3 Selenium WebDriver

De acordo com informações do SeleniumHQ (2019), e como já foi dito anteriormente, o Selenium WebDriver surgiu como uma solução as limitações do Selenium RC. Criou-se então uma forma de automatizar ações no navegador sem precisar executar comandos através do Selenium Core. Para isso então o Selenium WebDriver faz chamadas diretas aos navegadores utilizando-se do suporte nativo de cada navegador para automação.

O Selenium WebDriver utiliza *drivers* de cada navegador para que possa diretamente executar os comandos do Selenium. O Selenium com essa nova abordagem, conseguiu superar as principais limitações do Selenium RC. O Selenium WebDriver conseguiu simplificar a arquitetura do seu antecessor, por utilizar uma abordagem diferente.

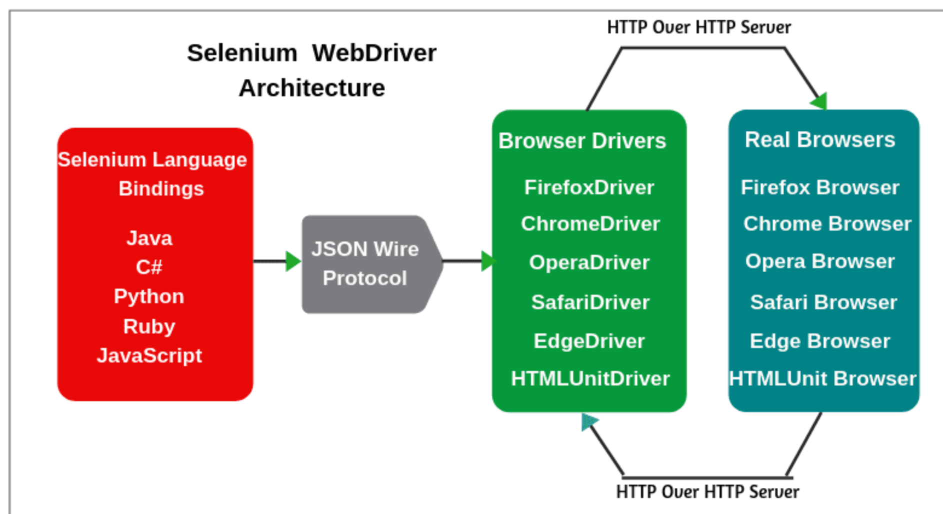

Figura 8 – Diagrama da arquitetura do Selenium WebDriver.

Fonte: Scientech Easy (2019)

Como representado na Figura 8, o Selenium WebDriver utiliza um protocolo de comunicação chamado *JSON Wire Protocol*, para estabelecer uma comunicação entre o programa de automação de testes e o *drivers* dos navegadores, que por sua vez automatizam os navegadores através de chamadas diretas ao navegador.

Vemos que o ponto chave do Selenium WebDriver é a utilização de *drivers* para estabelecer uma comunicação com os navegadores. Para cada navegador existe um *driver*

específico, sendo assim para automatizar os testes em diversos navegadores é necessário instalar esses *drivers* específicos para cada navegador em que se deseja executar os testes.

2.3.1.4 Selenium Grid

Segundo o SeleniumHQ (2019), a ferramenta Selenium Grid permite executar os testes em paralelo em múltiplas máquinas, com diferentes ambientes ou navegadores. Com essa solução é possível que o programa de testes feito com as ferramentas Selenium RC ou Selenium WebDriver sejam executados de forma mais rápida, pois são executados em paralelos em diferentes máquinas possibilitando inclusive a execução em máquinas virtuais.

Basicamente essa ferramenta é utiliza nas seguintes situações:

- Para executar os testes em múltiplos navegadores, em multiplas versões de navegadores, e em navegadores de sistemas operacionais diferentes.
- Para diminuir o tempo que os testes demoram para serem executados completamente.

2.3.2 Cypress

De acordo com Cypress (2019), a ferramenta Cypress tem o propósite de facilitar o trabalho de automação de testes de aplicações web. Com o Cypress é possível testar tudo que executa no navegador, isso se deve muito pela abordagem adotada pela ferramenta para automatizar testes.

Segundo (MACHADO, 2017) a primeira versão da ferrramenta foi liberada em 2015, mas seu lançamento oficial foi feito somente em 2017. Vemos que a ferramenta é bem recente e ainda está em fases de aprimoramentos.

Ainda de acordo com Cypress (2019), o Cypress é uma ferramenta que não se baseia no Selenium, ao invés disso tem uma arquitetura bem diferente do Selenium. A ferramenta possui recursos mais avançados em termos de automação de testes, pois graças a sua arquitetura, o mesmo possui mais controle de tudo que está sendo executado dentro do navegador.

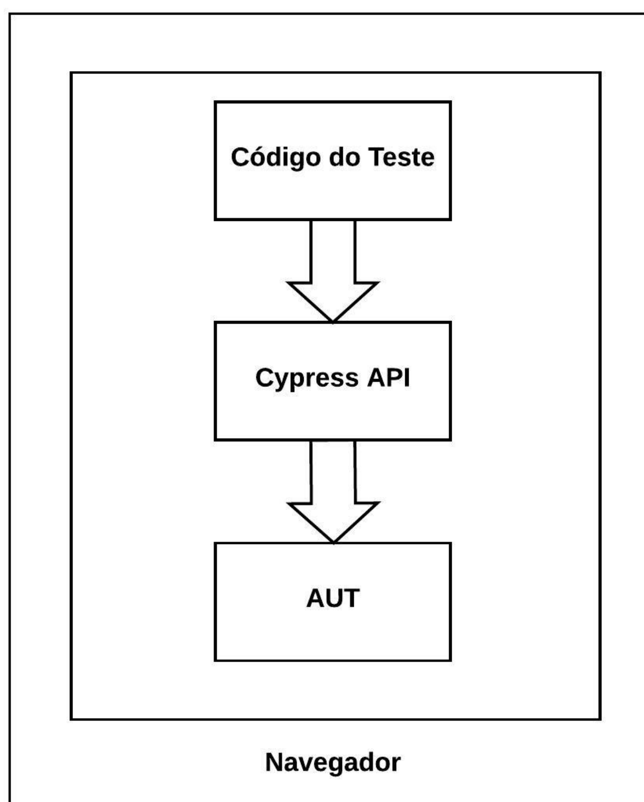

Figura 9 – Diagrama da arquitetura do Cypress.

Fonte: Próprio Autor

A arquitetura do Cypress é representada através da Figura 9, nela podemos notar que o Cypress funciona dentro do navegador, juntamente com a AUT. O código do testes faz uso da interface do Cypress API, que por sua vez é a responsável por executar comandos na AUT.

De acordo com Cypress (2019) o Cypress é executado sobre o Node (que é um interpretador de código em JavaScript). O código do teste também é escrito na linguagem JavaScript, não sendo possível até o momento escrever códigos em outras linguagens de programação.

Ainda de acordo com as informações da ferramenta, somente com a instalação do Cypress, já é possível escrever os testes, depurá-los e executá-los, fornecendo assim um ambiente de automação de testes completo.

2.3.3 Integração, Entrega e Implantação Contínua

Segundo a Red Hat (2019) para entregar aplicações com frequência aos clientes, se utiliza a automação de fases do desenvolvimento de aplicações e os principais conceitos relacionados a essa prática são os de integração, entrega e implantação contínua. O nome do

método utilizado nessa prática é o CI/CD (termo que vem do inglês *Continuos Integration, Continuos Delivery* e/ou *Deployment*).

Figura 10 – Diagrama representando o CI/CD.

Fonte: Red Hat (2019)

Como representado na Figura 10 o CI/CD possui diversos estágios, sendo cada um deles definido pela Red Hat (2019) como:

- *Continuous Integrations*: A integração contínua ou CI, serve para facilitar a integração de códigos, quando um time de desenvolvimento possui muitas pessoas trabalhando no mesmo código é comum que conflitos ocorram, e justamente no momento de se realizar a integração dos códigos alterados podem ocorrer problemas e a aplicação pode ser corrompida. Para isso o CI basicamente pega o código com mudanças consolidadas, validando através da criação da aplicação e execução geralmente de testes unitários e de integração. Sendo assim é dado uma maior confiança de que a aplicação não foi corrompida por alterações no código.

- *Continuous Delivery*: A entrega contínua ou CD, é uma etapa após o CI, serve para automatizar o lançamento do código validado pela etapa anterior em um repositório. Sendo o seu principal objetivo deixar o código pronto para ser lançando em ambiente de produção.

- *Continuous Deployment*: A implantação contínua ou também conhecida como outro CD, é a última etapa de um processo de CI/CD sólidos. O objetivo dessa prática é fazer com que o código validado e lançado no repositório pelos processos de integração e entrega, seja agora lançado no ambiente de produção. Na prática, as mudanças feitas pelo desenvolvedor podem ser disponibilizadas no ambiente de produção em poucos minutos, caso as alterações passem nos testes das etapas anteriores.

3 Metodologia e Características Analisadas

Neste capítulo são abordados as metodologias utilizadas para comparação das ferramentas de testes, bem como também elucida as características analisadas de cada ferramenta. As características aqui apresentadas serão necessárias para efeitos de comparação entre as ferramentas estudadas neste trabalho.

3.1 Metologia Adotada

A metodologia adotada foi a análise documental das ferramentas em estudo, dando maior foco nas características que são primordiais para as ferramentas de automação de testes. Diante dessa análise será feita uma comparação entre as ferramentas, que ressaltará suas características e contextos de utilização.

Para efeitos de análise, neste estudo serão consideradas somente as características que são crucias para a montagem de um bom projeto de automação de testes, as características que serão analisadas estão listadas abaixo:

1. Execução em sistemas operacionais.
2. Execução em navegadores.
3. Configuração do projeto de automação.
4. Linguagens de programação.
5. Recursos de escrita de testes.
6. Depuração dos testes.
7. Arquitetura das ferramentas.
8. Relatório de execução.
9. Documentação da ferramenta.
10. Possibilidade de uso em CI/CD.

Visto que essas características são importantes para a escolha de ferramentas de automação em um projeto de software, serão analisadas as diferenças de cada ferramenta e será montado um estudo comparativo entre elas.

3.2 Características Analisadas

Nesta seção serão abordadas as características e critérios de análise escolhidos para este estudo de ferramentas de testes E2E em aplicações web. Esta elucidação é necessária para justificar quais pontos foram analisados em cada ferramenta afim de extrair informações a respeitos de suas características.

3.2.1 Execução em Sistemas Operacionais

Para uma ferramenta de automação é importante saber em quais sistemas operacionais a ferramenta pode ser utilizada. Visto que hoje temos uma boa variedade de sistemas operacionais e distribuições diferentes em uso, é crucial que os testes feitos possam ser executados pela ferramenta em uma boa variedade de sistemas operacionais.

Um exemplo que podemos levar em consideração é a atual preocupação com usuários de aplicações móveis, antes da criação dos *smartphones* não havia motivos para se pensar em aplicações web sendo utilizadas em dispositivos móveis, mas com a evolução dos aparelhos esse tipo de usuário começou surgir e ficar em evidência. Atualmente é possível acessar páginas da web e ter aplicativos avançados em celulares, fazendo com que aplicações tenham que ser pensadas para esse tipo de utilização.

Portanto é importante saber o perfil do usuário da sua aplicação e saber se a ferramenta de testes consegue simular o usuário da aplicação da forma mais próxima da realidade. Como é falado por (SOMMERVILLE, 2011, p. 189) "A confiabilidade de um sistema depende do contexto em que ele é usado.", portanto é necessário avaliar o sistema nos mais diversos contextos possíveis.

Nesse aspecto serão considerados os principais sistemas operacionais da atualidade segundo a Western Governors University (2019). Portanto os sistemas operacionais que serão considerados neste estudo serão: Microsoft Windows, Apple macOS (ou macOS), Android OS (ou Android), Apple iOS (ou iOS) e Linux.

3.2.2 Execução em Navegadores

As aplicações web executam diretamente em navegadores, e como já mencionado anteriormente é importante ter a possibilidade de verificar o comportamento do sistema em diversos contextos, para aplicações web o contexto mais importante é o navegador em que se está sendo executada.

As ferramentas de automação de testes precisam oferecer a possibilidade de execução em diferentes navegadores de forma que simulem usuários reais. Para projetos que por regra de negócio é necessário a verificação do comportamento da aplicação em diferentes

navegadores, a avaliação desta característica é fundamental no momento da adoção da ferramenta de automação de testes.

Para estudo serão considerados os principais navegadores segundo o relatório mais recente da W3Counter (2019). Esse relatório pode ser visto na Figura 11. Portanto os navegadores considerados serão: Google Chrome, Safari, Edge, Internet Explorer, Firefox e Opera.

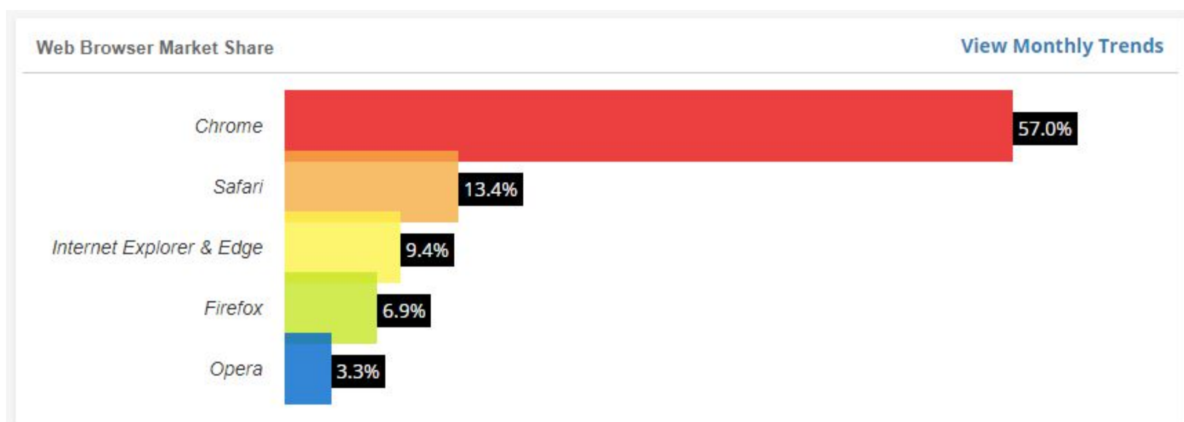

Figura 11 – Relatório de divisão de mercado entre os navegadores em 2019.

Fonte: W3Counter (2019)

3.2.3 Configuração do Projeto de Automação

Um dos aspectos das ferramentas de testes é a configuração do projeto, as ferramentas podem possuir dependências com tecnologias ou até mesmo com outras ferramentas. Além disso é necessário executar uma série de passos somente para configurar a ferramenta de testes em si para que ela funcione adequadamente.

Configurações complexas de ferramentas e que não são bem documentadas, são um verdadeiro incômodo para novos desenvolvedores que nunca tiveram contato antes com a ferramenta e suas tecnologias. Apesar de ser uma das partes mais chatas no desenvolvimento, é extremamente necessária para o bom funcionamento do projeto. Para algumas ferramentas é necessário escolher outras tecnologias ou *frameworks* para compor o projeto de automação, tornando o processo ainda mais difícil para desenvolvedores inexperientes.

Afinal de contas, a configuração do projeto de automação idealmente não deveria tomar muito tempo. Uma boa ferramenta facilita essa parte para que o desenvolvedor possa concentrar no que interessa, que no nosso caso é desenvolver os testes automatizados.

Neste quesito as duas ferramentas serão comparadas entre si de forma relativa, avaliando se a ferramenta depende de outras tecnologias para compor o projeto de automação e avaliando também o grau de complexidade de configuração relativa entre as ferramentas.

3.2.4 Linguagens de Programação

As linguagens de programação possibilitam que softwares sejam criados, por possibilitar a escritas de instruções que podem ser interpretadas por sistemas computacionais. No contexto de aprendizagem de linguagens de programação existem curvas de aprendizado específicas para cada linguagem, algumas sendo de fácil aprendizado e outras sendo mais difíceis. Por esse motivo comumente desenvolvedores escolhem se aperfeiçoar em uma linguagem de programação ou antes de começar o estudo de uma nova linguagem, avaliam o esforço necessário para se aprender uma nova linguagem.

Algumas ferramentas prevendo esse cenário, onde o aprendizado pode ser uma barreira para sua utilização, fornecem mais opções de linguagens para se trabalhar com a própria ferramenta ou então optam pelo uso de uma linguagem que seja de mais fácil aprendizado.

Neste contexto é importante a avaliação das linguagens disponíveis para escrever os códigos de automação de testes. Isso se deve à possibilidade de programadores já possuirem uma experiência prévia em alguma linguagem de programação que a ferramenta utiliza, que torna mais fácil o processo de aprendizado e utilização da ferramenta. Portanto quanto maior a variedade de linguagens que uma ferramenta disponibiliza para este tipo de aplicação em automação de testes, melhor a ferramenta é nesse aspecto.

A avaliação desse quesito será pela quantidade de linguagens de programação que a ferramenta disponibiliza para desenvolvimento dos testes. No caso serão consideradas para efeitos de comparação todas as linguagens de programação que as ferramentas estudadas permitem o uso, sendo elas: Java, Python, C#, Ruby, JavaScript, Kotlin.

3.2.5 Recursos de Escrita de Testes

O trabalho de escrita dos códigos de testes pode ser bem difícil sem o auxílio da ferramenta de testes. Recursos oferecidos pela ferramenta para auxiliar esse processo de escrita são importantes pelo fato de dar mais agilidade, para que o principal próposito da automação seja feito.

Recursos como *hot reload*, que serve para que após uma modificação no código fonte de um teste, os testes sejam reexecutados na aplicação com os comandos de automação atualizados. Essa abordagem possibilita que as alterações nos testes sejam verificadas de forma mais rápida e eficaz.

Outro recurso que pode auxiliar o processo de escrita é a ferramenta permitir o acompanhamento dos comandos de testes, conforme os mesmos sejam executados, possibilitando que o automatizador dos testes consiga ver em quais pontos o seu teste precisa melhorar ou até mesmo permite a correção mais fácil em casos de falha nos testes. Além disso é importante que a ferramenta possua uma interface agradável e bem intuitiva.

Um recurso bem útil que surgiu recentemente foi o chamado *time travel* que permite que seja visto o estado da aplicação no momento que foi executado um comando de teste específico, sendo possível visualizar seus elementos de interface no momento em que foi executado um comando passado. Isso é muito útil para investigar como está sendo executado um testes e possibilita que seja visto, em caso de falha, os possíveis problemas na aplicação ou dos testes no momento de execução.

Outro recurso vantajoso é o seletor de elementos que possibilita que os identificadores dos elementos visuais sejam capturados de forma mais fácil, os identificadores são importantes para que seja possível interagir com os mesmos através dos comandos de teste. Os identificadores são atributos dos elementos de uma aplicação web, que possibilita localizá-los de forma única na interface, muitas vezes são utilizados atributos do HTML como: id, class, name e etc.

Um recurso disponível em algumas ferramentas de testes é a espera automática de eventos, existe um certo problema com ferramentas mais antigas, que é a espera por eventos dentro do contexto da página. Pode ocorrer geralmente problemas de execução de um comando sem que o elemento alvo do comando esteja presente na tela da aplicação, pois os comandos podem ser executados antes de uma atualização que mostraria o elemento na tela da aplicação.

Para resolver esse problema de espera em ferramentas mais antigas é possível realizar diversas estratégias de espera, porém para realizar esse recurso é necessário identificar qual é a melhor abordagem de espera possível dentre várias opções existentes e também precisa aumentar a complexidade dos testes adicionando essas esperas. O recurso de espera automática faz com que essa complexidade não seja adicionada no código, pois a ferramenta espera automaticamente que elementos alvos estejam presentes na tela para então executar os comandos, tendo apenas um tempo *timeout* padrão, que por sua vez pode ser configurável.

Logo nesse quesito será avaliado a interface da ferramenta durante a execução dos testes, a disponibilidade do recurso de *hot reload*, a disponibilidade do recurso de *time travel*, a espera automática e o seletor de elementos.

3.2.6 Depuração dos Testes

A depuração dos testes é crucial para a manutenabilidade do código dos testes, é importante que se possa ter ferramentas para ver o estado da aplicação no momento que ocorre um erro. Os motivos para que esse quesito seja importante são simples, ao se escrever novos testes ou dar manutenção nos códigos antigos, ocorre por vezes falhas na execução do testes. Essas falhas do teste em aplicações web podem ser causadas por diversos motivos, sendo alguns deles por:

- Ocorrer erro na aplicação devido a alterações no código-fonte.
- Execução em ambientes diferentes do desenvolvido, isso pode estar relacionado com navegadores e OS.
- Configuração do projeto.
- Erros no código do teste automatizado.
- inconsistências na massa de dados utilizada para teste.
- Erros na ferramenta de automação.

Portanto é importante ter um dispositivo eficiente para detectar qual foi a causa da falha, e assim poder corrigir o mais rápido possível. Para isso as ferramentas geralmente oferecem meios de se depurar os testes realizados afim de encontrar problemas. Neste quesito as ferramentas serão comparadas relativamente para definir o nível em que esse recurso de depuração para rastreio de falhas é fornecido pela ferramenta.

No uso da ferramenta é possível que a depuração seja feita por recursos da própria ferramenta, ou da IDE na qual se esteja trabalhando. Portanto esse quesito vai avaliar se a ferramenta oferece algum recurso adicional para depuração, e se é possível realizar a depuração pela IDE.

3.2.7 Arquitetura das Ferramentas

A arquitetura das ferramentas será de importante debate, pois as diferenças nesse quesitos são importantes para definir o desempenho da ferramenta ao executar os testes, o nível de controle da ferramenta sobre o ambiente de teste e os seus limites, bem como conceituar sua complexidade arquitetural.

Para as ferramentas de testes E2E é essencial que o desempenho seja o máximo posssível, pois por natureza esses testes são relativamente mais lentos que os demais por estar em um alto nível. Por esse motivo será considerado falar sobre o desempenho das ferramentas devido sua arquitetura.

Outra coisa que é importante verificar é se a ferramenta permite que se tenha controle completo sobre os testes, podendo explorar todos os recursos da aplicação. Esse ponto deve ser bem avaliado quando se deseja realizar testes mais refinados.

Portanto essa característica irá avaliar a complexidade da arquitetura tendo como base as ferramentas Selenium RC, Selenium WebDriver e Cypress. Será considerado também se a ferramenta permite que seja feito testes com controle total do navegador onde se executa os testes E2E, também será avaliado o desempenho de cada ferramenta.

3.2.8 Relatório de Execução

O principal recurso que uma ferramenta de automatização de teste pode oferecer é um relatório de erros encontrados, que seja eficiente para encontrar a causa do erro. Esse é um ponto importante, pois não adianta muito uma ferramenta de testes detectar muitos erros e não ter muitos recursos para detalhar sobre os erros encontrados.

As ferramentas geralmente oferecem bons recursos nesse quesito, com vários artefatos que axliam na investigação dos erros. Afinal como dito antes as falhas ocorrem nos testes por vários motivos, por esse motivo é essencial ter um relatório que especifique e forneça um detalhamento sobre os erros encontrados.

As características dessas ferramentas serão avaliadas de forma relativa, comparando os recursos que o relatório de erros que cada ferramenta possui. Um bom relatório de ferramentas de testes em nível E2E costumam ter informações sobre os resultados dos testes após sua execução, as causas de falha dos testes, *screenshot* da tela após a execução dos testes e vídeos exibindo a execução dos testes.

3.2.9 Documentação

Uma boa documentação traz muita confiança e ajuda a equipe de desenvolvimento a conseguir entender a ferramenta e todos os recursos que ela possui. A documentação da ferramenta é crucial para que ela seja melhor aproveitada, visto que a mesma serve de apoio e ponto de partida para muitos, pois a documentação é vista como uma fonte confiável de informações.

Nesse quesito é importante que a documentação seja detalhada, possua exemplos e tenha uma cobertura de todos os recursos da ferramenta de forma organizada. Além disso espera-se que seja abordado também padrões e recursos da ferramenta, assim existe uma melhor garantia de que os padrões serão seguidos e que todos os seus recursos possam ser explorados de forma eficiente.

Além disso tudo é importante pensar na inclusão de pessoas na utilização da ferramenta, por isso as documentações precisam estar traduzidas e disponíveis em várias línguas. Sempre pensando na inclusão e na melhor transmissão de informação possível.

Nesse quesito portanto será avaliado se a documentação está detalhada nos conteúdos disponíveis, se a mesma está disponível em mais de uma língua, se ela possui todos os recursos que a ferramenta oferece no auxílio da atividade de automação de testes e também se oferece informações sobre todos os padrões que é aconselhado seguir com o uso da ferramenta.

3.2.10 Possibilidade de Uso em CI/CD

Para processos ágeis se costuma utilizar uma prática de CI/CD, que permite que aplicações sejam entregues de forma mais rápida para o cliente. Os testes automatizados nessa prática é algo essencial caso se queira entregar o software com qualidade de forma rápida.

Uma prática de CI/CD madura possui testes automatizados sendo exutados na aplicação, com o intuito de verificar se alterações no código fonte acarrateram em erros na mesma. Nesse cenário é importante que ferramentas de testes possam ser utilizadas nessa prática afim de contribuir com os testes.

Nesse quesito será avaliado somente se a ferramenta pode ser integrada à prática de CI/CD. Contribuindo assim para que seja criada uma prática de integração, entrega e implantação mais confiável.

4 Resultados e Discussões

Neste capítulo são abordados os resultados de análises das ferramentas, através de um estudo documental e bibliográfico. Serão consideradas informações providas pela documentação oficial de cada ferramenta.

4.1 Comparações

Nesta seção será feito algumas comparações e discussões sobre as características previamente mencionadas no Capítulo 3. Para cada característica teremos uma comparação que ajudará na conclusão de quais são os usos de cada ferramenta.

4.1.1 Execução em Sistemas Operacionais

As duas ferramentas possuem bons resultados nesse quesito, como mostrado através da Tabela 1. Podemos ver que nessa característica as ferramentas demonstram uma diferença apenas em alguns sistemas operacionais.

	Windows	**macOS**	**Linux**	**Android**	**iOS**
Selenium WebDriver	SIM	SIM	SIM	SIM	SIM
Cypress	SIM	SIM	SIM	NÃO	NÃO

Tabela 1 – Comparação de uso entre os principais OS.

Fonte: Próprio Autor

Isso se deve ao fato do Cypress ser uma ferramenta que foi criada com o propósito de testar aplicações web em sistemas operacionais de computadores pessoais. Aparelhos móveis não são o objetivo de utilização dessa ferramenta, segundo o Cypress (2019).

Apesar disso, as aplicações web quando são executadas em navegadores de aparelhos móveis possuem uma visualização diferente. Isso acontece pois muitas aplicações web estão responsivas pensando em sua utilização nesses aparelhos. O Cypress possui recursos para se realizar testes com a visualização da aplicação em aparelhos móveis. Para isso pode ser utilizado o *viewport*, que permite que seja feito verificações como se estivesse utilizando o navegador de um aperelho móvel. Porém o teste está sendo executado em um sistema operacional de um computador pessoal.

Com o Selenium já é possível fazer com que ele faça testes em aparelhos móveis, para isso é necessário utilizar *frameworks* específicos como o Appium. Com isso o Selenium

pode ter utilização no desenvolvimento de testes automatizados para aparelhos móveis, atingindo uma gama maior de sistemas operacionais em que o seu uso pode ser feito.

Por esse motivo o Selenium possui uma vantagem de poder ser utilizado em uma maior variedade de sistemas operacionais incluindo os que são projetados para aparelhos móveis.

4.1.2 Execução em Navegadores

Neste quesito fica mais evidente as diferenças entre as ferramentas, como mostrado na Tabela 2. Podemos ver que nessa característica o Selenium também possui uma vantagem de utilização sobre o Cypress, podendo ser utilizado em uma maior variedade de navegadores.

	Chrome	**IE**	**Edge**	**Firefox**	**Safari**	**Opera**
Selenium WebDriver	SIM	SIM	SIM	SIM	SIM	SIM
Cypress	SIM	NÃO	NÃO	NÃO	NÃO	NÃO

Tabela 2 – Comparação de uso entre os principais navegadores.

Fonte: Próprio Autor

A limitação do Cypress nesse quesito tem origem em sua implementação inicial. Sendo a ferramenta construída sobre uma implementação do *framework* Electron, que por sua vez se baseia no projeto *open source* Chromium, feito pela Google. Logo a aplicação do Cypress está sendo até o momento somente no Google Chrome e também no próprio Electron.

Nesta característica o Selenium tem uma vantagem, já que sua arquitetura favorece sua execução em uma variedade maior de navegadores, somente sendo preciso ter a implementação do *driver* correspondente ao navegador no projeto de testes.

Portanto nessa característica o Selenium possui uma vantagem sobre o Cypress, em relação a utilização da ferramenta em diferentes navegadores, além do navegador Google Chrome.

4.1.3 Configuração do Projeto de Automação

Neste quesito é possível notar pela Tabela 3 que o Cypress possui vantagens sobre o Selenium, por não depender de outras tecnologias ou ferramentas para compor o projeto de automação, também por sua baixa complexidade de configuração de projeto quando comparado com a complexidade de configuração do Selenium.

	Depende de Outras Tecnologias	**Complexidade de Configuração**
Selenium WebDriver	SIM	ALTA
Cypress	NÃO	BAIXA

Tabela 3 – Comparação entre as configurações das ferramentas.

Fonte: Próprio Autor

Como é conceituado em Cypress (2019), a ferramenta Cypress possui uma fácil configuração e por si só já engloba todas as tecnologias ou *frameworks* necessários para um projeto de automação de testes.

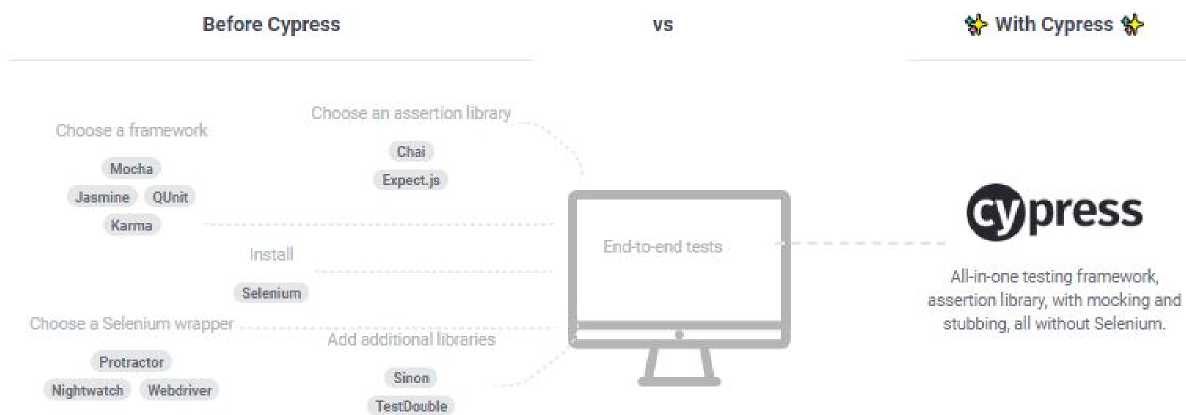

Figura 12 – Configuração de um projeto de testes com Cypress.

Fonte: Cypress (2019)

Como podemos observar na Figura 12, antes do Cypress era preciso escolher várias tecnologias para compor um projeto de testes de software. Isso tornava a configuração do projeto bem mais complexa e sendo necessário até mesmo analisar quais tecnologias iriam ser utilizadas para a composição do projeto.

Com o Cypress, a configuração do projeto fica muito mais simples, visto que o mesmo engloba todas as tecnologias necessárias para se realizar a atividade de automação de testes. O Selenium por sua vez precisa que outras tecnologias sejam englobadas no projeto para que ele possa ser aproveitado ao máximo.

Portanto vemos que o Cypress possui uma vantagem em relação ao Selenium que necessita que outras tecnologias sejam configuradas no projeto, por esse motivo também possui uma complexidade maior na sua configuração de projeto.

4.1.4 Linguagens de Programação

Neste quesito vemos através da Tabela 4, que o Selenium possui uma vantagem sobre o Cypress, pois possui uma maior variedade de linguagens de programação que podem ser utilizadas no projeto para a contrução dos testes automatizados.

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

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

Fonte: Próprio Autor

Pelas diferenças de arquitetura entre o Selenium WebDriver e o Cypress, que foram apresentadas no Capítulo 2, vemos que Selenium fornece mais opções de linguagens que podem ser utilizadas, por ter um desacoplamento das linguagens utilizadas.

O Cypress por sua vez foi construído sobre o JavaScript, que permite que ele seja executado dentro do navegador juntamente com a AUT. Isso permite que o Cypress inclusive possua mais controle sobre a automação, pois consegue manipular tudo que está sendo executado dentro do navegador.

Porém na comparação em relação as linguagens que podem ser utilizadas, o Selenium detém a vantagem, por justamente prover mais liberdade na escolha da linguagem, apresentando uma boa compatibilidade com uma quantidade razoável de linguagens de programação, sendo elas: Java, Python, C#, Ruby, JavaScript, Kotlin.

4.1.5 Recursos de Escrita de Testes

Neste quesito através da Tabela 5 vemos que o Cypress possui vantagens em recursos de escrita. Contando com todos os recursos analisados dentro do contexto de recursos que auxiliam a escrita de testes.

	Interface da Execução dos Testes	**Hot Reload**	**Time Travel**	**Espera Automática**	**Seletor de Elementos Via Interface da Ferramenta**
Selenium WebDriver	NÃO	NÃO	NÃO	NÃO	NÃO
Cypress	SIM	SIM	SIM	SIM	SIM

Tabela 5 – Comparação entre os recursos de escrita.

Fonte: Próprio Autor

4.1.5.1 Interface da Execução dos Testes

Esse recurso fornece ao automatizador informações sobre os testes que estão sendo executados, inclusive mostrando os erros quando falham e um detalhamento sobre as causas. Isso serve para orientar a criação dos testes e mesmo auxilia na correção de testes mal implementados. O Cypress possui uma interface própria que mostra o andamento da execução dos testes.

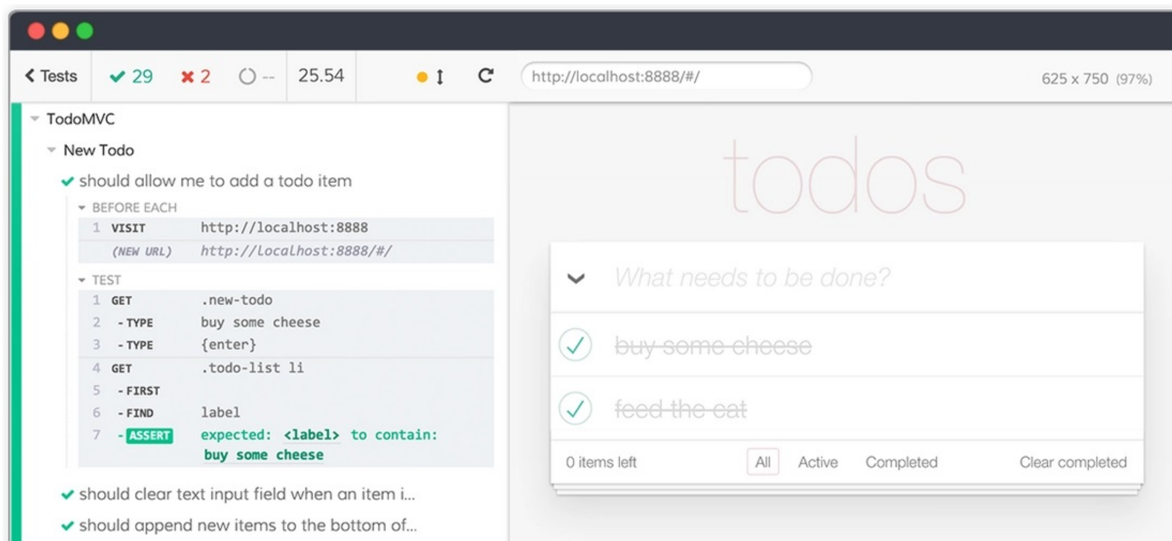

Figura 13 – Interface visual do Cypress sendo executado.

Fonte: Cypress (2019)

A Figura 13 mostra a interface da ferramenta Cypress, essa interface auxilia na criação dos testes pois mostra um detalhamento dos testes, além disso através dela é possível selecionar elementos em tela da aplicação, facilitando que os elementos sejam localizados.

A ferramenta Selenium não fornece uma interface própria, sendo necessária a integração com outras ferramentas para que a visualização da execução dos erros possa ser feita. Também pode ser feito uma visualização através da IDE, juntamente com algum *framework* de testes como o JUnit ou TestNG. Como o critério é se a ferramenta por si só oferece a visualização pela interface do usuário, então é considerado que o Selenium não possui esse recurso de forma nativa.

4.1.5.2 Seletor de Elementos Via Interface

Além disso na própria interface do Cypress é fornecido também um seletor de elementos da página web, de forma que facilita a escrita de testes, pois fornece de forma rápida e fácil o comando para interagir com o elemento.

Esse recurso não está disponível no Selenium. Para seleção de elementos no Selenium é necessário que faça-o diretamente no navegador, sendo ainda necessário conhecimento e experiência do desenvolvedor dos testes para a escolha da melhor forma de interagir com os elementos do sistema.

4.1.5.3 *Hot Reload*

O Cypress fornece os recursos de *hot reload* que consiste na atualização do teste logo após o arquivo de teste ser salvo, dispensando a necessidade de ter que manualmente executar novamente os testes.

Esse recurso não está disponível no Selenium. Também não foi encontrado nenhuma referência do recurso de *hot reload* na documentação oficial da ferramenta.

4.1.5.4 *Time Travel*

Outro recurso fornecido é o *time travel* que basicamente salva todos os estados da aplicação durante a execução dos testes e fornece através da interface a possibilidade de rever o estado da aplicação quando estava sendo executado um determinado comando. Isso auxilia no processo de entender como a aplicação estava ao ser executado o comando e assim agiliza o processo de escrita ou correção de erros.

Esse recurso não está disponível no Selenium. Bem como não foi encontrado nenhuma referência do recurso de *time travel* na sua documentação oficial.

4.1.5.5 Espera Automática

O Cypress possui esse recurso que auxilia muito a escrita de teste, isso está relacionado com o carregamento de elementos em uma aplicação web, ao realizar um comando sobre um elemento é feito uma espera automática pelo mesmo caso ele não esteja presente na aplicação web, isso considerando um *timeout* padrão que pode ser configurado

pela ferramenta. Caso se passe o tempo definido no *timeout* padrão é necessário verificar se ocorreu algum erro na aplicação ou implementação do teste, podendo ser possível definir um *timeout* específico no comando para usos isolados, caso tenha algum elemento que demore mais tempo para ser carregado e exibido em tela.

Esse problema de espera não é tão otimizado no Selenium, para que seja esperado um elemento surgir na tela, portanto é preciso fazer isso através da programação, utilizando alguma estratégia de espera. Portanto esse recurso não está disponível no Selenium.

4.1.6 Depuração dos Testes

Neste quesito através da Tabela 6 vemos as diferenças se sobresaindo entre as ferramentas, visto que ambas ferramentas possuem abordagens diferentes nesta característica em si.

	Opção de Depuração pela Ferramenta	**Opção de Depuração pela IDE**
Selenium WebDriver	NÃO	SIM
Cypress	SIM	NÃO

Tabela 6 – Comparação entre as formas de depuração.

Fonte: Próprio Autor

A depuração feita pelo Selenium depende da IDE em que está sendo escrito os testes, bem como varia de acordo com a linguagem de programação escolhida para o projeto. Não tendo em sua documentação SeleniumHQ (2019) nenhum registro de alguma opção de depuração que a própria ferramenta ofereça.

O Cypress por sua vez oferece uma opção de depuração próprio da ferramenta, sendo possível realizar a depuração a partir da própria ferramenta, segundo a documentação do Cypress (2019) a ferramenta é capaz de depurar tudo que está sendo executado dentro do navegador. A exibição da depuração é feita através do *console* das ferramentas de desenvolvedor dos navegadores, como é mostrado na Figura 14.

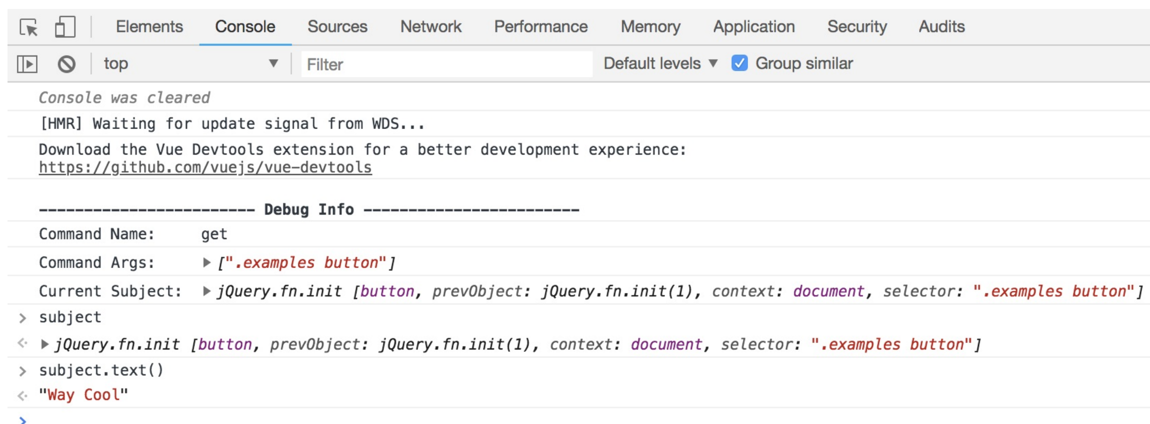

Figura 14 – Opção de depuração do Cypress no Google Chrome.

Fonte: Cypress (2019)

4.1.7 Arquitetura das Ferramentas

Neste quesito foi considerado o Selenium RC também para dar uma melhor noção de como foi a evolução dessa característica em cada ferramenta. Vemos através da Tabela 7 que houve uma evolução afim de diminuir a complexidade da arquitetura e providenciar o aumento do desempenho das ferramentas.

	Complexidade da Arquitetura	**Permite Controle Total do Navegador**	**Desempenho**
Selenium RC	ALTA	NÃO	RUIM
Selenium WebDriver	MÉDIA	NÃO	BOM
Cypress	BAIXA	SIM	ÓTIMO

Tabela 7 – Comparação entre as arquiteturas das ferramentas.

Fonte: Próprio Autor

A partir dos resultados é possível notar que as ferramentas evoluíram de modo que a complexidade da arquitetura fosse diminuindo, como discutido no Capítulo 2 existe uma diferença nítida entre as arquiteturas das ferramentas, sendo a do Cypress mais simples estando em uma proximidade maior com o navegador, visto que o Cypress funciona dentro do mesmo laço de execução da aplicação web. Essa arquitetura também permite que o Cypress tenha controle total de tudo que está sendo executado dentro do navegador no mesmo laço de execução, segundo Cypress (2019).

Essa arquitetura do Cypress também permite que o desempenho seja melhor, visto que o mesmo executa no mesmo laço de execução da aplicação, sua arquitetura simples

faz com que seu desempenho seja melhor que as demais ferramentas aqui destacadas. Isso tudo faz com que o Cypress possua um melhor resultado nesse quesito.

4.1.8 Relatório de Execução

Em relação a essa característica vemos por meio da Tabela 8 que as ferramentas não possuem tantas diferenças dentro desse contexto, sendo destaque apenas a diferença entre a geração de vídeos da execução de testes que o Cypress oferece a mais como recurso.

	Resultado dos Testes	**Causas de Falhas**	**Screenshot Após Execução dos Testes**	**Vídeos da Execução dos Testes**
Selenium WebDriver	SIM	SIM	SIM	NÃO
Cypress	SIM	SIM	SIM	SIM

Tabela 8 – Comparação entre os relatórios de execução.

Fonte: Próprio Autor

Neste quesito as duas ferramentas se equivalem bastante, pois seus relatórios mostram o resultado dos testes exibindo quais testes passaram e quais testes falharam, além disso em casos de falha também mostra a causa da falha do erro. Nesses dois pontos as duas ferramentas não variam muito e mantém uma consistência.

No recurso de *screenshot* após a execução dos erros, as duas ferramentas fornecem essa opção, entretanto no Cypress a configuração desse recurso é mais fácil e segundo informações do Cypress (2019) o processo de registro da *screenshot* é automático para testes que falham durante a execução. Ainda de acordo com a documentação a gravação de vídeos também é automático, não sendo preciso fazer configurações antes de executar os testes.

4.1.9 Documentação

Nesse quesito as ferramentas possuem uma boa diferença como mostrado pela Tabela 9, o Cypress possui uma melhor documentação em relação a documentação oficial do Selenium.

	Detalhada	**Disponível em Mais de Uma Língua**	**Possui Todos os Recursos da Ferramenta**	**Possui Todos os Padrões da Ferramenta**
Selenium WebDriver	SIM	SIM	NÃO	NÃO
Cypress	SIM	SIM	SIM	SIM

Tabela 9 – Comparação entre as documentação das ferramentas.

Fonte: Próprio Autor

Ambas ferramentas possuem uma documentação bem detalhada dos conteúdos que estão disponíveis, detalhando muito bem o uso de recursos e padrões da ferramenta. Os conteúdos da documentação servem muito bem como guia para implementações com a ferramenta nos mais variados contextos. Ambas as documentações também estão disponíveis em mais de uma língua, que mostra a preocupação na inclusão de mais pessoas no acesso as informações sobre as ferramentas.

Uma ressalva nos outros quesitos da documentação possuir todos os recursos e padrões, é que o Selenium está passando no momento por uma alteração drástica em sua documentação oficial SeleniumHQ (2019), portanto alguns tópicos ainda não estão completos, por esse motivo faltam algumas informações sobre a ferramenta. Portanto nesse quesito o Cypress (2019) possui uma documentação mais completa até o momento.

4.1.10 Possibilidade de Uso em CI/CD

Neste quesito as ferramentas não possuem divergências sendo possível o uso em práticas de CI/CD, os resultados dessa comparação podem ser observados na Tabela 10, onde mostra que não existem diferenças entre as ferramentas nessa característica.

	Uso em CI/CD
Selenium WebDriver	SIM
Cypress	SIM

Tabela 10 – Comparação entre o Uso em CI/CD.

Fonte: Próprio Autor

As duas ferramentas possuem a opção de executar os testes sem a necessidade de ser por meio de uma interface visual do navegador, sendo esse modo de execução denominado de *headless*, isso permite que os testes possam ser executados dentro de um *pipeline* de teste, que por sua vez fazem parte das práticas de CI/CD. Sendo assim não existem restrições na utilização das ferramentas dentro desse contexto.

5 Considerações Finais

Vemos portanto que as ferramentas aqui estudadas possuem aplicações distintas e muitas diferenças dentro dos critérios analisados. Sendo que o Selenium possui uma aplicação mais ampla podendo ser executado em diferentes ambientes, sendo utilizado em uma maior variedade de sistemas operacionais e navegadores, inclusive podendo ser aplicado em automação de testes em aparelhos móveis. Também oferece uma maior liberdade na utilização das linguagens de programação e tecnologias que podem ser utilizadas para compor o projeto de automação.

Por outro lado o Selenium também possui desvantagens em outros critérios como o de configuração do projeto por ter uma configuração mais complexa. Também possui menos recursos para auxiliar na escrita de testes e para identificação de problemas nos relatórios de teste. Também possui uma arquitetura que faz com que tenha um desempenho mais baixo que o Cypress e também não é possível ter controle total do que está sendo executado dentro do navegador.

O Cypress por sua vez possui as desvantagens de ambientes nos quais pode ser executado, sendo limitado a sua utilização em diferentes sistemas operacionais, também não sendo possível executar a ferramenta em sistemas operacionais de aparelhos móveis. Além disso a ferramenta possui uma limitação em relação a quantidade de navegadores em que pode ser executada.

Por outro lado é possível notar que a ferramenta Cypress possui mais recursos em relação ao Selenium em termos de escrita de testes, configuração, depuração e execução de testes. Possuindo inclusive uma melhor performance na execução de testes, devido sua arquitetura bem simplificada. Também vale salientar que possui melhores recursos na geração de relatórios de teste.

No contexto de testes automatizados em nível E2E para aplicações web, é notado que o Cypress possui mais recursos, que facilitam o processo de teste. Apesar de suas limitações a ferramenta consegue fornecer recursos bem mais atrativos que compensam seus pontos fracos, fazendo com que a atividade de automação de testes E2E seja mais fácil e eficiente.

Referências

BARTIÉ, A. *Garantia da qualidade de software*. [S.l.]: Gulf Professional Publishing, 2002. Citado 6 vezes nas páginas 12, 17, 19, 20, 21 e 28.

CLERISSI, D. et al. Towards the generation of end-to-end web test scripts from requirements specifications. In: IEEE. *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. [S.l.], 2017. p. 343–350. Citado na página 25.

COHN, M. *The Forgotten Layer of the Test Automation Pyramid*. Mike Cohn, 2009. (Acessado em 10/11/2019). Disponível em: <https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid>. Citado 2 vezes nas páginas 23 e 24.

CYPRESS. *JavaScript End to End Testing Framework*. [S.l.], 2019. Disponível em: <https://www.cypress.io/>. Citado 9 vezes nas páginas 33, 34, 44, 46, 48, 50, 51, 52 e 53.

FOWLER, M. *The Practical Test Pyramid*. Martin Fowler, 2018. (Acessado em 11/11/2019). Disponível em: <https://martinfowler.com/articles/practical-test-pyramid.html>. Citado 5 vezes nas páginas 23, 24, 25, 26 e 28.

GONTIJO, F. L. *Diretriz para a aplicação do teste ponta a ponta em uma aplicação web*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2019. Citado na página 19.

ISO/IEC. *ISO/IEC 25010. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. [S.l.], 2011. Citado 2 vezes nas páginas 18 e 19.

MACHADO, O. *Ferramentas de testes de aceitação: uma odisséia*. 2017. <https://medium.com/@otavio/ferramentas-de-testes-de-aceita%C3%A7%C3%A3o-uma-odiss%C3%A9ia-8eff4c96da9e>. (Acessado em 16/11/2019). Citado na página 33.

NETO, A. Introdução a teste de software. *Engenharia de Software Magazine*, v. 1, p. 22, 2007. Citado na página 22.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado 6 vezes nas páginas 12, 17, 19, 20, 21 e 22.

RED HAT. *O que é CI/CD?* 2019. <https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd>. (Acessado em 17/11/2019). Citado 2 vezes nas páginas 34 e 35.

RIOS, E.; MOREIRA, T. *Teste de software*. [S.l.]: Alta Books Editora, 2006. Citado 2 vezes nas páginas 12 e 25.

SCIENTECH EASY. *Selenium WebDriver Tutorial | Architecture & Benefits*. Scientech Easy, 2019. (Acessado em 16/11/2019). Disponível em: <https://www.scientecheasy.com/2018/12/selenium-webdriver-tutorial.html>. Citado na página 32.

SELENIUMHQ. *Selenium Documentation - Browser Automation*. [S.l.], 2019. Disponível em: <https://www.seleniumhq.org/docs/>. Citado 7 vezes nas páginas 29, 30, 31, 32, 33, 50 e 53.

SOMMERVILLE, I. *Engenharia de Software, 9a*. [S.l.]: Pearson Education do Brasil, 2011. Citado 8 vezes nas páginas 12, 13, 17, 19, 22, 23, 27 e 37.

THOUGHTWORKS, C. C. de Tecnologia da. *ThoughtWorks Technology Radar*. [S.l.], 2019. (Vol. 20). Disponível em: <https://www.thoughtworks.com/pt/radar>. Citado na página 28.

VELOSO, J. et al. Avaliação de ferramentas de apoio ao teste de sistemas de informação. *Anais do VI Simpósio Brasileiro de Sistemas de Informação (SBSI), Marabá, PA*, 2010. Citado na página 28.

W3COUNTER. *W3Counter: Global Web Stats - October 2019*. 2019. <https://www.w3counter.com/globalstats.php?year=2019&month=10>. (Acessado em 18/11/2019). Citado na página 38.

WACKER, M. Just say no to more end-to-end tests. *Google Testing Blog*, 2015. Citado na página 26.

WESTERN GOVERNORS UNIVERSITY. *5 Most Popular Operating Systems*. 2019. <https://www.wgu.edu/blog/5-most-popular-operating-systems1910.html>. (Acessado em 18/11/2019). Citado na página 37.