SELENIUM, ROBOT E CYPRESS: UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS DE AUTOMAÇÃO DE TESTE

Ana Carolina da Silva Ferreira [1]
Darielson Araújo de Souza [2]
Márcia Terezinha Tonieto [3]
Rodrigo Viana Castelo Branco [4]

RESUMO

Para melhorar a qualidade dos softwares, foram desenvolvidas ferramentas de automação de testes e redução de erros. Este trabalho visa avaliar, entre as ferramentas de teste atuais, qual se destaca quanto à velocidade e qualidade dos testes do lado do usuário em ambiente Web. As ferramentas escolhidas foram Cypress, Selenium WebDriver e Robot Framework. Aplicou-se um formulário eletrônico divulgado amplamente em redes sociais, contendo perguntas sobre a usabilidade, facilidade de instalação e preferência dos desenvolvedores sobre cada uma das ferramentas em estudo. Foram criados sete cenários de testes onde as ferramentas de automação foram rodadas avaliando a eficiência e eficácia dos softwares de testes escolhidos na ótica da ISO/IEC 25015:2014. Identificou-se preferência dos desenvolvedores pelo Cypress, devido sua usabilidade e facilidade de instalação. Quanto à eficiência nos testes, Cypress se destacou na facilidade e rapidez de aprendizado e instalação, seguido pelo Robot, que possui como desvantagem a curva de aprendizado maior, mas é extremamente eficaz. Por fim, o Selenium WebDriver é a ferramenta mais simples, porém menos robusta, não sendo recomendada para testes complexos.

Palavras-chave: Ambiente Web. Eficiência. Qualidade de Software. Testes.

ABSTRACT

To improve the quality of the software, test automation and error reduction tools were developed. This work aims to evaluate, among the current testing tools, which one stands out in terms of the peed and quality of user side tests in a Web environment. The tools chosen for the tests were Cypress, Selenium WebDriver, and Robot Framework. An electronic form widely publicized on social networks was applied, containing questions about usability, ease of installation and developers' preference for each of the tools under study. Seven test scenarios were created in which the automation tools were run, evaluating the efficiency and effectiveness of the test software chosen from the perspective of ISO/IEC 25015:2014. Developers' preference for Cypress was identified, due to its usability and ease of installation. As for efficiency in the tests, Cypress stood out in terms of ease and speed of learning and installation, followed by Robot, which has the biggest learning curve as a disadvantage, but is extremely effective. Finally, Selenium WebDriver is the simplest but least robust tool, and is not recommended for complex tests.

Key-words: Efficiency. Software quality. Tests. Web environment.

[1] Bacharel em Sistemas de Informação - Faculdade Lourenço Filho. E-mail: carol28ferreira@gmail.com
Versão adaptada do trabalho de conclusão de curso.
[2] Bacharel em Ciência da Computação - Universidade Estadual do Piauí - Mestre em Engenharia Elétrica e de Computação – Universidade Federal do Ceará - E-mail: darielson.souza@flf.edu.br.
[3] Bacharel em Ciência da Computação; Mestre em Computação – Universidade Estadual do Ceará – E-mail: marciatonieto@flf.edu.br.
[4] Bacharel em Engenharia da Computação – IFCE - Especialista em Engenharia de Software Estácio de Sá - E-mail: rodrigo.viana@flf.edu.br

1 INTRODUÇÃO

Os testes de software inicialmente eram feitos pelos progrmadores que desenvolviam e avaliavam seus programas. Na década de 60 o *debugging* passaou a ser utlizado como ferramenta para testar cõdigos, mas o trabalho era excessivamente manual exige muito tempo do desenvolvedor, um profissional altamente requisitado e de alto valor. Assim, no mundo com uma demanda cada vez maior por tecnologia, houve a necessidade de desenvolver softwares que substituíssem o trabalho do teste manual, que ajudasse a antecipar erros e falhas no projeto e ainda garantir a qualidade do software (MONITORA, 2019).

É notória a importância da automação de testes no âmbito do desenvolvimento de softwares. Todo programa precisa ser testado antes de ir para produção, garantindo que não passem erros que sejam prejudiciais na comercialização do produto ou que gerem riscos ao usuário final (BOAS; LOPES; SILVA, 2016).

Os testes automatizados também se mostram muito importantes no aspecto humano, pela impessoalidade nos testes realizados, e econômico, pois não há problemas envolvendo rotatividade de equipe e há uma redução de pelo menos 3 vezes o tempo gasto em testes manuais (CHICANELLI et al., 2019).

É necessário pensar nos testes em todo o processo de desenvolvimento, identificando que testes que necessitam de muitas repetições e alta complexidade merecem automatização, reduzindo o tempo gasto e garantindo a qualidade dos testes (ROMANINI, 2019).

Uma forma de garantir que os testes sejam realizados da forma mais otimizada possível é utilizando o conceito da Pirâmide de Testes, que define níveis para ajudar o desenvolvedor a realizar testes por nível de necessidade, por tempo dispendido e custo (ALMEIDA, 2020). Ela está dividida em testes de ponta a ponta, que é onde se imita o comportamento do usuário e tende a ser o nível de teste mais complexo e de mais alto custo; os teste de integração no centro, onde se verifica como os módulos do sistemas se comunicam entre si, como comunicação da aplicação com o banco de dados, por exemplo, e a base da pirâmide contempla os testes de unidade, onde se verifica a menor unidade de código da aplicação e tende a ser o nível mais rápido (CAMPOS, 2019).

Há uma série de ferramentas de testes que podem ser utilizadas, mas não há um consenso de qual ferramenta seria a melhor ou a ideal para cada caso. Logo, é interessante conhecer as diversas opções disponibilizadas pelo mercado (OKEZIE; ODUN-AYO; BOGLE, 2019).

Assim, este estudo se justifica na necessidade de conhecer as ferramentas de teste, seu desempenho quanto ao tempo e custo que geram, também pelo fato de não haver muitos trabalhos em língua portuguesa relacionados ao tema, fazendo sentido estudar o estado da arte e realizar a experimentação com um novo ponto de vista. A escolha das ferramentas se deu por serem familiares à autora deste estudo e, no caso do Selenium, por sua popularidade na comunidade.

Sabendo que as ferramentas de automação são indispensáveis para a manutenção da qualidade de software e que colaboram com a redução de possíveis erros e custos, este trabalho visa avaliar, entre as ferramentas de teste mais utilizadas, qual a que se destaca quanto à velocidade e qualidade dos testes do lado do usuário (de ponta a ponta) em ambiente Web.

A partir disso, foram definidos os seguintes objetivos específicos:

- Descrever três ferramentas de teste do mercado, com base na literatura;
- Realizar um estudo de caso avaliando na prática as três ferramentas de automatização com testes de ponta a ponta;
- Relatar os resultados encontrados e quais ferramentas possuem melhor desempenho com base nos testes realizados.

2 FUNDAMENTAÇÃO TEÓRICA

Abordaremos inicialmente a pesquisa exploratória em fontes bibliográficas a respeito do estado da arte do tema proposto. Iniciou-se com breves explanações sobre os trabalhos relacionados ao tema, a história dos testes, dos tipos de testes de software e sua importância.

2.1 Trabalhos Relacionados

Neste contexto são apresentados resultados de diversos autores que fizeram comparações similares a que foi realizada nesta monografia e que serviram de base para a sua execução. Cada autor apresenta seu ponto de vista quanto à qualidade das ferramentas com base no teste realizado.

Para Joshi (2020), o Cypress é focado em testes Front End e é considerado o mais utilizado por desenvolvedores Javascript por sua alta velocidade de execução. Porém, seu ponto negativo seria restringir a linguagem e o tipo de estrutura de teste que o desenvolvedor pode utilizar. Por outro lado, o Selenium é uma ferramenta multi linguagem e multi navegador, que não se restringe apenas aos testes, mas a outros propósitos.

Psujek, Radzik e Koziel (2021) utilizaram como critérios de qualidade o tempo de teste de cada ferramenta e a disponibilidade de documentação. Selenium revelou-se a melhor ferramenta em termos de tempo de execução do teste. Quando a documentação e suporte à comunidade o TestComplete, ferramenta paga, foi considerada a melhor, mas Selenium foi a segunda e Cypress a terceira nesse quesito. Por fim, concluíram de forma geral que cada ferramenta deve ser avaliada conforme a necessidade do desenvolvedor e do programa a ser testado.

Outro autor que concorda com a visão de que cada ferramenta tem suas vantagens, conforme o software a ser testado é Pakarinen (2020), que avaliou o desempenho de diversos frameworks, inclusive o Selenium, o Robot e o Cypress, realizando todos os testes, desde o desenvolvimento até o fim da aplicação. Ele fez uma análise mais profunda que a comparação que foi realizada nesta pesquisa, mas o processo realizado por ele serviu de base para a execução destes testes.

Okezie, Odun-ayo e Bogle (2019) realizaram uma comparação entre as ferramentas Selenium, TestComplete, Ranorex, Appium, Quick Test Professional, OpenScript, Janova, Rational Functional Tester, recomendando que, na seleção de qualquer ferramenta, sejam considerados fatores como o tamanho do projeto, o custo orçado para o teste e a plataforma do projeto. Na pesquisa desenvolveram um quadro comparativo indicando, conforme a literatura, se a ferramenta era aplicável a Desktop, Web ou Mobile, se é Open Source, se tem usabilidade e se exige muito conhecimento técnico. Entre outras conclusões, os autores acreditam que o Selenium é o mais vantajoso por ser open source e ideal para testes em Web.

Por outro lado, Mobaraya e Ali (2019) identificaram que o Cypress poderia ser uma ferramenta mais adaptada aos padrões atuais de projetos Web do que o próprio Selenium. Desenvolveram, então, uma pesquisa usando uma nova abordagem, o Test Driven Development (TDD) e no padrão de design Page Object Model (POM) com o Cypress como ferramenta de teste aplicada ao site do AliExpress, na conta de usuário e na finalização de pedido. Essa é a pesquisa mais parecida com o objetivo deste trabalho, cujos cenários de teste puderam ser aproveitados.

Uma abordagem muito interessante foi realizada por Pelivani e Cico (2021), que demonstraram a importância de utilizar tanto testes manuais como testes automatizados, ressaltando a necessidade de unir as duas ferramentas. Ele analisa os frameworks Selenium, Robot, Cypress, WebDriverIO e Gauge apontando vantagens e desvantagens quanto ao tempo de execução, geração de relatório, facilidade de configuração, necessidade de conhecimento em

programação e linguagem de programação. Sua pesquisa demonstrou que não existe um pódio, mas que cada ferramenta é ideal para situações específicas, indicando como é relevante que os desenvolvedores de testes conheçam diversas ferramentas e sua performance para decidir qual se aplica melhor a cada projeto.

Rollwagen et al. (2020) realizaram uma comparação entre as ferramentas BadBoy, Sikuli e Selenium, utilizando-as durante o desenvolvimento da aplicação web. Os autores disponibilizaram os casos de Testes em Testes de Funcionalidades; Tratamento com o Código, Tempos de Execução e Documentação. Definiram ainda como indicadores os testes concluídos sem erros, testes concluídos com erros e testes não concluídos, para então concluir a ferramenta mais indicada nesses casos, similar ao tipo de teste executado neste trabalho.

2.2 Tipos de testes de software

Os testes podem ser do tipo estáticos, quando se referem a revisões, percursos ou inspeções, ou dinâmicos, quando a execução do código acontece com uma coleção de hipóteses enquanto o próprio software está em execução (PELIVANI; CICO, 2021). Basicamente, nos testes estáticos não há execução do código, apenas uma avaliação visual do código, enquanto nos testes dinâmicos o código é rodado para averiguar seu funcionamento. Os testes dinâmicos são o foco deste trabalho.

Entre os testes dinâmicos, há testes que podem ser realizados tanto de forma manual como de forma automatizada e autores como Crispim e Gregory (2009), Monitora (2019) e Umar e Zhanfang (2019) descrevem esses testes:

Testes de caixa branca ou Testes unitários - Também conhecido como teste estrutural ou caixa de vidro, nos testes de caixa branca o profissional tem acesso ao código fonte e pode validar sua lógica, verificando o caminho, o fluxo de dados e se teria o resultado esperado. É basicamente um teste visual que exige muito conhecimento técnico e maior custo.

Testes de caixa preta ou Teste Funcional - Nele o testador não tem acesso ao código, focando seu trabalho em lançar inputs no sistema e verificar se os resultados alcançados estão conforme os resultados requisitados. Monitora (2019) afirma que é primordial testar também as formas como um usuário pode inserir dados no programa, como verificar se um campo aceita letras enquanto deveria aceitar somente números, por exemplo. Esse é um teste que pode ser automatizado.

Testes de regressão - Quando são necessárias mudanças de funcionalidade em um programa em desenvolvimento, pode acontecer algum comprometimento da lógica descrita anteriormente. Assim, o teste de regressão deve ser feito a cada mudança no código, mesmo que uma mudança pequena. Como pode ser automatizado e há

Teste de Usabilidade - Este teste visa conhecer o comportamento de um software ou plataforma do ponto de vista do usuário, verificando disposição de itens e botões, seu funcionamento em cada dispositivo e navegador, se está agradável aos olhos e se é de fácil entendimento.

Segurança - Os testes de software evoluíram muito com o tempo e um know-how maior foi adquirido pelos profissionais da área. Dentre os novos recursos utilizados para garantir o pleno funcionamento de um programa de computação estão os testes de segurança. Esse tipo de teste verifica a segurança do software no que diz respeito à proteção a ataques diversos a que pode estar submetido como hackers e vírus bem como na lida dos dados que são inseridos pelo usuário.

Integração - Nesse teste, em vez de se atestar funcionalidades do software, se analisa a integração entre as diferentes unidades que formam o sistema. São averiguados aspectos como a interface e a dependência entre os componentes.

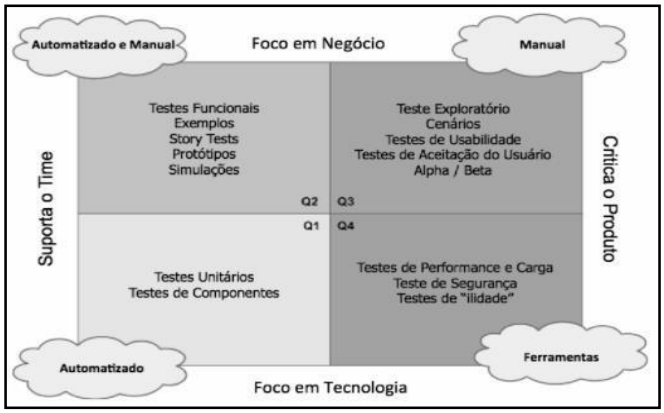
Teste de Performance ou Desempenho - O teste de desempenho determina como o software será executado em termos de capacidade de resposta e estabilidade, sob várias condições e carga de trabalho, se os componentes demoram para carregar, por exemplo. O software pode passar por testes de estresse e carga, resistência, pico, ponto de absorção, entre outros.

Teste de Instalação - Este teste verifica se, sob diferentes condições de hardware e software, o sistema pode ser instalado ou se haverá algum problema durante a instalação.

Teste de Manutenção - Os testes de manutenção precisam ser realizados para garantir que as atualizações promovidas em um sistema funcionem corretamente, garantindo que o software continue operante e não fique defasado.

Entre tantos tipos e profundidades de teste, a escolha de qual método utilizar se torna difícil, especialmente considerando a necessidade cada vez maior de formar times ágeis. Pensando nisso, Crispim e Gregory (2009) descreveram o Quadrante de Testes Ágeis (Figura 1) como uma forma de facilitar essa escolha.

Figura 1: O Quadrante de Testes Ágeis.



Fonte: Adaptado de Crispim e Gregory (2009).

A partir do quadrante é possível escolher com mais facilidade, conforme o foco do time de testes e de desenvolvimento. Por exemplo, no Quadrante 1 (Q1),o foco é o tipo de tecnologia e facilitar o trabalho do time de desenvolvimento, então há uso de ferramentas automatizadas para testes de unidade e componente.No Quadrante 2 (Q2) há um foco maior no ponto de vista do usuário, onde a funcionalidade do software é testada, podendo ser realizada de forma tanto manual quanto automatizada (CRISPIM; GREGORY, 2009).

Quando há necessidade de garantir que o produto desenvolvido vá atender o cliente, mesmo após rodar testes automatizados, é necessário recorrer a testes manuais, rodando a aplicação da mesma forma que o usuário faria. É isso que o Quadrante 3 (Q3) representa, há um foco na necessidade prática do usuário final e na necessidade de criticar cada detalhe do sistema desenvolvido. Por fim, o Quadrante 4 (Q4) apresenta os testes mais técnicos, que exigem conhecimento de ferramentas que podem simular situações de alta carga para o software, focando não mais no usuário, mas exclusivamente na qualidade técnica (CRISPIM; GREGORY, 2009).

2.3 Qualidade de Software e a importância dos testes

Behavior Driven Development (BDD) é Desenvolvimento Orientado ao Comportamento, ou seja, é uma técnica de desenvolvimento de software ágil que leva o time de desenvolvimento a pensar primeiramente no comportamento do usuário para entender o que deve ser feito. Para isso, são criados testes de cenários, comumente chamados de cenários BDD, aplicados a cada funcionalidade do sistema (MARQUES; FERNANDES, 2020).

Essa técnica foi desenvolvida por Dan North enquanto dava aulas de Test Driven Development (TDD ou Desenvolvimento Orientado a Testes) para alunos desenvolvedores. Pelo fato de muitos alunos demonstrarem dúvidas frequentes sobre o TDD, North percebeu que poderia adaptar a técnica de forma que, não só seus alunos, mas qualquer desenvolvedor, testador, analista ou profissional de negócios pudesse compreender, acabando com uma barreira de comunicação que existia. Assim, com a ajuda de seu colega Chris Matts, reformularam o TDD incluindo as boas práticas já conhecidas e o novo conceito (CEVERINO; NASCIMENTO, 2016).

Graças a evolução promovida por North, Marques e Fernandes (2020) puderam demonstrar que os cenários BDD são relativamente simples de serem elaborados, visto que podem ser escritos em linguagem próxima da linguagem natural, como exemplificado no cenário da Figura 2 a seguir:

Figura 2: Cenário BDD de listagem de posts em um blog.

```
#language: pt
#encoding: utf-8

Cenario: Listar todos os posts
  Dado que existam 2 posts criados
  Dado que eu esteja na pagina inicial
  Quando eu clicar no link para "posts"
  Entao eu devo ver a lista de postagens com 2 itens
```

Fonte: Marques e Fernandes (2020).

Essa linguagem na qual os cenários BDD são escritos tem como objetivos a documentação e a automação de testes. Essa linguagem é o Gherkin que se assemelha a um Caso de Uso e, como é perceptível na figura anterior, segue a ordem: título da funcionalidade, descrição da funcionalidade e cenários, ou seja, passos da interação entre usuário e sistema (CEVERINO; NASCIMENTO, 2016).

2.4 Selenium

O Selenium é um conjunto de ferramentas de teste funcional e é mais utilizado no mundo por sua confiabilidade, já que possui mais de 10 anos no mercado e tem uma comunidade forte para apoiar novos desenvolvedores. Porém, isso também pode ser uma desvantagem, já que, sendo mais robusta e antiga, torna-se difícil de integrar com outras ferramentas mais modernas e apresenta limitações ao lidar com elementos dinâmicos de sistemas web mais modernos (MOBARAYA; ALI, 2019; UMAR; ZHANFANG, 2019).

Selenium utiliza a abordagem rec-and-play (Record and Playback), ou seja, simula um usuário acessando o programa, onde a ferramenta grava scripts, os observa e reproduz. Os scripts de testes são gerados em Java, Ruby, C#, Pearl, PHP, HTML e Python e podem ser rodados em vários navegadores (SILVA, 2016, PAKARINEN, 2020).

Entre as principais vantagens do Selenium estão o fato de ser open source, suporte às principais linguagens, como Java, C#, PHP, Python, Ruby; permite a execução dos testes em qualquer navegador com suporte a Javascript; suporta diversos sistemas operacionais, permite debug dos scripts de teste e utilização de breackpoints e também a realização de testes de sistema e testes de regressão (SILVA, 2016). O autor lembra o fato de o Selenium se dividir em quatro ferramentas, o Selenium IDE, que é o ambiente integrado para construção de casos de teste e funciona como extensão do Firefox e é a parte responsável pela gravação de ações; o Selenium Remote Control (RC), que dispõe de uma API (Application Programming Interface) e bibliotecas para cada uma das linguagens de programação; o Selenium WebDriver, que é que a junção do Selenium RC com o Selenium IDE que pode ser integrado com outras ferramentas de teste, como o JUnit e, por fim, o Selenium GRID, torna possível a execução dos testes em várias máquinas ao mesmo tempo, reduzindo o tempo gasto, especialmente em conjuntos de testes grandes.

Para este trabalho foi utilizado o Selenium WebDriver por sua curva de aprendizado menor e por ser o mais indicado pelos autores dos estudos supracitados para testes simples e em ambiente web.

2.5 Robot

Desenvolvida inicialmente na Nokia Networks, com o código liberado em 2008, o Robot Framework é uma estrutura de automação genérica que pode ser usada para automação de teste e automação de processo robótico (RPA). Lançado sob a Licença Apache 2.0 e com a maioria das bibliotecas e ferramentas do seu ecossistema também de código aberto, seus recursos podem ser estendidos por bibliotecas implementadas com Python ou Java. É independente do sistema operacional e do aplicativo e sua estrutura é implementada usando Python e também é executada em JVM e .NET (ROBOT, 2021).

O sistema Robot Framework pode ser executado a partir da linha de comando e a criação dos testes é simples, basta que o desenvolvedor crie um arquivo terminado por robot no qual os testes são escritos na linguagem de programação Python e usando a sintaxe do Robot Framework. Quando a execução é bem sucedida, ele mostra relatórios com listam sucessos, falhas, tempo de teste e etapas de teste (PAKARINEN, 2020).

O Robot é de código aberto, é muito fácil de instalar e ajuda na criação e execução de casos de teste, pois graças a sua estrutura de palavras-chave, não precisa de grande conhecimento técnico para ser utilizado. Além dos casos de teste orientados por palavras- chave (por exemplo, para abrir o navegador, a palavra-chave usada é “Open Browser”), ele também tem suporte para testes por comportamento e por dados, além de um bom suporte para bibliotecas externas, sendo a Selenium Library a mais utilizada (TUTORIALS POINT, 2021).

Entre as principais vantagens de uso do Robot, estão o fato de ser open source, de sua documentação ter muita clareza e bastante conteúdo e também pelo fato de ser de fácil aprendizagem em comparação com o Cypress e ter suporte para o Chrome, Edge, Firefox, Safari, Opera e Internet Explorer (MALM, 2020).

2.6 Cypress

Cypress é uma solução de automação de testes em JavaScript. O Cypress, diferindo-se do Selenium e do Robot que são multiplataformas, ele foi desenvolvido para simplesmente executar testes em front-end de forma consistente e em pouco tempo, pois seus comandos de automação são principalmente síncronos e na memória (JOSHI, 2020).

De acordo com a Cypress (2021), é possível utilizá-lo para Testes ponta a ponta, Testes de integração e Testes Unitários, testando qualquer coisa que seja executada em um navegador. Por trás do Cypress está um processo de servidor Node. O processo Cypress e o Node se comunicam, sincronizam e realizam tarefas de um para o outro constantemente. Ele é instalado localmente, o que permite que ele faça capturas de tela e vídeos dos testes e falhas. Há uma série de vantagens no Cypress que atraem os desenvolvedores, especialmente seus recursos especiais (CYPRESS, 2021):

> **Viagem no tempo** - O Cypress tira fotos enquanto os testes são executados e é possível acessá-los passando o mouse sobre os comandos no Log de comandos.
>
> **Depurabilidade** - Permite depuração diretamente de ferramentas como “ferramentas do desenvolvedor”.
>
> **Espera automática** - Função do Cypress que entra em modo de espera automaticamente até que algum comando novo seja incluído.

Spies, Stubs e Clocks - Verifica e controla o comportamento das funções, respostas do servidor ou temporizadores.

Controle de tráfego de rede - Permite controlar e testar facilmente casos extremos sem comprometer o servidor.

Capturas de tela e vídeos - É possível visualizar capturas de tela tiradas automaticamente em caso de falha ou vídeos de todo o seu conjunto de testes quando executado a partir da linha de comando.

Entre outras vantagens do Cypress, além da velocidade de execução, está a possibilidade de ser executado nos navegadores Chrome, Edge, Firefox; sua documentação traz muitos exemplos de aplicação, facilitando o entendimento dos iniciantes na ferramenta, mas exige conhecimento em desenvolvimento. Por fim, o campo Log possui um recurso de histórico que permite testar facilmente cada revisão por etapas, otimizando o processo de correção de falhas (MALM, 2020).

3 ESTUDO DE CASO

Neste tópico serão apresentados os métodos utilizados para avaliar a qualidade dos softwares de testes escolhidos, o Selenium, o Robot e o Cypress, na ótica da ISO/IEC 25015:2014 e também na óptica dos desenvolvedores que utilizam essas ferramentas, por meio de um Formulário Google. O formulário visou verificar a preferência e experiência de desenvolvedores quanto às ferramentas desse estudo.

3.1 Procedimentos metodológicos

O questionário continha quatro perguntas sobre facilidade de instalação, ferramenta mais utilizada pelo desenvolvedor, qual a de melhor usabilidade e qual ferramenta seria a última escolha do profissional de testes. Ficou disponível durante o mês de outubro de 2021 e foi divulgado em redes sociais, especialmente em grupos com alta presença de profissionais de qualidade de software com registro de 32 participantes. Quando o formulário foi encerrado, foi realizado o tratamento dos dados e foram gerados gráficos para facilitar o entendimento dos resultados.

Para realizar os scripts de teste foi escolhido o site da Faculdade Lourenço Filho e utilizadas as ferramentas Selenium, Robot e Cypress. A escolha se deu por serem consideradas as melhores e mais utilizadas ferramentas de automação de testes, conforme os autores citados neste trabalho.

Quadro 1: Cenários de Testes.

Teste	Descrição
1	Verificar o funcionamento de cada item do menu principal
2	Verificação dos campos do formulário de egresso
3	Verificação das publicações da flf
4	Verificação do campo pesquisar
5	Verificação dos botões desktop e mobile
6	Login no aluno online desktop
7	Login no aluno online mobile

Fonte: Autora.

3.1.1 Resultados da pesquisa do Formulário Google

Foi solicitado no formulário que cada participante selecionasse respectivamente: a ferramenta considerada de instalação mais fácil, a ferramenta mais utilizada para os testes do respondente, a ferramenta de melhor usabilidade e a ferramenta que o desenvolver deixaria como última opção de escolha. Essas perguntas objetivas foram pensadas como parte da avaliação nos moldes da ISO/IEC 25010:2011 e também por determinar respostas facilmente analisáveis.

Com as respostas dos desenvolvedores foram criados gráficos para facilitação de entendimento dos resultados. Quase metade dos participantes (43,75%) afirmou que o Cypress é a ferramenta de mais fácil instalação, seguido pelo Selenium (34,28%) e o Robot (18,75%). Apenas 3,13% escolheram outra opção de ferramenta não listada. Esses dados foram representados na Figura 3 a seguir.

Figura 3: Gráfico de instalação mais fácil.

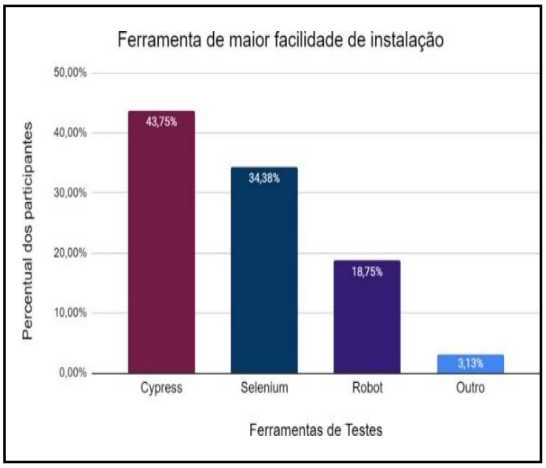
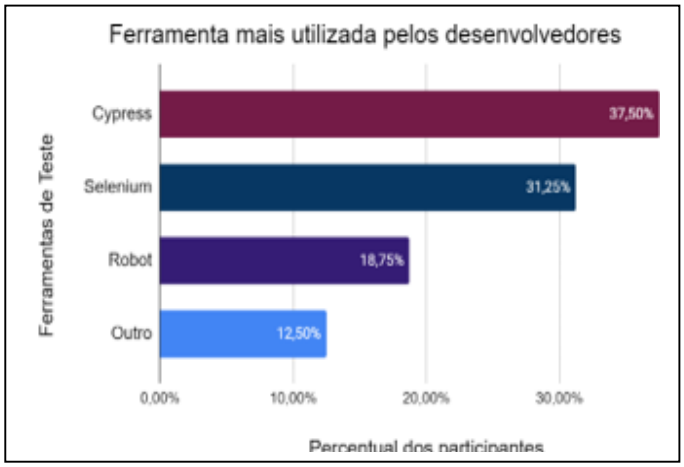


Figura 4: Gráfica de Ferramenta mais utilizada.



Fonte: Autora.

A segunda pergunta do questionário foi voltada para as ferramentas mais utilizadas pelos profissionais. Novamente o Cypress e o Selenium se destacaram como preferidos, com 37,50% e 31,25% dos votos respectivamente, representada na Figura 4 (acima).

Quando se trata de usabilidade, há uma pequena mudança na escolha dos desenvolvedores. Apesar de o Cypress permanecer como posição de mais vantagem, com 40,63% de escolha, o Robot apareceu nesse ranking em segundo lugar, com 28,13% de escolha, como na Figura 5.

Figura 5: Gráfico de melhor usabilidade.

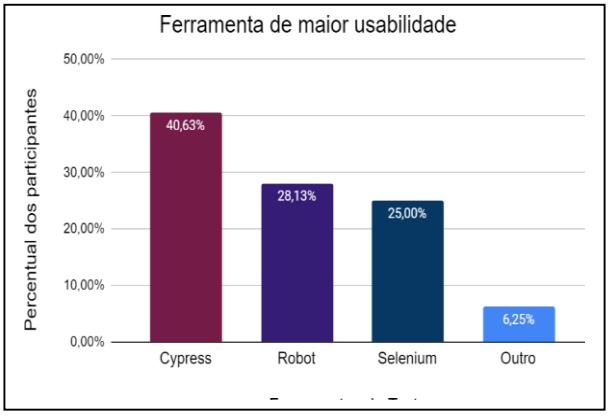
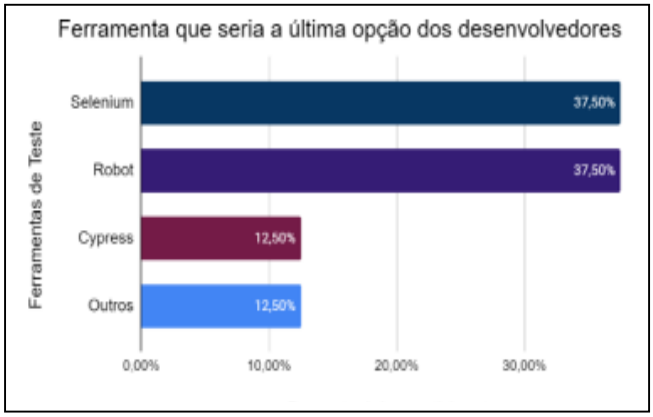


Figura 6: Gráfico da última opção de escolha.



Fonte: Autora.

Por fim, os participantes foram questionados qual seria a ferramenta escolhida como última opção, levando ao empate das ferramentas Selenium e Robot, com 37,50%. Aparentemente não há relação entre esse gráfico e os demais, visto que os percentuais não são equivalentes ao gráfico de ferramentas mais utilizadas representado na fihta 6 (acima).

Considerando esses dados, em preferência pelos desenvolvedores, o Cypress foi considerado a melhor ferramenta nos aspectos avaliados, seguido do Selenium e do Robot. Os aspectos que se mostraram mais relevantes na escolha do software de teste, conforme percentuais dos gráficos, foram a facilidade de instalação e a usabilidade.

3.1.2 Análise a partir da ISO/IEC 25010:2011 - Testes

3.1.2.1 Casos de testes com Cypress A configuração do projeto:

Para executar o projeto do Cypress com comandos minimamente verbosos, foram utilizados os seguintes atalhos de comandos: cypress:open, cypress e cypress:chrome. A cada abreviação foram adicionados scripts, sendo possível executar por meio de npm run antes de cada comando abreviado.

A cada arquivo criado com o final .spec.js, um novo teste ficará disponível para ser codificado e ficará também disponível na visualização pelo navegador.toda vez que for usado o comando npm run cypress:open esse modo de visualização ficará disponível para manipulação dos testes e assim conseguir detalhadamente como o teste está sendo executado e caso ocorra um erro ele mostra o ponto exato desta falha, por consequência essa opção traz um bom desempenho para a realização dos testes. No arquivo cypress.json foram criados parâmetros com dados de url, textos de validação e informações de login para serem utilizados dentro dos testes

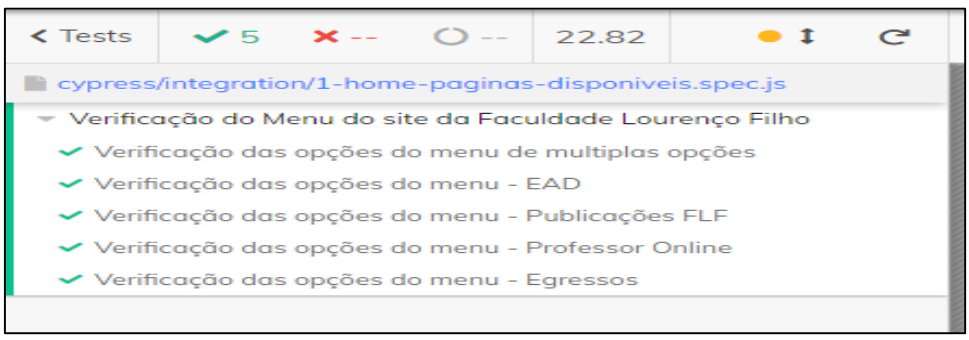
3.1.2.2 O Desenvolvimento dos testes:

Casos de testes 1 - Verificar o funcionamento de cada item do menu principal:

Foi criado o arquivo 1-home-paginas-disponiveis.spec.js para realização dos testes no menu da home e verificar se cada opção do menu funciona corretamente. Nesse arquivo cada opção do menu foi testada pelo click e comparada a sua descrição para ser verificada sua existência, e caso existisse o teste seria concluído com sucesso. No caso de páginas que abrem fora da URL (Uniform Resource Locator) do site, foi utilizado o comando invoke e should para verificar se a página da opção clicada foi aberta.

Depois de executar o comando npm run cypress:open, uma página no navegador foi aberta e ao clicar no arquivo 1-home-paginas-disponiveis.spec.js, todos os testes foram executados, indicando os testes com sucesso e os testes com erro (Figura 7).

Figura 7: Execução dos testes da verificação do menu com sucesso feito com cypress



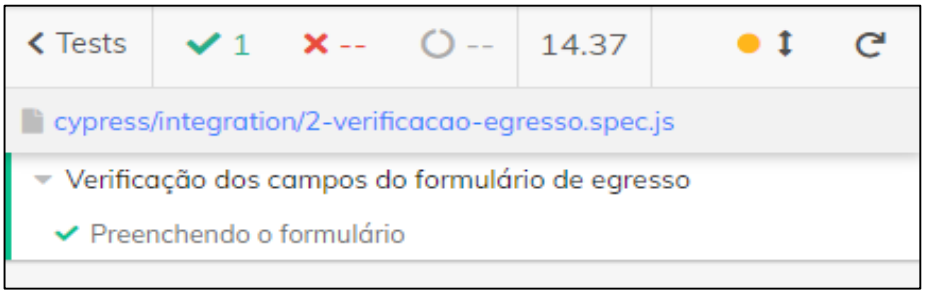
Fonte: Autora.

Casos de testes 2 - Verificação dos campos do formulário de egresso:

Foi criado o arquivo 2-verificacao-egresso.spec.js para validação do formulário de egresso. Nesse arquivo cada opção do formulário foi testada, inserindo informações nos campos input com o comando type e os campos select com o comando select, usando os seus identificadores para localizá-los e realizando o envio do formulário em seguida, no final foi validado a mensagem de retorno que aparece ao enviar as informações e caso seja uma retornar igual ao já espero o teste é considerado válido.

Depois de executar o comando npm run cypress:open uma página no navegador será aberta e ao clicar no arquivo todos os testes foram executados e será mostrado os testes com sucesso e os testes com erro (Figura 8).

Figura 8: Execução dos testes da verificação do formulário de egresso feito com cypress.



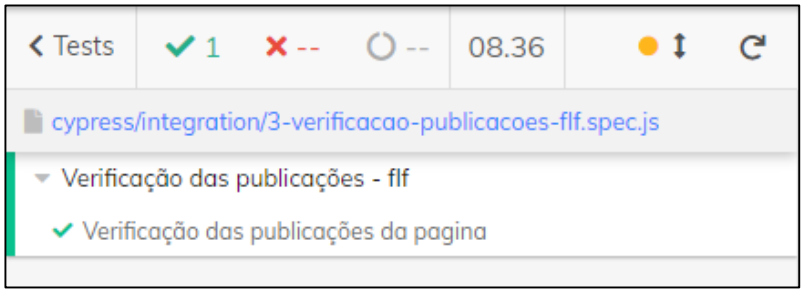
Fonte: Autora.

Casos de testes 3 - Verificação das publicações da flf:

Foi criado o arquivo 3-verificacao-publicacoes-flf.spec.js para realizar os testes de verificação de quantidade de publicações. Nesse arquivo a página de publicações foi acessada e foi verificado pela estrutura HTML a quantidade de publicações usando o comando should com a regra de ter pelo menos um item, e se essa regra for válida o teste será aceito.

Depois de executar o comando npm run cypress:open uma página no navegador será aberta e ao clicar no arquivo todos os testes foram executados e será mostrado os testes com sucesso e os testes com erro (Figura 9).

Figura 9: Execução dos testes de validação das quantidade de publicações feitas com cypress.



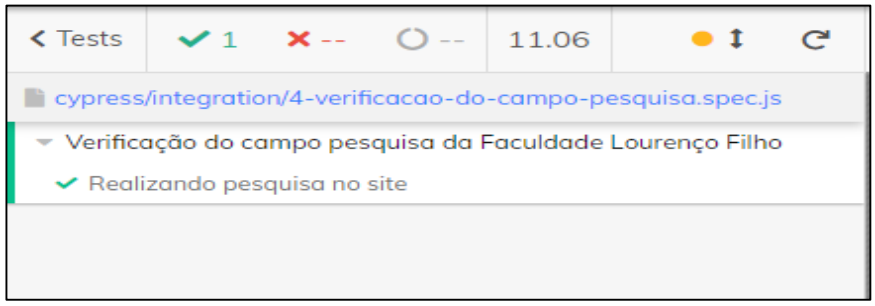
Fonte: Autora.

Casos de testes 4 - Verificação do campo pesquisar:

Para realização dos testes do campo de pesquisa do site, foi criado o arquivo "verificacao-do-campo-pesquisa.spec.js". Nesse arquivo foi pesquisado "tecnologia" e verificado se pelo menos um registro foi retornado pelo site.

Depois de executar o comando npm run cypress:open uma página no navegador foi aberta e ao clicar no arquivo todos os testes foram executados, indicando os testes com sucesso e com erros (Figura 10).

Figura 10: Execução dos testes de validação do campo pesquisar feito com cypress.



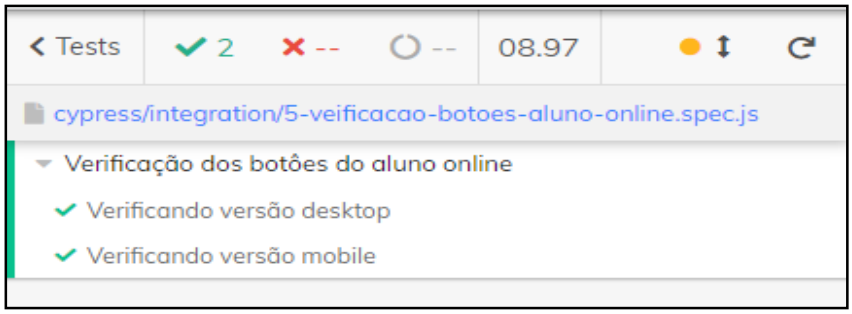
Fonte: Autora.

Casos de testes 5 - Verificação dos botões desktop e mobile:

Foi gerado o arquivo 5-veificacao-botoes-aluno-online.spec.js para realizar os testes da página Aluno online e validar as opções de login. Nesse arquivo cada opção de login foi testada clicando em desktop ou mobile, assim é possível identificar o funcionamento dos botões ou se alguma mudança foi realizada, gerando o controle de qualidade.

Depois de executar o comando npm run cypress:open uma página no navegador foi aberta e ao clicar no arquivo todos os testes foram executados, indicando aqueles com sucesso e com erro (Figura 11).

Figura 11: Execução da validação dos botões de login em aluno online feito com cypress.



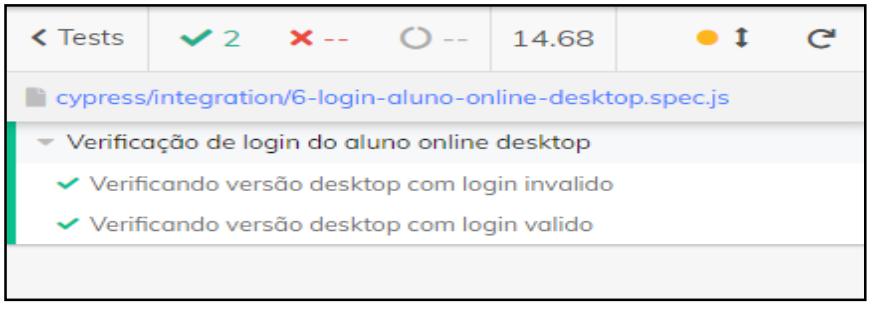
Fonte: Autora.

Casos de testes 6 - Login no aluno online desktop:

O arquivo 6-login-aluno-online-desktop.spec.js foi criado para realizar os testes da página Aluno Online e validar o formulário de login. Nesse arquivo foram testados os campos de login, com um login invalido e um login válido na opção desktop.

Depois de executar o comando npm run cypress:open uma página no navegador foi aberta e ao clicar no arquivo todos os testes foram executados, indicando aqueles com sucesso e com erro (Figura 12).

Figura 12: Execução dos testes de validação de login no modo desktop feito com cypress.

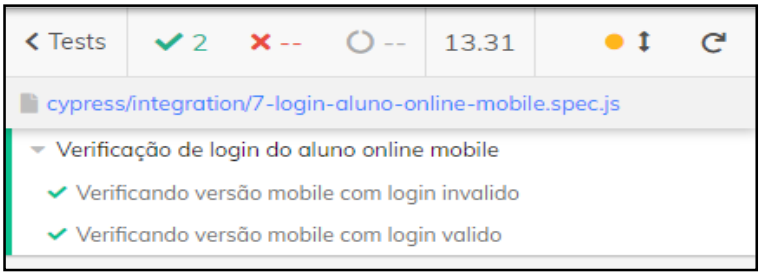


Fonte: Autora.

Casos de testes 07 - Login no aluno online mobile:

Criando o arquivo 7-login-aluno-online-mobile.spec.js para realizar os testes da página Aluno online e validando o formulário de login. Neste arquivo foram testados os campos de login, com um login invalido e um válido na opção mobile. Depois de executar o comando npm run cypress:open uma página no navegador foi aberta e ao clicar no arquivo todos os testes foram executados, indicando aqueles com sucesso e com erro (Figura 13).

Figura 13: Execução dos testes de validação de login no modo mobile feito com cypress.



Fonte: Autora.

Resultado final dos testes:

Na execução do comando npm run cypress:open ao clicar na opção Run, todos os testes foram executados e um resumo dos testes foram mostrados (Figura 14). Arquivos de prints e vídeos são criados dentro do projeto, assim cada ação dos testes são salvos, ficando mais fácil de identificar o erro caso um teste não passe.

Figura 14: Execução dos testes do projeto feito com cypress com sucesso e com erro

Fonte: Autora.

3.1.1.1 Casos de testes com Selenium

A configuração do projeto:

Cada teste criado com a extensão side foi reconhecido como teste e pôde ser executado

O Desenvolvimento dos testes:

Casos de testes 1 - Verificar o funcionamento de cada item do menu principal:

Criando o arquivo 1-home-paginas-disponiveis.side para realizar os testes no menu do site para verificar se cada opção do menu funciona. Nesse arquivo as opções Instituição, Cursos, Serviços, Financiamento, Biblioteca, Atendimento, EAD, Publicações FLF, Professor online e Egressos foram testadas pelo click para ser verificado se existe.Depois de clicar na opção de Run All Tests, todos os testes foram executados, e foram exibidos os testes com sucesso e com erro (Figura 15).

Figura 15: Execução dos testes da verificação do menu com sucesso feito com selenium.

https://fff.edu.br/

	Command	Target	Value
1	open	/	
2	set window size	1295x695	
3	click	xpath=//a[contains(text(),'Instituição')]	
4	click	xpath=//a[contains(text(),'Cursos')]	
5	click	xpath=//a[contains(text(),'Serviços')]	
6	click	xpath=//a[contains(text(),'Financiamento')]	
7	click	xpath=//a[contains(text(),'Biblioteca')]	
8	click	xpath=//a[contains(text(),'Atendimento')]	
9	click	xpath=//a[contains(text(),'Egressos')]	
10	click	xpath=//a[contains(text(),'Publicações FLF')]	
11	execute script	window.history.go(-1)	
12	click	xpath=//a[contains(text(),'Professor Online')]	
13	execute script	window.history.go(-1)	
14	click	xpath=//a[contains(text(),'EAD')]	
15	close		

Fonte: Autora.

Casos de testes 2 - Verificação dos campos do formulário de egresso:

Criando o arquivo 2-verificacao-egresso.side para realizar os testes da página Egressos e validando o formulário. Nesse arquivo cada opção do formulário foi testada inserindo uma informação sendo digitada ou selecionada e depois enviada, no final é validado a mensagem de retorno e caso seja igual ao esperado o teste será válido.Depois de clicar na opção de Run All Tests, todos os testes foram executados, e foram exibidos os testes com sucesso e com erro (Figura 16).

Figura 16: Execução dos testes da verificação do formulário de egresso feito com selenium.

https://fff.edu.br/

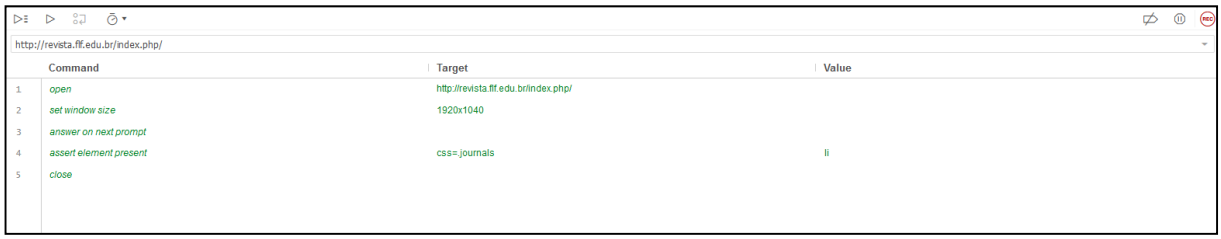
	Command	Target	Value
1	open	https://fff.edu.br/	
2	set window size	1920x1040	
3	click	xpath=//a[contains(text(),'Egressos')]	
4	type	id=nome_aluno	teste
5	type	id=matricula_aluno	3115111
6	type	id=data_nascimento_aluno	1996-10-30
7	type	id=ANO_CURSO	sistemas
8	select	id=turno_cursado	label=Noite
9	type	id=ANO_ALUNO	2021
10	type	id=work_empresa	distrito
11	type	id=justificativa_indicacao_curso	sim
12	select	id=select_regime_work	label=Emprego CLT
13	type	id=criticas_sugestoes	teste para o tcc
14	click	id=enviar	
15	assert element present	xpath=//div[contains(text(),'Agradecemos a sua mensagem.')]	
16	close		

Fonte: Autora.

Casos de testes 03 - Verificação das publicações da flf:

Criando o arquivo 3-verificacao-publicacoes-flf.side para realizar os testes de verificação de publicação na página. Nesse arquivo foi verificado a quantidade das publicações pela estrutura definida do HTML, caso o retorno das publicações sejam maior que um o teste será considerado válido. Depois de clicar na opção de Run All Tests, todos os testes foram executados, e foram exibidos os testes com sucesso e com erro (Figura 17).

Figura 17: Execução dos testes de validação das quantidades de publicações feitas com Selenium.

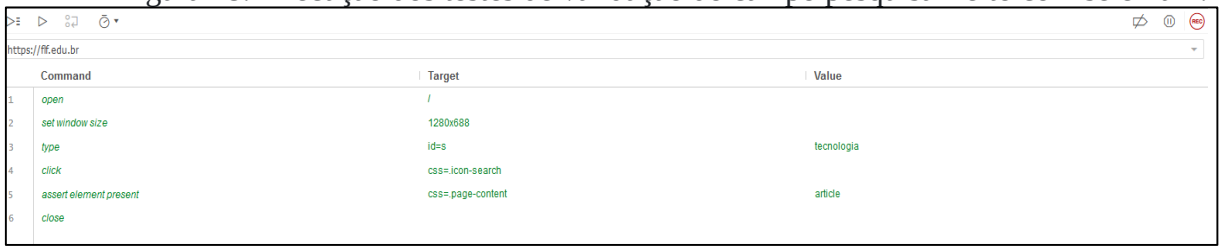


Fonte: Autora.

Casos de testes 04 - Verificação do campo pesquisar:

Criando o arquivo 4-verificacao-do-campo-pesquisa.side para realizar os testes no campo de pesquisa. Nesse arquivo foi inserido a palavra “tecnologia” e foi verificado se obteve resultados pela pela estrutura HTML, verificando se tem pelo menos um resultado. Depois de clicar na opção de Run All Tests, todos os testes foram executados, e foram exibidos os testes com sucesso e com erro (Figura 18)

Figura 18: Execução dos testes de validação do campo pesquisar feito com selenium.

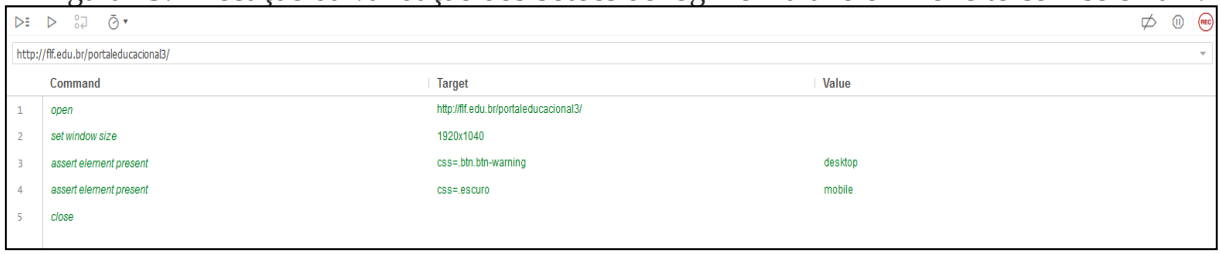


Fonte: Autora.

Casos de testes 05 - Verificação dos botões desktop e mobile:

Criando o arquivo 5-veificacao-botoes-aluno-online.side para verificar as opções disponíveis de login da página aluno online. Nesse arquivo cada opção de login foi verificada (desktop e mobile) pelo seu identificador, caso sejam encontrados, um clique será feito e o teste será válido. Depois de clicar na opção de Run All Tests, todos os testes foram executados e foram exibidos os testes com sucesso e com erro (Figura 19).

Figura 19: Execução da validação dos botões de login em aluno online feito com selenium.

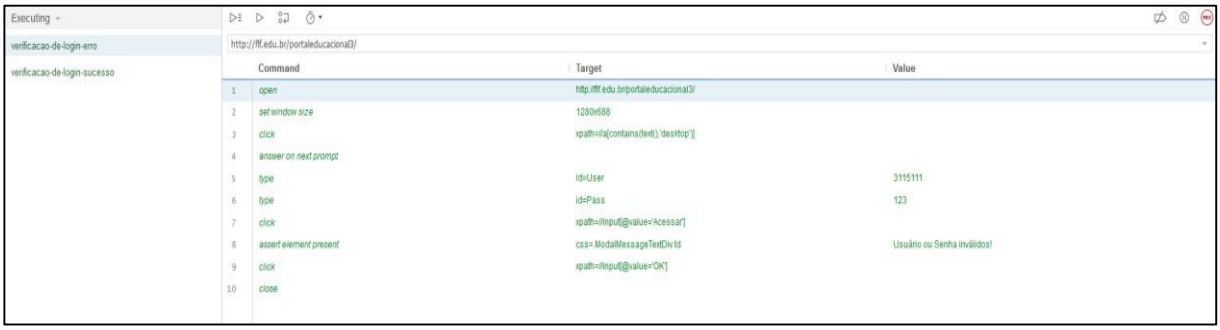


Fonte: Autora.

Casos de testes 06 - Login no aluno online desktop:

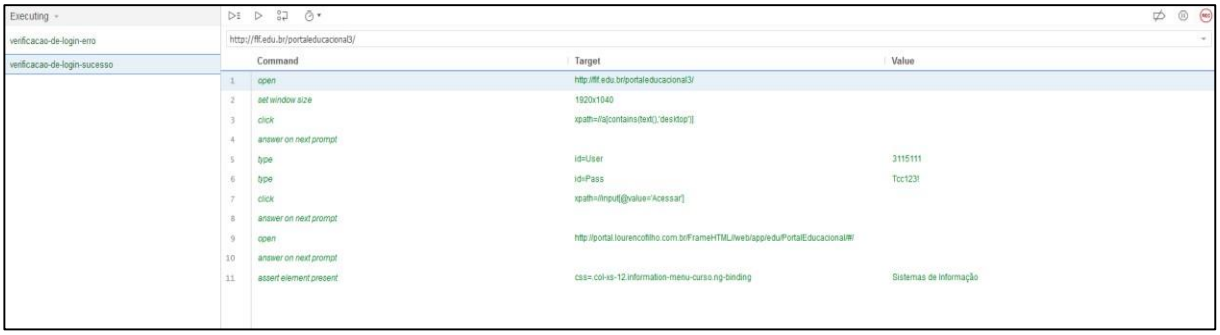
Criando o arquivo 6-login-aluno-online-desktop.side para realizar a verificação do login no modo desktop. Nesse arquivo foi inserido o usuário, a senha e realizando um click no botão de entrar, cada campo inserido é clicado pelo seu identificador, depois do login é verificado se está logado na página desktop, a verificação também é feita para um login invalido e é verificado se retorna uma mensagem de erro e caso aconteça alguma alteração nesse modo será possível identificar e isso gera um controle de qualidade. Depois de clicar na opção de Run All Tests, todos os testes foram executados, e foram exibidos os testes com sucesso e com erro (Figura 20) e (Figura 21).

Figura 20: Execução dos testes de validação de login no modo desktop feito com selenium.



Fonte: Autora.

Figura 21: Execução dos testes de validação de login no modo desktop feito com selenium.

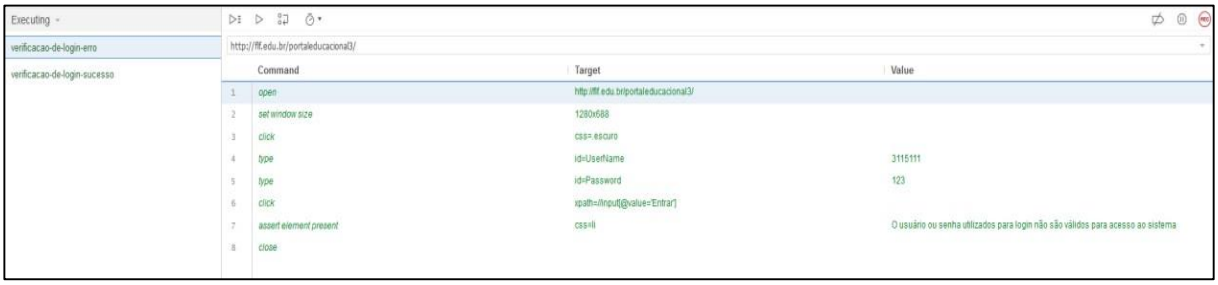


Fonte: Autora.

Casos de testes 07 - Login no aluno online mobile:

Criando o arquivo 7-login-aluno-online-mobile.side para realizar a verificação do login no modo mobile. Nesse arquivo foi inserido o usuário, a senha e realizando um click no botão entrar, cada campo inserido é clicado pelo seu identificador, depois do login é verificado se está logado na página mobile, a verificação também é feita para um login invalido e é verificado se retorna uma mensagem de erro, caso a mensagem seja o esperado o teste é considerado válido. Depois de clicar na opção de Run All Tests, todos os testes foram executados, e foram exibidos os testes com sucesso e com erro (Figura 22) e (Figura 23).

Figura 22: Execução dos testes de validação de login no modo mobile feito com selenium.



Fonte: Autora.

Figura 23: Execução dos testes de validação de login no modo mobile feito com selenium.

Fonte: Autora.

Resultado final dos testes:

Ao clicar no botão Run All Tests todos os testes foram executados e um resumo dos testes foram mostrados (Figura 24).

Figura 24: Execução de todos os testes do projeto com sucesso.

Fonte: Autora.

Quando existe algum erro nos testes, esse é marcado em vermelho, facilitando a identificação do teste que não passou (Figura 25).

Figura 25: Execução de todos os testes do projeto com erro.

Fonte: Autora.

3.1.1.3 Casos de testes com Robot

A configuração do projeto:

A cada arquivo criado com o final .robot, um novo teste ficará disponível para ser codificado (Figura 58), toda vez que for usado o comando "robot -d ./resultados testes" será executado todos os testes, e quando o comando "robot -d ./resultados

./testes/nome_do_arquivo.robot" for inserido apenas o arquivo selecionado será executado, assim é possível conseguir detalhadamente como o teste está sendo aplicado, mostrando no final os sucessos e erros.

Foi criado um arquivo chamado resource.robot, com o propósito de organizar as variáveis de identificadores dos campos, URL do site e valores dos testes para ter uma maior organização dentro do projeto, resultando em uma leitura mais clara e limpa ao decorrer do desenvolvimento.

O Desenvolvimento dos testes:

Casos de testes 01 - Verificar o funcionamento de cada item do menu principal:

Criando o arquivo 1-home-paginas-disponiveis.robot para realizar a validação da disponibilidade do menu do site Faculdade Lourenço Filho. Nesse arquivo foi testado o clique nas opções do menu, Instituição, Cursos, Serviços, Financiamento, Biblioteca, Atendimento, EAD, Publicações FLF,Professor online e Egressos, caso todas essas opções sejam encontradas o teste é considerado válido. Depois de executar o comando robot -d ./resultados ./testes/1-home-paginas- disponiveis.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro.

Casos de testes 02 - Verificação dos campos do formulário de egresso:

Criando o arquivo 2-verificacao-egresso.robot para realizar a validação do formulário de egresso. Nesse arquivo foi inserido informações nos inputs e select, localizando cada campo por seus identificadores, ao terminar de inserir as informações o botão de enviar será acionado, caso o retorno da mensagem seja a esperada o teste é aceito como válido. Depois de executar o comando robot -d ./resultados ./testes/2-verificacao- egresso.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro.

Casos de testes 03 - Verificação das publicações da flf:

Criando o arquivo 3-verificacao-publicacoes-flf.robot para realizar a validação de quantidade de publicações. Nesse arquivo foi testado quantas publicações existem, a quantidade de publicações é pega pela estrutura do HTML da página e caso tenha mais de uma publicação o teste será válido. Depois de executar o comando robot -d ./resultados ./testes/3-verificacao-publicacoes- flf.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro.

Casos de testes 04 - Verificação do campo pesquisar:

Criando o arquivo 4-verificacao-do-campo-pesquisa.robot para realizar a validação do campo de pesquisa. Nesse arquivo foram testados os resultados do campo de pesquisa com a palavra "tecnologia" e ao pesquisar se no HTML retornar pelo menos um item como resultado, o teste será válido. Depois de executar o comando robot -d ./resultados ./testes/4-verificacao-do-campo- pesquisa.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro.

Casos de testes 05 - Verificação dos botões desktop e mobile:

Criando o arquivo 5-veificacao-botoes-aluno-online.robot para realizar a validação da disponibilização dos botões de online do aluno online. Nesse arquivo foi testado a existência dos

botões de login desktop e mobile por meio de seus identificadores, no final será realizado um clique e caso seja encontrado os botões o teste será válido. Depois de executar o comando robot -d ./resultados ./testes/5-veificacao-botoes-aluno-online.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro.

Casos de testes 06 - Login no aluno online desktop:

Criando o arquivo 6-login-aluno-online-desktop.robot para realizar a validação de login invalido e válido na opção desktop. Nesse arquivo foi testado os campos de login do modo desktop, o primeiro cenário foi o comportamento de login invalido e verificado a mensagem de retorno, por último foi validado o login válido e caso seja realizado o login e validado a mensagem o teste será válido. Depois de executar o comando robot-d./resultados./testes/6-login-aluno-online- desktop.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro.

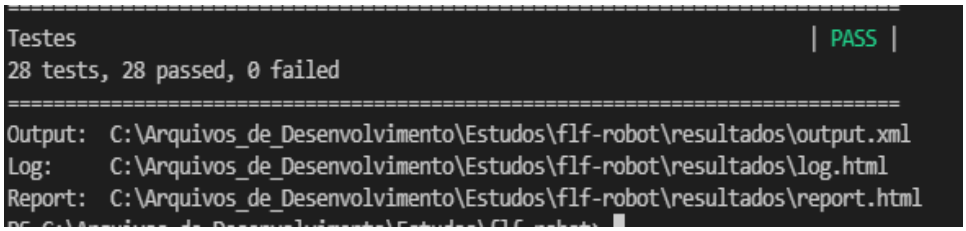
Casos de testes 07 - Login no aluno online mobile:

Criando o arquivo 7-login-aluno-online-mobile.robot para realizar a validação de login invalido e válido na opção mobile. Nesse arquivo foi testado os campos de login do modo mobile, o primeiro cenário foi o comportamento de login invalido e verificado a mensagem de retorno, por último foi validado o login válido e caso seja realizado o login e validado a mensagem o teste será válido.Depois de executar o comando robot -d ./resultados ./testes/7-login-aluno-online-mobile.robot, todos os casos de testes foram executados, e ao finalizar os resultados foram exibidos com sucesso ou com erro

Resultado final dos testes:

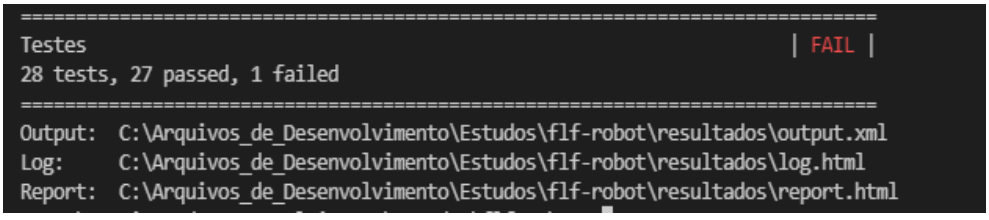
Ao executar o comando robot -d ./resultados testes todos os testes foram executados e um resumo dos testes foram mostrados (Figura 26), caso exista erros nos testes, o erro ficará em vermelho e será possível identificar qual teste não passou (Figura 27).

Figura 26: Execução de todos os testes do projeto feito com robot com sucesso.



Fonte: Autora.

Figura 27: Execução de todos os testes do projeto feito com robot com erro.



Fonte: Autora.

4 DISCUSSÃO

Tratando-se da experiência durante os testes com cada uma das ferramentas, identificou-se aspectos muito relevantes para colaborar na escolha da melhor ferramenta para situações similares a este trabalho.

Cypress se mostrou uma solução com baixa curva de aprendizado, com muita documentação e comunidade bastante ativa, o que viabiliza e acelera sua instalação e execução. Nele os testes são codificados e para pode fazer ou entender o desenvolvimento com cypress é necessário ter pelo menos um conhecimento de Javascript e também um pouco de lógica de programação. Os comandos não são complicados de entender, mas a maior parte do processo é manual. Depois de executar os testes e verificar o navegador, fica disponível uma opção para pegar os elementos em HTML, e assim facilita um pouco nesse processo de desenvolvimento de testes. Quando executado o navegador abre com todos os arquivos de testes e mostra várias opções para executar arquivo por arquivo ou executar todo os testes juntos, também existe a opção de ver cada passo do teste e pegar os identificadores dos elementos HTML.

Para o Selenium, utilizando a ferramenta do navegador, se mostrou bastante prática, a codificação fica padronizada e pode ser feito todos os testes só pelo painel, mas para processos maiores, ele passa a perder praticidade, pois é necessário estudar a própria lógica de suas funções e isso leva um tempo maior, existe documentação disponível, mas muitas vezes na prática fica difícil passar a lógica para a sua codificação.

O Robot Framework possui uma grande curva de aprendizado, pois é orientado a Python e com opções de desenvolvimento do Selenium e no início do processo de desenvolvimento pode parecer bem cansativo, mas depois de pesquisar e entender a ferramenta se torna incrivelmente prática, para grandes sistemas é bem útil, para pequenos sistemas talvez não seja o mais prático.

5 CONCLUSÃO

Nesta pesquisa foi realizado um estudo comparativo entre os as ferramentas de automação cypress, selenium e robot framework, a fim de buscar a melhor ferramenta em relação ao aprendizado e tempo de desenvolvimento dos testes feito no site da faculdade Lourenço Filho.

A automação de testes está ligada diretamente à qualidade final do produto e faz sentido ser feito quando há garantias que não haverão mudanças significativas no produto final e quando é necessário ter resultados rápidos com o menor custo. Para os desenvolvedores que participaram do formulário sobre as ferramentas de teste, o Cypress é a melhor opção de ferramenta de teste.

Foi identificado neste trabalho que, para testes rápidos em páginas web, a ferramenta de automação Cypress se mostrou mais eficiente, rápida e intuitiva, reduzindo custos e tempo no processo de teste de qualidade de software. Apesar de em algumas etapas ser manual, sua curva de aprendizado foi a menor em comparação aos demais softwares.

Para projetos simples e com pequenas funções a serem testadas, o Selenium Web Driver se mostrou eficaz, mas quando utilizado em testes mais complexos passa a se tornar manual e exige grande esforço para entender a lógica.

O Robot Framework, apesar de uma grande curva de aprendizado, para testes complexos e para grandes projetos, que exigem a melhor qualidade possível, ele é bastante recomendável. Se o desenvolvedor adquirir bastante prática nessa ferramenta, poderá realizar todo tipo de testes em quaisquer projetos.

REFERÊNCIAS

ALMEIDA, Aloísio. Pirâmide de Testes: porque você precisa saber. **Blog Devporaí.** [S.l.], 9 de out. 2020. Disponível em: https://devporai.com.br/piramide-de-testes/. Acesso em: 10 set. 2021.

ALMEIDA**,** Gustavo. **Evolução e desafios em Teste de Software**. 2020. Disponível em: https://vtex.com/pt-br/blog/produto/evolucao-e-desafios-em-teste-de-software/ Acesso em: 30 set. 2021

BOAS, Joana; LOPES, Nuno; SILVA, João. Software Testing Automation for the ISO/IEC 25051 Standard. Atas da 16ª Conferência da Associação Portuguesa de Sistemas de Informação, [S.L.], v. 16, n. 1, p. 212-230, 24 set. 2016. **Associação Portuguesa de Sistemas de Informação,** APSI. http://dx.doi.org/10.18803/capsi.v16.231-236. Disponível em: http://revista.apsi.pt/index.php/capsi/article/view/488/442. Acesso em: 26 jun. 2021.

CAMPOS, Camila. **A pirâmide de testes**. 2019. Disponível em: https://medium.com/creditas-tech/a-pir%C3%A2mide-de-testes-a0faec465cc2. Acesso em: 27 jun. 2021.

CEVERINO, Aparecida. NASCIMENTO, Fernando Paes. Utilização da técnica de desenvolvimento orientado por comportamento (BDD) no levantamento de requisitos. **Revista Interdisciplinar Científica Aplicada**, Blumenau, v.10, n.3, p.40-51, TRIII 2016. ISSN 1980-7031. Disponível em: https://rica.unibes.com.br/rica/article/view/728/753. Acesso em: 30 set. 2021.

CHICANELLI, Rachel; VECCHIATO, Daniel; VENTURA, Thiago; GOMES, Raphael; TAKAGI, Nilton & MENDES, Luana. **Aspectos sociais, humanos e econômicos da utilização de testes automatizados no desenvolvimento de sistemas.** Conferencia Iberoamericana en Sistemas, Cibernética e Informática, 18, Flórida, CISCI, 2019. p. 6-10. Disponivel em: https://www.researchgate.net/profile/Nilton-Takagi/publication/337621126_Aspectos_sociais_humanos_e_economicos_da_utilizacao_de

____testes_automatizados_no_desenvolvimento_de_sistemas/links/5de07f6a4585159aa451973e/ Aspectos-sociais-humanos-e-economicos-da-utilizacao-de-testes-automatizados-no-desenvolvimento-de-sistemas.pdf. Acesso em: 21 jun. 2021.

CYPRESS. **Why Cypress?** Disponível em: https://docs.cypress.io/guides/overview/why-cypress. Acesso em: 30 out. 2021.

ISO/IEC 25010:2011 - System and software quality models. Disponível em: https://www.iso.org/standard/35733.html . Acesso em: 21 jun. 2021.

JOSHI, Vipin. **Cypress vs Selenium - Selecione sua estrutura de automação.** Índia: Cynoteck, 2020. Disponível em: https://cynoteck.com/pt/blog-post/cypress-vs-selenium- select-your-automation-framework/. Acesso em: 20 jun. 2021.

MALM, Anu. **UI-testiautomaation aloitus Robot Frameworkia hyväksi käyttäen**. Tampere: Tampere University Of Applied Sciences, 2020. 60 f.. Disponível em: https://www.theseus.fi/bitstream/handle/10024/349265/Malm_Anu.pdf?sequence=2. Acesso em: 28 set. 2021.

MARQUES, Nicholas N.; FERNANDES, Rafael A. **Um arcabouço para a geração automatizada de testes funcionais a partir de cenários BDD**. 2020. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)—Universidade de Brasília, Brasília, 2020.

MOBARAYA, Fatini; ALI, Shahid. Technical Analysis of Selenium and Cypress as Functional

Automation Framework for Modern Web Application Testing. 9Th International Conference On Computer Science, Engineering And Applications (Iccsea 2019), v. 9, n. 18, p. 27-46, 21 dez. 2019. **Aircc Publishing Corporation**. http://dx.doi.org/10.5121/csit.2019.91803. Disponível em: https://aircconline.com/csit/abstract/v9n18/csit91803.HTML. Acesso em: 11 set. 2021.

MONITORA. **Quais os tipos de testes de software e por que automatizá-los?** 11 de fev. 2019. Blog Monitora. Disponível em: https://www.monitoratec.com.br/blog/quais-os-tipos- de-testes-de-software-e-por-que-automatiza-los/. Acesso em: 11 set. 2021.

OKEZIE, F.; ODUN-AYO, I.; BOGLE, S.. A Critical Analysis of Software Testing Tools. **Journal Of Physics:** Conference Series, [S.L.], v. 1378, n. 042030, p. 1-12, dez. 2019. IOP Publishing. http://dx.doi.org/10.1088/1742-6596/1378/4/042030. Disponível em: https://iopscience.iop.org/article/10.1088/1742-6596/1378/4/042030/pdf. Acesso em: 10 set. 2021.

PAKARINEN, Toni. Automaattisen Testauksen Ohjelmistoja. In: PAKARINEN, Toni. **Ohjelmien automaattinen testaus**. Tampere: Tampere University Of Applied Sciences, 2020. Cap. 5. p. 21-44. Disponível em: https://www.theseus.fi/bitstream/handle/10024/347029/Pakarinen_Toni.pdf?sequence=3. Acesso em: 11 set. 2021.

PELIVANI, Elis; CICO, Betim. A comparative study of automation testing tools for web applications. In: MEDITERRANEAN CONFERENCE ON EMBEDDED COMPUTING

(MECO), 10., 2021, Budva. **A comparative study of automation testing tools for web applications.** [S.L.]: IEEE, 2021. p. 1-6. Disponível em: https://ieeexplore.ieee.org/abstract/document/9460242/references#references. Acesso em: 11 set. 2021.

PSUJEK, M.; RADZIK, A.; KOZIEŁ, G. Comparative analysis of solutions used in Automated Testing of Internet Applications. **Journal of Computer Sciences Institute,** v. 18,

p. 7-14, 30 Mar. 2021. Disponível em: https://ph.pollub.pl/index.php/jcsi/article/view/2373. Acesso em: 21 jun. 2021.

ROBOT. **Robot Framework Introduction**. Disponível em: https://robotframework.org/#introduction cesso em: 30 set. 2021.

ROLLWAGEN, André Fernando et al. COMPARATIVO ENTRE FERRAMENTAS DE AUTOMAÇÃO DE TESTES DE SOFTWARE PARA SISTEMAS WEB. In: JORNADA DE PESQUISA, 24., 2020, Ijuí. **Anais [...]** . Ijuí: Unijuí, 2020. p. 1-11. Disponível em: https://publicacoeseventos.unijui.edu.br/index.php/salaoconhecimento/article/view/18489/17223. Acesso em: 11 set. 2021.

ROMANINI, Isabela Ziviani; SOTTO, Eder Carlos Salazar. SELENIUM WEB DRIVER NA EVOLUÇÃO DOS TESTES MANUAIS. **Revista Interface Tecnológica**, [S.L.], v. 16, n. 2, p. 112-123, 26 dez. 2019. Interface Tecnológica. http://dx.doi.org/10.31510/infa.v16i2.627. Disponível em: https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/627 . Acesso em: 27 jun. 2021.

SILVA, Wagner Francalino. **Relato de experiência em automação de testes funcionais em uma empresa de TI**. 2016. 90 f.. Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Engenharia de Software, Quixadá, 2016. Disponível em: https://www.repositoriobib.ufc.br/000026/0000264f.pdf. Acesso em: 25 set. 2021.

TUTORIALS POINT. **Robot Framework Tutorial**. Disponível em: https://www.tutorialspoint.com/robot_framework/index.htm. Acesso em: 30 set. 2021.