UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Faculdade de Engenharia Elétrica

Graduação em Engenharia de Controle e Automação

GUSTAVO SALES DE OLIVEIRA

ANÁLISE DE FERRAMENTAS USADAS NO PROCESSO DE AUTOMAÇÃO DE TESTES A FIM DE EXPLICITAR CENÁRIOS PARA MELHOR UTILIZAÇÃO

UBERLÂNDIA

2023

GUSTAVO SALES DE OLIVEIRA

ANÁLISE DE FERRAMENTAS USADAS NO PROCESSO DE AUTOMAÇÃO DE TESTES A FIM DE EXPLICITAR CENÁRIOS PARA MELHOR UTILIZAÇÃO

Trabalho de Conclusão de Curso de Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Câmpus Santa Mônica, como requisito paraa obtenção do título de Graduação em Engenharia de Controle e Automação.

Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica

Orientador: Prof. Dr. Josué Silva De Morais

UBERLÂNDIA

2023

Sales, Gustavo

Análise de ferramentas usadas no processo de automação de testes a fim de explicitar cenários para melhor utilização / **Gustavo Sales de Oliveira**. – **UBERLÂNDIA**,**2023**.

Orientador: Prof. Dr. Josué Silva De Morais

Trabalho de Conclusão de Curso – Universidade Federal de Uberlândia – UFU Faculdade de Engenharia Elétrica. **2023**.

Inclui bibliografia.

1. Análise 2. Ferramentas 3. Automação 4. Testes I. Prof. Dr. Josué Silva De Morais.II. Universidade Federal de Uberlândia. III. Faculdade de Engenharia Elétrica. IV.Engenharia de Controle e Automação.

DEDICATÓRIA

Dedico este trabalho à minha mãe, Mirene Rodrigues Sales, ao meu irmão, Devanir Sabino de Oliveira Júnior.

AGRADECIMENTOS

Agradeço a minha mãe Mirene Rodrigues Sales, e ao meu irmão, Devanir Sabino de Oliveira Júnior, que sempre estiveram ao meu lado, me apoiando, dando força, estrutura e amor independente do momento, dos mais difíceis até os melhores.

Agradeço a todos que fizeram parte dessa etapa na minha vida, tais como Fausto Humberto, Alexsander Camilo, Igor Alvim, Pedro Henrique Tonim, Luiz Felipe Bueno, Elcio Rezende, Felipe Teles, Lincoln Társio, que permaneceram ao meu lado desde jornadas de estudos até em momentos de diversão que ficariam gravados na minha memória para sempre.

Agradeço aos professores e todos os colaboradores da Universidade Federal de Uberlândia (UFU). Em especial aos da Faculdade de Engenharia Elétrica e ainda mais aos que pertencem ao curso de Engenharia de Controle e Automação, por toda a informação compartilhada e pelos serviços oferecidos à educação com o objetivo de não apenas formar profissionais, mas também formar pessoas.

A doutoranda Résia Morais agradeço por todo o conhecimento e momentos que compartilhamos e vivenciamos durante o período da minha graduação. Ao professor Josué Silva de Morais um agradecimento em singular por toda sua disponibilidade, paciência e por ser meu orientador dando todo suporte necessário.

“O maior inimigo do
conhecimento não é a ignorância, é
a ilusão do conhecimento.”;
(Stephen Hawking)

RESUMO

Os softwares e aplicações computacionais têm se tornado cada vez mais presentes no cotidiano, sendo aplicados em diversas áreas, como saúde e indústria. Contudo, ao final do desenvolvimento desses produtos, é possível que surjam falhas que impactam a usabilidade do cliente. Para antecipar a identificação desses defeitos, são realizados testes, os quais destacam pontos críticos. Essas verificações podem ser conduzidas manualmente ou de forma automatizada, utilizando códigos e ferramentas para efetuar as ações e validar resultados. No contexto da automação de testes, a escolha adequada das ferramentas é fundamental para tornar o processo eficiente e menos complexo. Este trabalho visa analisar elementos relevantes para indicar contextos e a ferramenta mais apropriada. Ao estudar o Java com framework Selenium, Python com framework Robot e JavaScript com Cypress, foi possível identificar situações ideais para a utilização de cada ferramenta. Notavelmente, considerando o conhecimento técnico, o Robot pode ser indicado para níveis iniciais, o Cypress para o ponto intermediário, e o Selenium a partir do nível anterior/avançado.

Palavras-chave: Automação, Testes, Selenium, Robot, Cypress

ABSTRACT

Software and computational applications are becoming increasingly prevalent in daily life, being applied in various fields such as healthcare and industry. However, at the end of the development of these products, it is possible that flaws may arise and hinder customer usability. To proactively identify these defects, tests are conducted to highlight critical points. These checks can be performed manually or in an automated manner, using code and tools to carry out actions and validate results. In the context of test automation, the appropriate choice of tools is crucial to make the process efficient and less complex. This work aims to analyze relevant elements to indicate contexts and the most suitable tool. By studying Java with the Selenium framework, Python with the Robot framework, and JavaScript with Cypress, it was possible to identify ideal situations for the use of each tool. Particularly, considering technical knowledge, Robot may be suitable for initial levels, Cypress for the intermediate point, and Selenium from the previous level/advanced.

Keywords: Automation, Testing, Selenium, Robot, Cypress.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IDE	*Ambiente de desenvolvimento integrado*
JDK	*Java Development Kit*
PO	*Padrão de Projeto Objeto de Página ou Page Object*
POO	*Programação orientada a objetos*
TIC	*Tecnologia da Informação e Comunicação*

SUMÁRIO

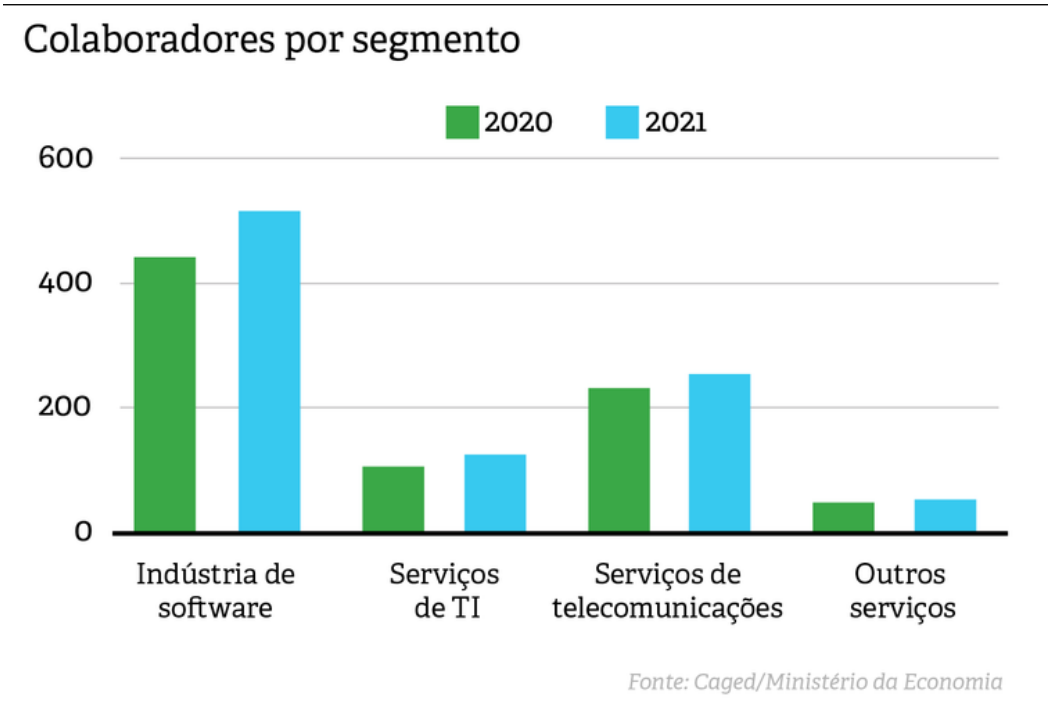
1. INTRODUÇÃO

O termo software faz-se cada vez mais presente no cotidiano das pessoas. Tal expressão representa um conjunto de dados, instruções e programas que operam em sistemas computacionais a fim de executar tarefas especificas. Assim, aplicativos e produtos estabelecidos em variadas plataformas como celulares e computadores se encaixam dentro desse conceito.

Um dos pontos que possibilitou sua popularização é o fato da grande variedade de áreas em que ele pode ser aplicado, desde o campo da saúde, produção industrial e segurança até bancário. Sua implementação possibilita padronização de processos e sua maior rastreabilidade para que custos desnecessários sejam eliminados.

Um aspecto relevante é o impacto da Industria de Software dentro da economia do país. Esta repercussão por ser vista ao analisar o número de profissionais na esfera de tecnologia. Segundo Agência Brasil (2022), a Indústria de Software e Serviços teve um aumento de 6,5% em 2021 e emprega 55% dos trabalhadores no setor de Tecnologia da Informação e Comunicação (TIC).

Figura 1 - Dispersão de trabalhadores no setor TIC



Fonte: Agência Brasil (https://agenciabrasil.ebc.com.br/economia/noticia/2022-07/industria-de-software-e-servicos-de-tic-cresceram-65-em-2021, acessado em julho de 2023)

Para que as aplicações tenham seus objetivos alcançados, é necessário que durante seu uso não haja defeitos que impossibilitem ou prejudiquem sua utilização pelo cliente. A fim de detectá-los as atividades de testes realizam essas verificações, e podem ser incluídas desde o início do ciclo de vida do software.

Essas validações fazem com que falhas sejam precocemente identificadas e corrigidas, evitando que elas gerem gastos extras nas etapas finais do produto. Essas averiguações podem ser realizadas de forma manual ou automatizada. Em situações em que os testes devem ser automatizados, a escolha da ferramenta a ser utilizada tem suma importância, pois pode tornar o processo mais eficiente.

A automação de testes para web é uma prática essencial que utiliza ferramentas e scripts automatizados para validar o correto funcionamento de aplicações web. Em vez de depender exclusivamente de testes manuais, a automação permite uma execução rápida e repetitiva de testes, garantindo que as funcionalidades estejam em conformidade com os requisitos. Essa abordagem não apenas aprimora a eficiência, proporcionando uma cobertura mais abrangente em um curto período, mas também contribui para a detecção precoce de defeitos, especialmente quando integrada a pipelines de integração contínua.

Por fim, o trabalho proposto tem como objetivo analisar ferramentas utilizadas para automação de testes em aplicações web para que a aplicação delas possa ser faz mais eficiente e adequada tanto para benefício do projeto tanto para o profissional que faz uso desse recurso.

2. REFERENCIAL TEÓRICO

Neste capítulo, são mostrados algumas teorias e conceitos que moldaram a análise, proporcionando uma visão abrangente do conhecimento existente sobre o tema.

2.1. Testes

Para que uma aplicação esteja funcionando corretamente faz-se necessário que durante o seu desenvolvimento sejam feitos diversos testes em diferentes níveis durante o processo de criação do software. Essas validações têm como função garantir a qualidade do produto objetivando evitar defeitos e verificar se os requisitos especificados foram atendidos.

Os testes contribuem para que sejam reduzidos o rico de problemas durante a execução do software em produção, assim quanto mais cedo forem incorporados no ciclo de vida de desenvolvimento maior será o impacto na qualidade da aplicação pois as falhas passam a ser detectadas em estágios iniciais do processo. Dessa forma os defeitos que poderiam gerar grandes dificuldades para o produto seriam reduzidos.

Quando se fala em testes tem-se em mente alguns princípios, dentre eles:

1) No ciclo de vida do desenvolvimento é importante incluir os testes, pois eles evitam falhas e reduzem futuras alterações que podem ser onerosas em questão de tempo e dinheiro.
2) Testes exaustivos não são viáveis, logo é importante compreender o atual contexto, ou seja, prioridades e riscos para que esses pontos respectivos sejam testados.
3) Os testes e a massa de dados utilizados para as validações devem ser atualizados se foram utilizados muitas vezes pois dessa forma eles não encontraram novas falhas.

Quando se trata da revisão de testes já existentes é importante salientar o ponto conhecido como “Paradoxo do Pesticida” (BEIZR, Boris. *Software Testing Techniques*). O paradoxo em testes é uma metáfora que destaca a armadilha potencial de utilizar sempre os mesmos conjuntos de testes para avaliar um software. Assim como os insetos podem se tornar resistentes a um pesticida específico com o

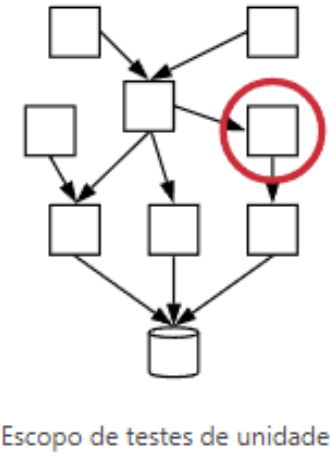
tempo, os testes contínuos e repetitivos sem variação podem deixar de identificar novos tipos de defeitos. Ele ressalta a importância de revisar e ajustar regularmente os casos de teste, incorporando novos cenários e estratégias, para garantir a eficácia contínua na detecção de defeitos.

Tendo em vista que existem diversos elementos e camadas que podem ser validadas foram criados os níveis de testes os quais contemplam atividades de testes que são relacionadas a etapas dentro do ciclo de vida de desenvolvimento do software. Os níveis de teste são:

1) Testes de componentes ou unitários;
2) Testes de Integração;
3) Testes de Sistemas;
4) Testes de Aceite;

Nos Testes de componentes ou unitários o objetivo é testar elementos separadamente, ou seja, validar se o componente funciona conforme os critérios desejados. Nesse nível falhas como erros de cálculo ou estrutura dos componentes são identificadas. Normalmente essas validações são executadas pelo desenvolvedor no processo de criação do item pois são diretamente ligadas ao código.

Figura 2 - Alvo do teste unitario

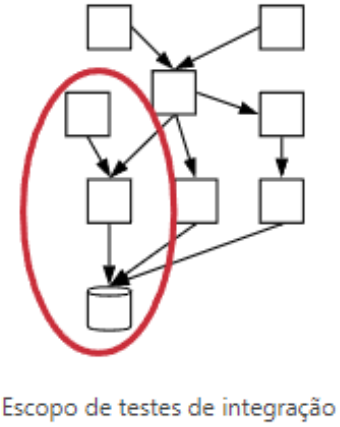


Fonte: Engenharia de Software Moderna (https://engsoftmoderna.info/cap8.html, acessado em julho de 2023)

Para os Testes de integração o foco é testar as iterações feitas entre componentes ou sistemas. Estas validações analisam o software e seus subsistemas

ou bancos de dados a fim de encontrar problemas como dados incorretos, falhas de comunicação e chamadas entre eles.

Figura 3 - Objetivo do teste de integração

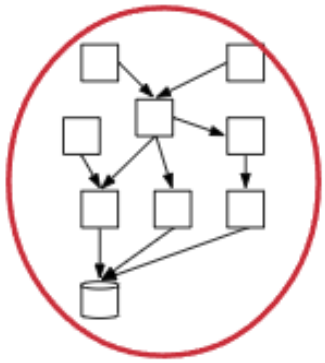


Fonte: Engenharia de Software Moderna (https://engsoftmoderna.info/cap8.html, acessado em julho de 2023)

Os testes de Sistemas concentram-se em validar o produto como um todo para que todas as tarefas desenvolvidas estejam de acordo com os critérios planejados. Assim, não só os aspectos ligados a comportamento são analisados, mas também testes relacionados a desempenho, segurança e usabilidades são realizados.

Para os testes de Aceite os objetivos são semelhantes ao nível de testes de Sistema, porém eles possuem uma visão direcionada para uso do usuário final. Dessa forma, pontos como ambiente real de aplicação do software, sua performance e usabilidade do sistema para o cliente são fundamentais para identificar problemas e melhorias.

Figura 4 - Intuito do teste de sistema

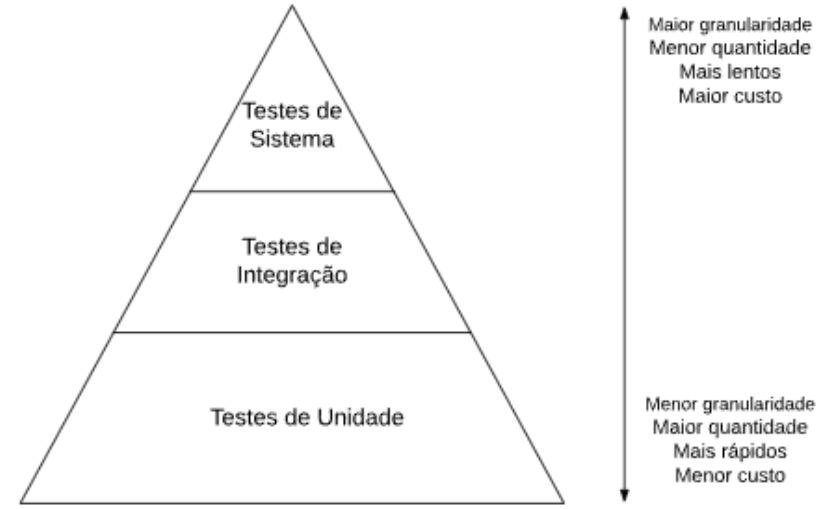


Escopo de testes de sistema

Fonte: Engenharia de Software Moderna (https://engsoftmoderna.info/cap8.html, acessado em julho de 2023)

Para tornar esses níveis mais fáceis de entender e aplicar, foi criada a Pirâmide de testes por Mike Cohn em seu livro *Succeeding with Agile*. A partir dela torna-se intuitivo determinar qual seria o esforço e quantidade de testes em cada nível dentro do ciclo de vida de desenvolvimento da aplicação.

Figura 5 - Pirâmide de Testes



Pirâmide de testes

Fonte: Engenharia de Software Moderna (https://engsoftmoderna.info/cap8.html, acessado em julho de 2023)

2.2. Tipos de testes

É importante entender os tipos de testes os quais são grupos de atividades que objetivam validar um aspecto particular do sistema, além de serem realizados dentro dos níveis de testes. Dentre os tipos podem ser destacados os Testes funcionais, Testes não funcionais, Teste caixa-branca e Teste relacionados à mudança.

O Teste funcional envolve a validação de funções para que esses estejam de acordo com o esperado para o software. Ele pode ser realizado em todos os níveis. Já para o Teste não funcionado o foco é verificar o estado do sistema, ou seja, questões como performance, usabilidade e segurança são fundamentais.

No Teste de caixa-branca a estrutura interna e arquitetura do produto são validadas, assim atividades que analisam o código e fluxo de dados da aplicação são realizadas para identificar possíveis defeitos. Os Testes relacionados à mudança são testes que são feitos quando ocorrem alterações no software a fim de garantir que as novas modificações não introduziram ou desencadearam novas falhas. Essas mudanças incluem correções de defeitos anteriormente encontrados ou a atualização do sistema.

2.3. Formas de Execução de testes

Quando se trata de execução de testes tem-se duas maneiras diferentes de serem realizados. Essas formas recebem os nomes de Testes manuais e Testes automatizados.

Os testes manuais se caracterizam por ser uma atividade de teste que necessita da análise humana, ou seja, se orienta pelo fato de que o humano irá julgar se o comportamento mostrado representa algo que é esperado ou não. Por ser uma atividade manual possibilita que sejam identificadas algumas falhas que não seriam detectadas por testes automatizados.

Já os testes automatizados utilização de ferramentas para a criação e execução de *scripts* afim de realização do fluxo de teste feito manualmente e validá-lo. Dessa forma, há um grande aproveitamento de tempo pois torna-se possível um maior número de repetição de testes além de poderem ser executados ao mesmo tempo. Porém com dito anteriormente para testes automatizado é necessário manter a manutenção e atualização dos cenários para que eles possam continuar eficientes.

Ao comparar ambas as vertentes de testes, podem ser observadas algumas diferenças:

Figura 6 - Comparação entre testes manuais e automatizados

Teste manual de software	Teste automatizado de software
Está sujeito a erros humanos	Oferece mais assertividade por ser realizado por máquinas
Possibilita testes fiéis de usabilidade	Não consegue simular uma situação real do usuário
Possibilita encontrar bugs extras devido à atuação analítica	Possibilita agilidade e repetição de procedimentos sem exaustão

Fonte: Testing Company (https://www.testingcompany.com.br/blog/teste-manual-de-software-ou-automatizado-qual-leva-a-melhor, acessado em julho de 2023)

Os testes automatizados necessitam que as telas e elementos que são integrados sejam mapeados a fim de que a ferramenta utilizada possa executar as ações assim como o usuário as realiza. Outro ponto importante é que se torna possível a geração de imagens/evidências e relatório que contém o fluxo e o resultado das validações feitas pela automação.

Para esses testes existem diversas ferramentas baseadas em diferentes linguagens de programação. Dentre elas, tem-se:

1) Java com uso do *framework* Selenium;
2) Python com uso do *framework* Robot;
3) JavaScript com o uso do Cypress;

2.4. Automação de testes para aplicações web

A automação de testes em aplicações web é uma estratégia imperativa adotada em processos de criação de software para aprimorar a eficiência e a precisão no processo de verificação da aplicação. Essa prática envolve o uso de ferramentas especializadas, como Selenium e Cypress, para automatizar testes, permitindo a execução rápida e repetida de cenários específicos.

A utilização do processo automatizado de teste implica em diversos benefícios como o fato de que a velocidade de execução é aumentada significativamente, possibilitando uma cobertura abrangente de testes em um curto espaço de tempo. Além disso, a consistência na execução dos testes é garantida, eliminando erros humanos e promovendo a confiabilidade nos resultados.

A detecção precoce de defeitos é uma das principais contribuições da testagem automatizada. Ao automatizar testes de regressão, por exemplo, é possível identificar rapidamente possíveis impactos adversos em novas funcionalidades ou alterações no código, facilitando correções imediatas.

Em um cenário de desenvolvimento ágil, a automação de testes é crucial para a integração contínua, permitindo a execução automática de testes sempre que há uma atualização no código-fonte. Isso não apenas agiliza o processo de entrega, mas também assegura que a qualidade seja mantida ao longo do ciclo de vida do software.

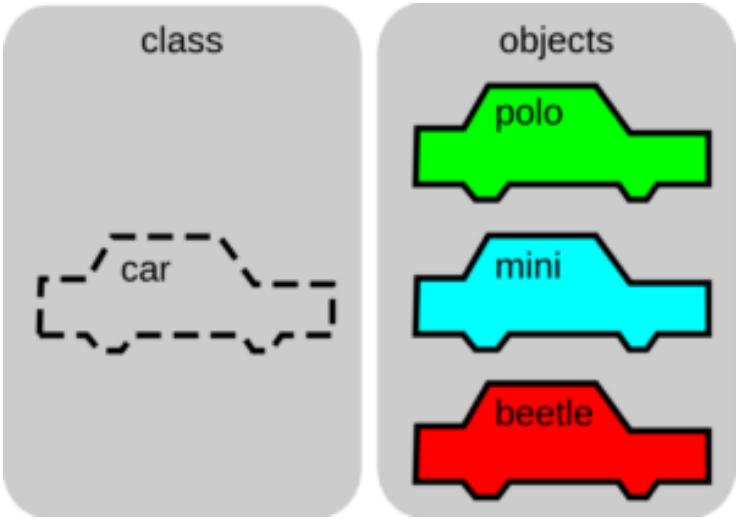
Na automação de testes web a identificação de elementos da tela é um passo crucial, pois envolve a localização e interação com os elementos da interface do usuário, como botões, campos de texto e links. O mapeamento preciso é essencial para que os scripts de automação possam realizar ações específicas nos elementos corretos.

Em suma, a automação de testes em aplicações web se revela como uma abordagem essencial para os analistas de qualidade, promovendo eficiência, consistência e confiabilidade nos processos de verificação de software, contribuindo diretamente para o sucesso e a qualidade das aplicações desenvolvidas.

2.5. Programação orientada a objetos

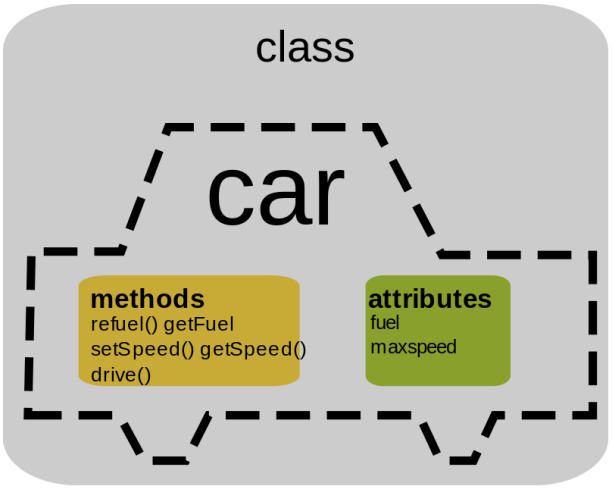
A criação de projetos, recentemente, tem usado com frequência a programação orientada a objetos (POO), pois ela permite uma maior proximidade entre os objetos criados no mundo virtual e os existentes na realidade. Com sua utilização é possível desenvolver classes que são a representação de um conjunto de propriedades em comum de um grupo de elementos observados, e a partir disso são construídas unidades menores, os objetos, que demonstram um agregado de atributos e comportamentos que podem ser específicos ou gerais.

Figura 7 - Exemplificação de classes e objetos em POO



Fonte: Alura (https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos, acessado em julho de 2023)

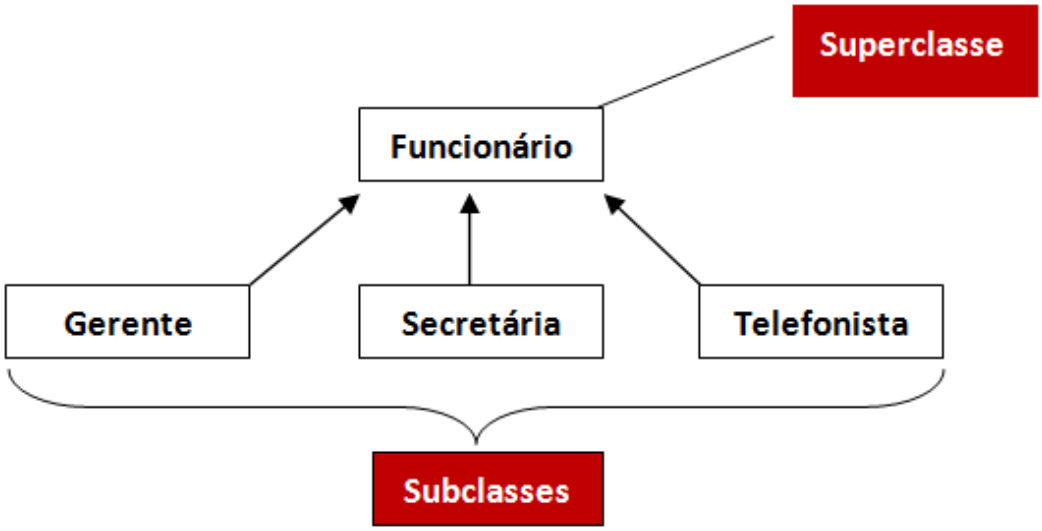
Figura 8 - Exemplificação de atributos e métodos em POO



Fonte: Alura (https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos, acessado em julho de 2023)

Por fim, é valido ressaltar a concepção de subclasses que herdam métodos de uma superclasse criada para ser um modelo mais genérico.

Figura 9 - Conceito de classes para POO



Fonte: DevMedia (https://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366, acessado em julho de 2023)

Assim, pode-se observar que com o uso POO faz-se possível tornar um código mais fiel a realidade e flexível. Entretanto essa não é uma tarefa fácil, pois ela requer um bom entendimento do problema real juntamente com seus elementos e demanda soluções para problemas específicos com uso de mecanismos mais gerais.

2.6. Padrões de Projeto

O desenvolvimento de um projeto é uma atividade bastante difícil, quando ela é executada com o objetivo de tornar o código final reutilizável e flexível. Estes dois fatores citados são importantes, pois possibilita o reaproveitamento do código caso haja necessidade em algum momento no futuro.

A reutilização de código consiste em usar soluções já desenvolvidas e documentadas para resolver impedimentos enfrentados. Dessa forma, a construção do projeto passa a ser mais dinâmica e menos complexa.

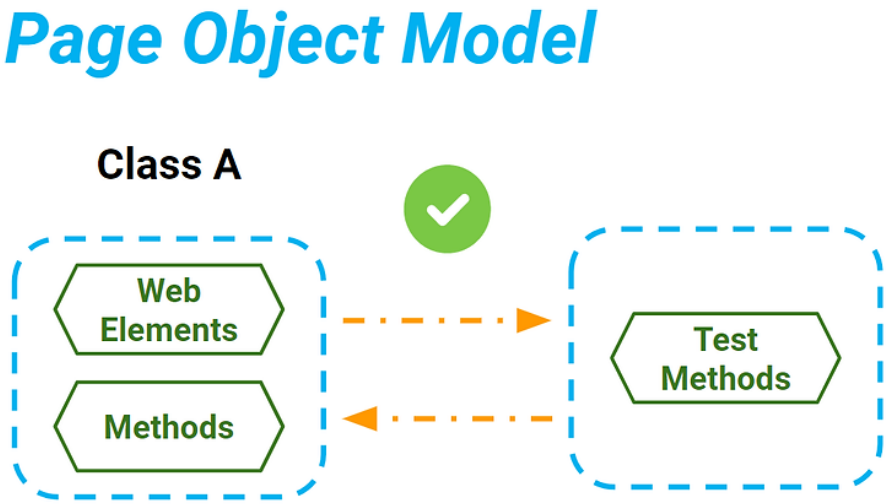
A partir disso, a utilização de modelos poderia auxiliar e tornar mais uniforme o processo de desenvolvimento de código. Em conformidade a esse pensamento em 1995 um grupo de 4 cientistas conhecidos como Gangue dos Quatro (*Gang of Four*

ou *GoF*) composta por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides introduziram esse conceito para os projetos de software com a publicação do livro chamado Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos. Dessa forma, a ideia de utilizar padrões de projetos para o desenvolvimento de aplicações se tornou algo extremamente importante.

Os padrões de projetos são modelos que foram aplicados e validados para o processo de desenvolvimento de uma aplicação. É essencial considerar que para cada contexto faz-se necessária uma análise de qual deles seria mais bem utilizado. Para o assunto de testes um dos padrões mais utilizados é o Objeto de Página (Page Object ou PO).

Para o padrão Objeto de Página o objetivo é encapsular os atributos e métodos em seus respectivos grupos dentro do projeto a fim de separar elementos pertencentes ao sistema e as especificações dos testes. Assim, modificações e atualizações se tornam menos complexas pois podem ser feitas de forma independente quando se trata de aspectos do software ou referentes as atividades de validações.

Figura 10 - Padrão de Projeto PO para testes



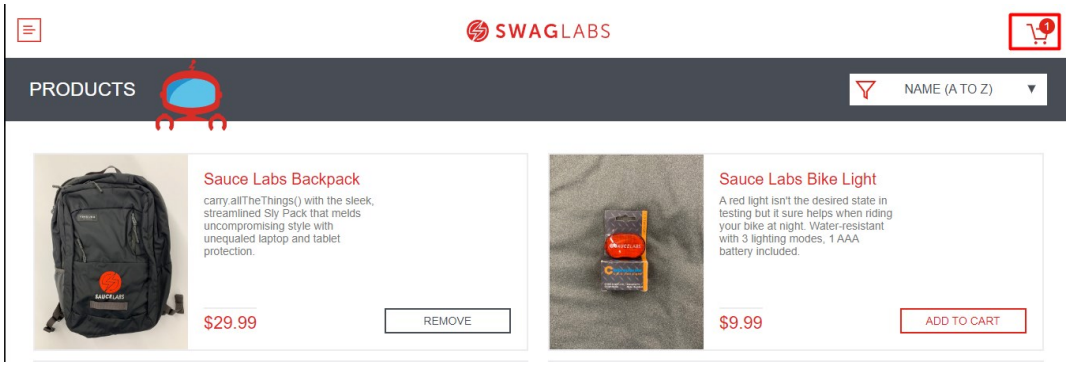
Fonte: Automated Testing (https://www.automatedtestingwithtuyen.com/post/page-object-model-pattern, acessado em julho de 2023)

3. METODOLOGIA

Neste capítulo será feito o desenvolvimento do objetivo proposto, que inclui o detalhamento das ferramentas, sua complexidade e usabilidade para criação de testes de aceitação em aplicações web. Além disso, serão discutidas as linguagens de programação utilizadas.

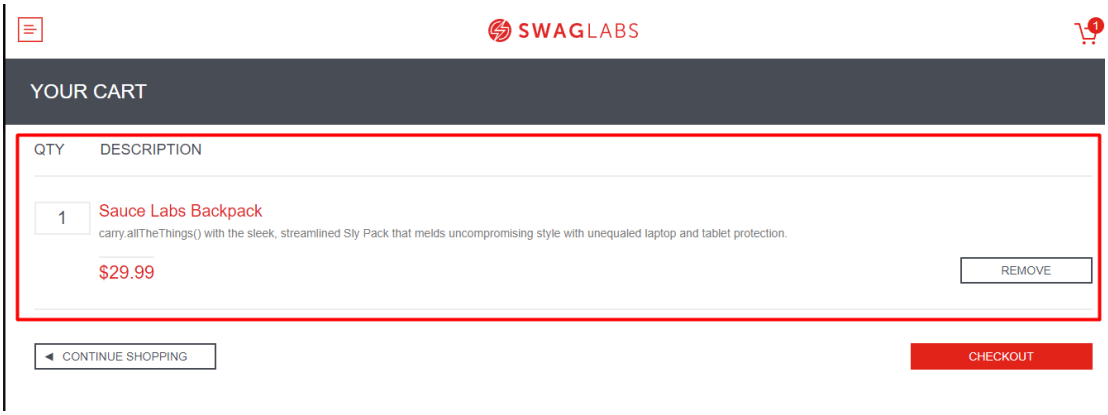
Tem-se como fonte de análise a aplicação de vendas destinada para a prática de testes automatizados denominada SWAGLABS hospedada no seguinte endereço de internet: https://www.saucedemo.com/. O cenário de teste avaliado pelas ferramentas será o fluxo em que o usuário inicia com acesso e realiza a inclusão de um item desejado para compra, a fim de validar se o item foi corretamente adicionado.

Figura 11 - Adição do produto desejado ao carrinho



Fonte: Adaptado de SWAGLABS (https://www.saucedemo.com/, acessado em julho de 2023)

Figura 12 - Validação do item incluído ao carrinho



Fonte: Adaptado de SWAGLABS (https://www.saucedemo.com/, acessado em julho de 2023)

3.1. Java com framework Selenium

Java é uma linguagem de programação desenvolvida inicialmente por James Goslining em 1995. Atualmente o Java conta com diversas características dentre eles a robustez, segurança e alta performance. É importante ressaltar que é uma linguagem que independe de plataforma, ou seja, a aplicação pode ser executada em diferentes sistemas operacionais.

O Selenium é uma ferramenta para criação de testes de aceitação automatizados, pois permite uma integração com o navegador a fim de simular os fluxos que o usuário realiza na aplicação e assim validá-los. Para utilizá-lo com o java faz-se necessário que o ambiente de desenvolvimento esteja preparado com os seguintes componentes:

1) Instalação do Java *Development Kit* (JDK);
2) Instalação de um Ambiente de desenvolvimento integrado (IDE);
3) O driver do navegador de internet;
4) Adição dos recursos, como o *framework* Selenium, para criação de testes no projeto.

A presença do JDK faz-se necessária pois ele é um conjunto de instrumentos que permitem a criação de aplicações com a linguagem Java. Já a IDE é uma ferramenta que integra vários recursos importantes para o desenvolvimento de código agilizando esse processo.

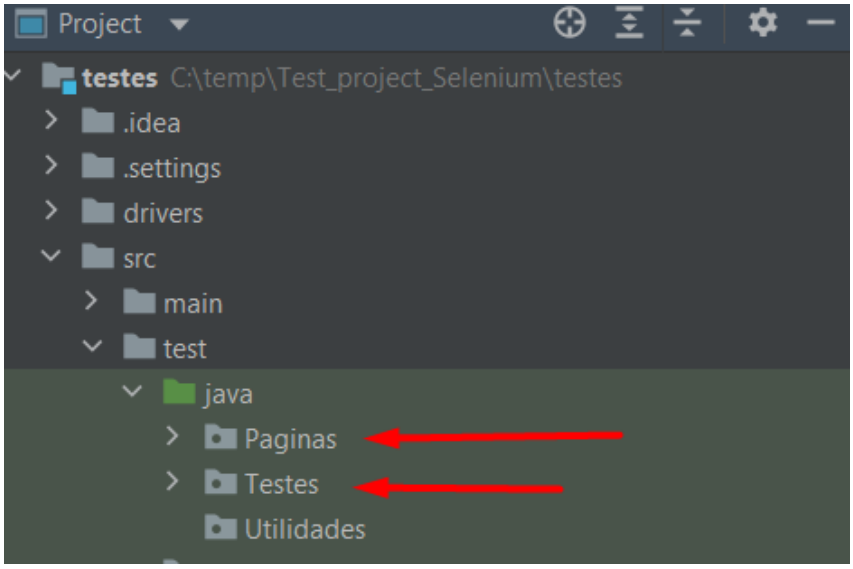
Da mesma forma que o cliente executa os fluxos do produto utilizando o browser, o Selenium também precisa utilizá-lo para simular as ações e executar os testes. Assim, para obter esse acesso o framework utiliza o driver do navegador desejado para realizar essas atividades de forma automatizada.

Por fim, para realizar e estruturar as validações o Java possibilita que sejam adicionadas ao projeto alguns recursos, dentre eles o Selenium, permitindo que classes e métodos externos possam ser utilizados no código. Dessa forma, o reaproveitamento de soluções que já foram criadas se torna mais simples e a produção do software mais eficiente.

3.1.1. Codificando o teste

Ao iniciar o projeto é importante estruturá-lo com o uso do padrão Objeto de Página a fim de separar códigos referentes as páginas do produto e os testes. Essa divisão é feita por meio da criação das pastas Páginas e Testes respectivamente.

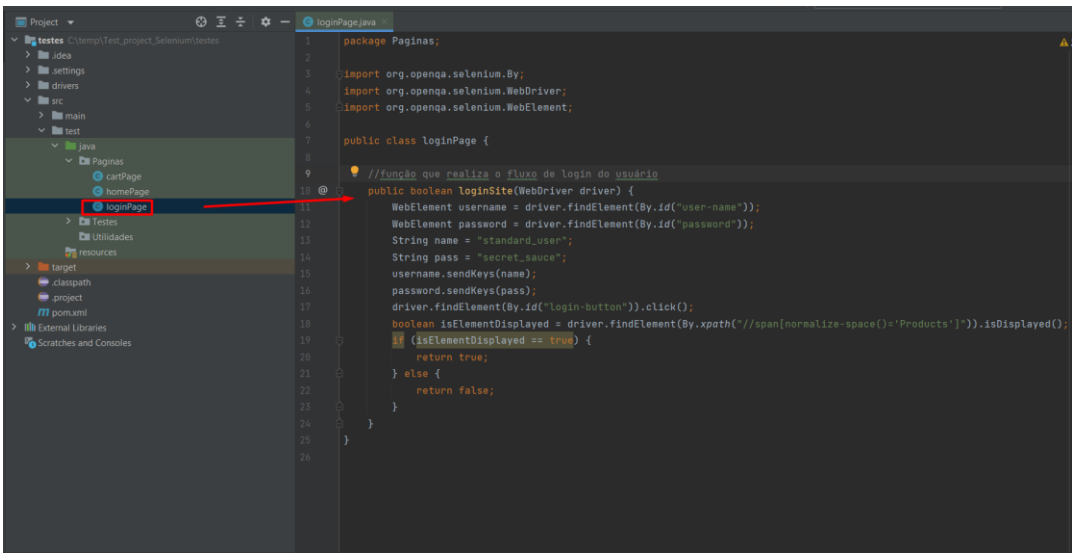
Figura 13 - Estrutura de PO em um projeto com Java e Selenium



Fonte: Autor

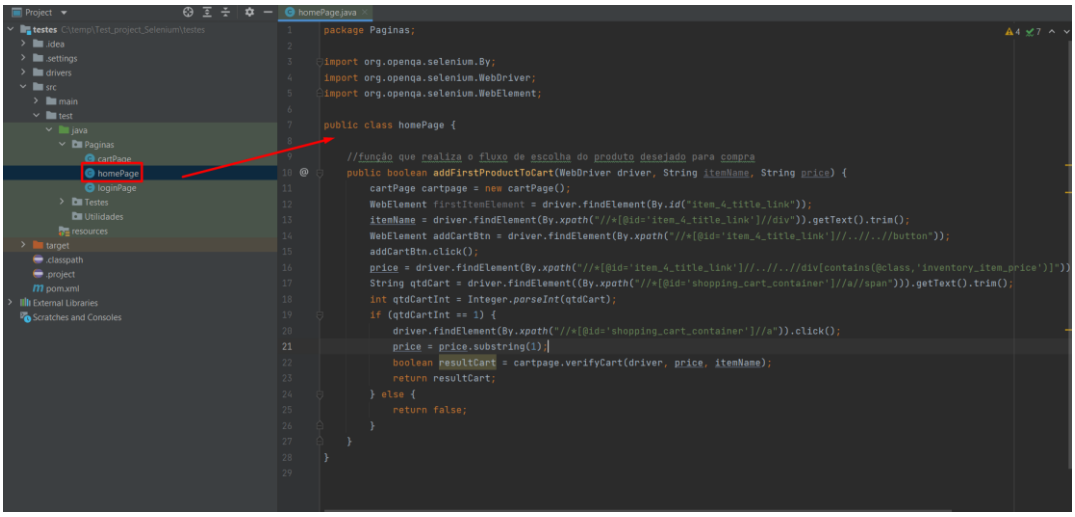
Em seguida, feita análise do cenário a ser testado são criadas algumas classes para representar cada um dos trechos do sistema em que ocorre iteração. Elas contêm métodos específicos que emulam as ações realizadas na respetiva página.

Figura 14 - Funções codificada para página de login



Fonte: Autor

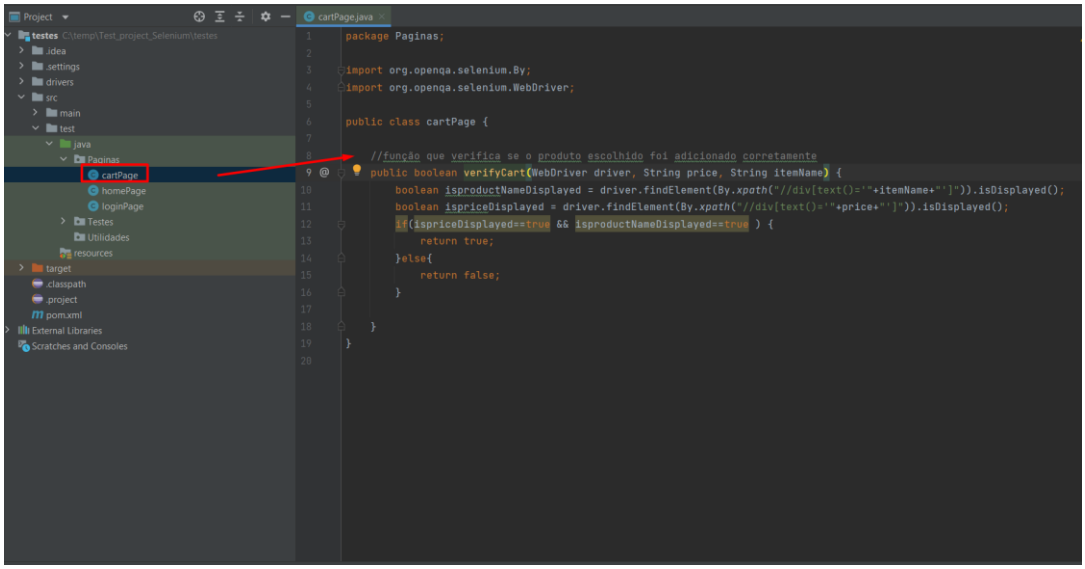
Figura 15 - Funções codificada para página de principal



```
package Paginas;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class homePage {

    //função que realiza o fluxo de escolha do produto desejado para compra
    public boolean addFirstProductToCart(WebDriver driver, String itemName, String price) {
        cartPage cartpage = new cartPage();
        WebElement firstItemElement = driver.findElement(By.id("item_4_title_link"));
        itemName = driver.findElement(By.xpath("//*[@id='item_4_title_link']//div")).getText().trim();
        WebElement addCartBtn = driver.findElement(By.xpath("//*[@id='item_4_title_link']//../..//button"));
        addCartBtn.click();
        price = driver.findElement(By.xpath("//*[@id='item_4_title_link']//../..//div[contains(@class,'inventory_item_price')]")).
        String qtdCart = driver.findElement((By.xpath("//*[@id='shopping_cart_container']//a//span"))).getText().trim();
        int qtdCartInt = Integer.parseInt(qtdCart);
        if (qtdCartInt == 1) {
            driver.findElement(By.xpath("//*[@id='shopping_cart_container']//a")).click();
            price = price.substring(1);
            boolean resultCart = cartpage.verifyCart(driver, price, itemName);
            return resultCart;
        } else {
            return false;
        }
    }
}
```

Fonte: Autor

Figura 16 - Funções codificada para página do carrinho

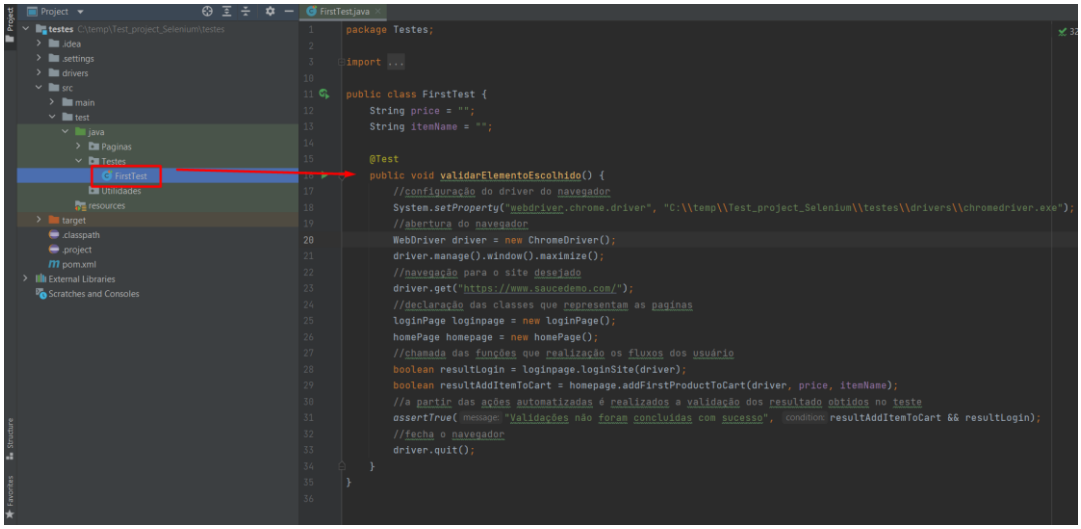


```
package Paginas;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class cartPage {

    //função que verifica se o produto escolhido foi adicionado corretamente
    public boolean verifyCart(WebDriver driver, String price, String itemName) {
        boolean isproductNameDisplayed = driver.findElement(By.xpath("//div[text()='"+itemName+"']")).isDisplayed();
        boolean ispriceDisplayed = driver.findElement(By.xpath("//div[text()='"+price+"']")).isDisplayed();
        if(ispriceDisplayed==true && isproductNameDisplayed==true ) {
            return true;
        }else{
            return false;
        }

    }
}
```

Fonte: Autor

Por fim, dentro da pasta de Testes é estruturada a classe que possui o código da validação desejada. Ela utiliza dos métodos feitos para as páginas a fim de que juntas simulem todo o fluxo do usuário. Além disso, também é realizada a análise dos resultados obtidos com as ações para gerar o resultado do teste.

Figura 17 - Codificação do cenário da validação



Fonte: Autor

Para a geração de evidências e relatórios das iterações realizadas pela automação faz-se necessário a codificação de novos métodos dentro do projeto. A criação dessas funções pode ser feita utilizando alguns recursos do Selenium que são direcionados para executar esse tipo de atividade.

3.2. Python com *framework Robot*

O Python é uma linguagem bastante popular e tem como uma de suas características ser mais amigável ao usuário, por meio de uma estrutura clara e voltada para o entendimento humano tornando-se mais intuitiva. Ele é uma linguagem de programação orientada a objetos, tem seu código aberto e possibilita criar atividades diversas desde simples cálculos até algoritmos de inteligência artificial.

Já o *framework Robot* é uma ferramenta de automação de processos que pode ser utilizada para atividades direcionadas a testes. Ele é estruturado de forma a permitir a utilização de palavras chaves que representam ações já codificadas (*keyword-driven testing approach*). Além disso possibilita a criação de novos termos para funções especificas, fazendo com que a leitura do script seja entendível a qualquer pessoa.

Para criar automações com Python e o *framewok* Robot é necessário que o ambiente de desenvolvimento possua os seguintes elementos:

1) Instalação de uma versão do Python;
2) Instalação de um Ambiente de desenvolvimento integrado (IDE);
3) Instalação do *framework* Robot e os recursos para criação de testes;
4) O driver do navegador de internet.

A instalação de uma versão do Python admite que seja possível criar e executar projetos escritos com essa linguagem. Assim como mencionado anteriormente, a presença de um ambiente de desenvolvimento integrado é necessária para que não só a codificação, mas também a estruturação da aplicação seja mais eficiente.

O *framework* Robot é um elemento que deve ser adicionado após a configuração do Python pois ele é um conjunto de soluções que será utilizado juntamente com a linguagem de programação. Em seguida, recursos específicos para atividades de validação podem ser incluídos a aplicação para tornar possível a utilização de soluções já desenvolvidas em outras classes.

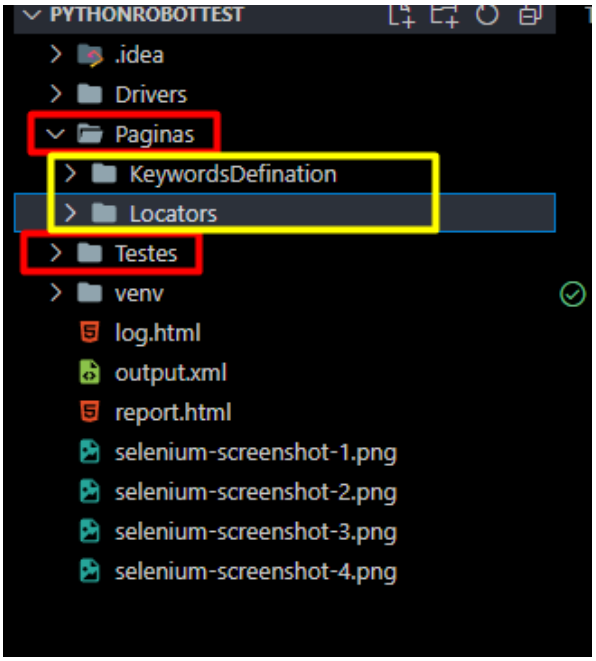
Por fim, para similar os fluxos do produto assim como o usuário os executam é necessário que haja o acesso ao navegador de internet. O Python e o *framework Robot* fazem essa conexão por meio do driver do browser escolhido.

3.2.1. Codificando o teste

A construção da aplicação tem como referência o padrão de projeto Objeto de Página. Na linguagem Python com o uso do *framework* Robot a estrutura é feita por meio da criação de pastas para isolar codificações relacionas as páginas e aos testes. Entretanto, dentro do contexto das páginas tem-se duas categorias diferentes chamadas de *Locators* e *KeywordsDefination*.

Em *Locators* encontra-se o mapeamento dos elementos que serão interagidos nas validações em cada tela. Já em *KeywordsDefination* existem a codificação de novas palavras chaves que representam ações em uma respectiva página.

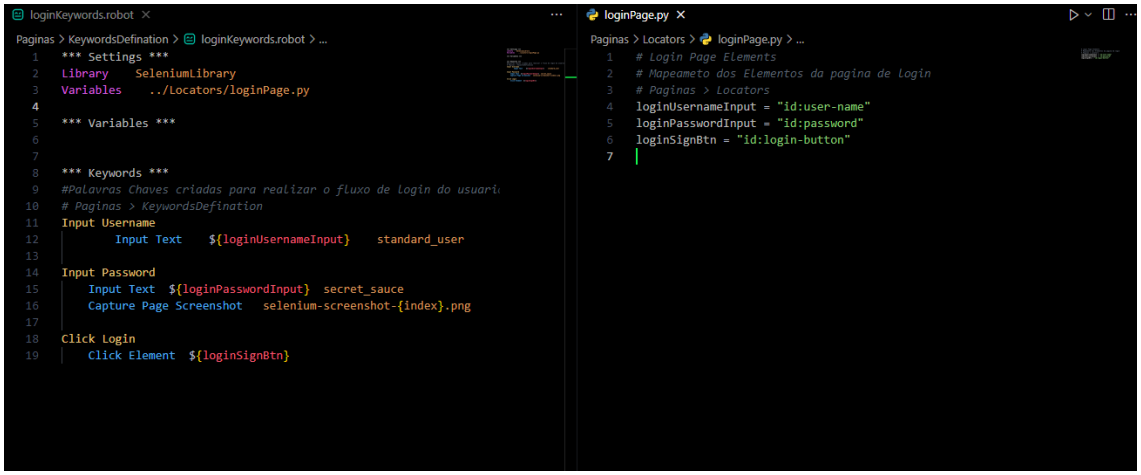
Figura 18 - Padrão PO para o projeto Python com framework Robot



Fonte: Autor

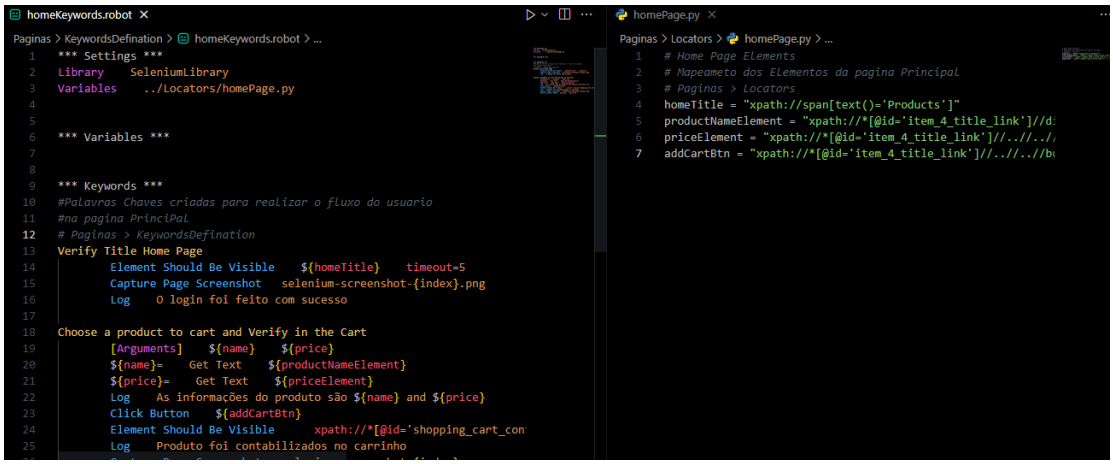
Posteriormente, ao compreender as atividades a serem validadas são criados alguns arquivos com a extensão .py para representar o mapeamento de elementos das páginas e suas funções e outros do tipo .robot que contém a codificação de novas palavras chaves criadas.

Figura 19 - Codificação das novas palavras chaves e mapeamentos dos elementos para tela de login



Fonte: Autor

Figura 20 - Codificação das novas palavras chaves e mapeamentos dos elementos para tela de principal



Fonte: Autor

Figura 21 - Codificação das novas palavras chaves e mapeamentos dos elementos para tela do carrinho

Fonte: Autor

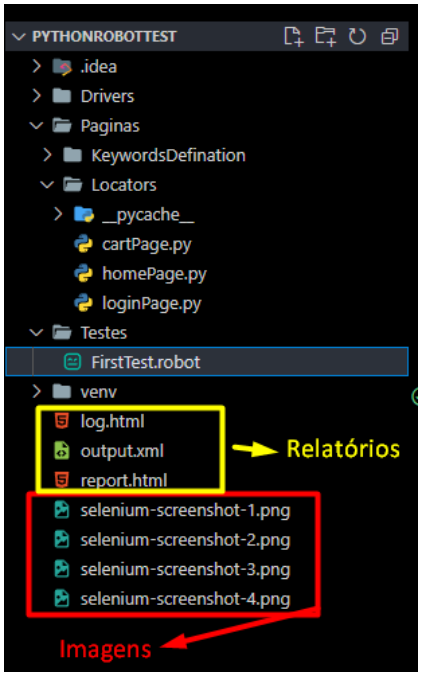
Ao final, na pasta Testes é criado o arquivo com a extensão *.robot* ,que possui o código da validação desejada. O teste é codificado com o uso de palavras chaves facilitando o entendimento de qualquer pessoa que queira ler o *script*.

Figura 22 - Codificação do cenário de teste

Fonte: Autor

Um ponto importante a ser destacado é o fato de que a geração de evidências e relatórios das iterações feitas pela automação pode ser feita de forma bem simples e rápida por meio do uso das palavras chaves que são destinadas para essa funcionalidade, como por exemplo *Capture Page Screenshot*. Dessa forma, imagens e documentos são gerados ao fim da execução do cenário de teste.

Figura 23 - Evidências e Relatórios gerados



Fonte: Autor

3.3. JavaScript com Cypress

A linguagem JavaScript é bastante utilizada para aplicações de internet porque consegue tornar as páginas mais iterativas fazendo com que a experiencia do usuário seja mais agradável durante a navegação. Entretanto seu uso não se restringe somente a isso, pois ele pode ser aplicado em diversas atividades como automação de tarefas, banco de dados, criação de jogos, entre outras áreas.

O JavaScipt é suportado por todos os navegadores, ou seja, eles conseguem executar códigos escritos nessa linguagem sem a necessidade de recursos adicionais. Em situações em que o browser não é o ambiente de desenvolvimento torna-se necessária a instalação do software chamado Node.js para que ele realize a execução dos scripts criados.

O Cypress é um framework para atividades de testes que pode ser aplicado em todos os níveis e utiliza a linguagem JavaScript. Ele é uma ferramenta de fácil configuração e permite que os resultados das validações feita possam ser visualizados durante a execução dos testes.

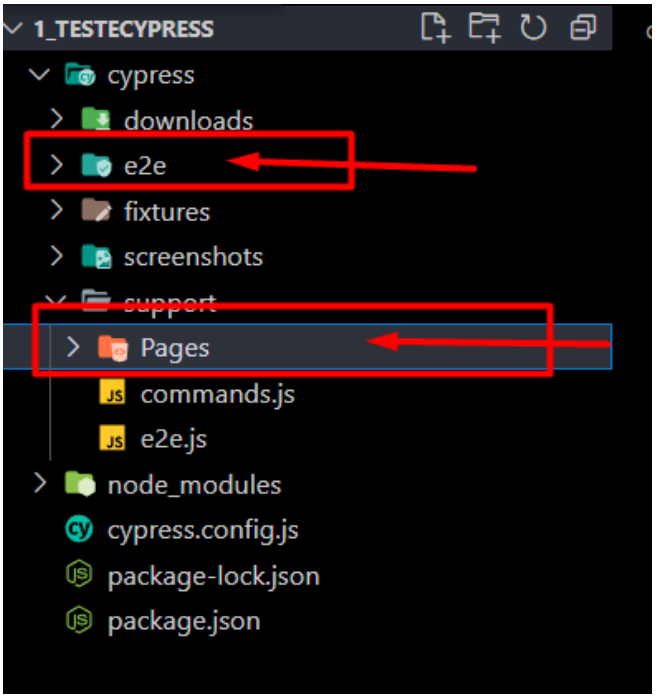
De acordo com o que foi discutido, para iniciar as automações de testes com o Cypress faz-se essencial que o ambiente de desenvolvimento seja preparado com os seguintes itens:

1) Instalação do Node.js;
2) Instalação de um Ambiente de desenvolvimento integrado (IDE);
3) Instalação do Cypress no local do projeto;

3.3.1. Codificando o teste

A organização do projeto é feita por meio da adaptação do padrão Objeto de Página em relação a estrutura utilizada pela ferramenta em questão. Dessa forma, para segregar os cenários de validação e os elementos das páginas forma criadas pastas chamadas *e2e* e *Pages*, respectivamente.

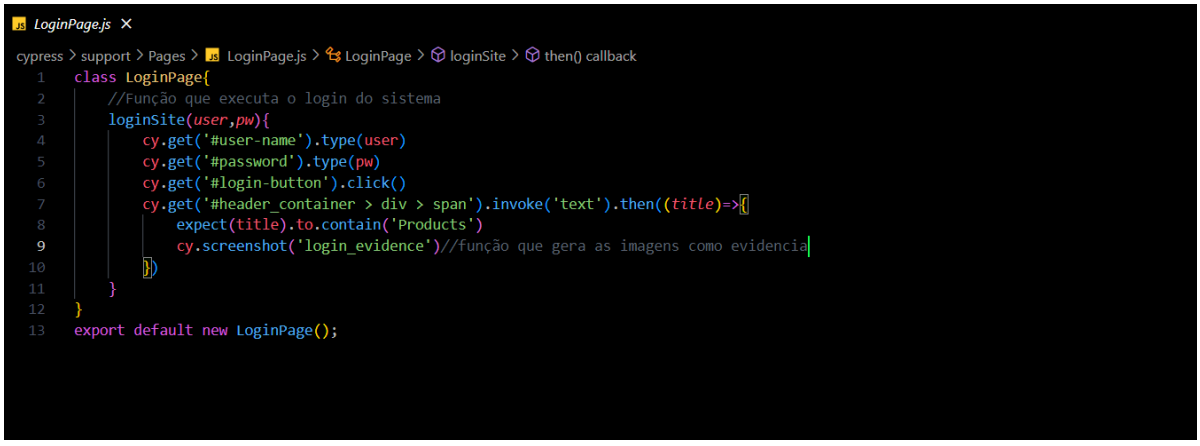
Figura 24 - Estrutura de PO para um projeto JavaScript com Cpress



Fonte: Autor

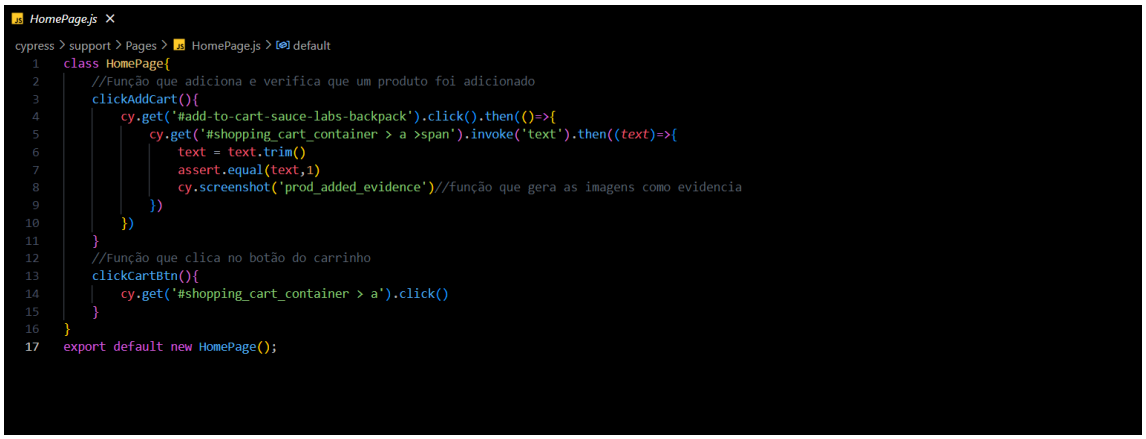
Ao analisar o fluxo que deve ser avaliado são estruturados os códigos que representam cada tela do sistema. Esses *scripts* são arquivo do tipo JavaScript(.js)

Figura 25 - Código dos métodos da página de login



Fonte: Autor

Figura 26 - Código dos métodos da página principal



```
HomePage.js
cypress > support > Pages > HomePage.js > default
class HomePage{
    //Função que adiciona e verifica que um produto foi adicionado
    clickAddCart(){
        cy.get('#add-to-cart-sauce-labs-backpack').click().then(()=>{
            cy.get('#shopping_cart_container > a >span').invoke('text').then((text)=>{
                text = text.trim()
                assert.equal(text,1)
                cy.screenshot('prod_added_evidence')//função que gera as imagens como evidencia
            })
        })
    }
    //Função que clica no botão do carrinho
    clickCartBtn(){
        cy.get('#shopping_cart_container > a').click()
    }
}
export default new HomePage();
```

Fonte: Autor

Figura 27 - Código dos métodos da página do carrinho

```
CartPage.js
cypress > support > Pages > CartPage.js > CartPage > validarPrecoProd > then() callback
class CartPage{
    //Função que valida o nome do produto adicionado
    validarNomedoProduto(nome){
        cy.xpath('//div[text()="'+nome+'"]').should('be.visible')
        cy.screenshot('name_prod_evidence')//função que gera as imagens como evidencia
    }
    //Função que valida o preço do produto adicionado
    validarPrecoProd(preco){
        cy.xpath('//*[@id="item_4_title_link"]//..//../div[contains(@class,"pricebar")]//div').invoke('text').then((text)=>{
            assert.equal(text,preco)
            cy.screenshot('price_prod_evidence')//função que gera as imagens como evidencia
        })
    }
}
export default new CartPage();
```

Fonte: Autor

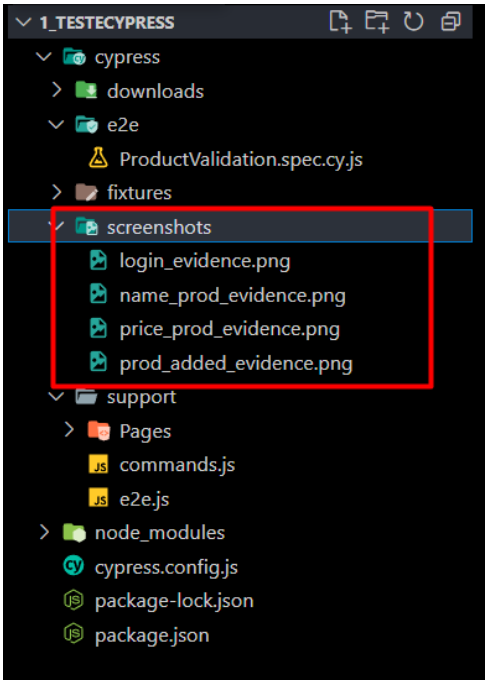
Posteriormente, o teste é codificado seguindo a estrutura da ferramenta por meio da criação de um arquivo do tipo .spec.js.cy. Nele todas as funcionalidades que foram anteriormente sintetizadas podem ser chamadas a fim de que juntas simulem e validem o contexto analisado.

Figura 28 - Codificação do cenário de teste

Fonte: Autor

O Cypress possui métodos já criados para a geração de imagens do sistema durante a execução da automação. Uma das funções que pode ser usada para isso é chamada *cy.screenshot()*.

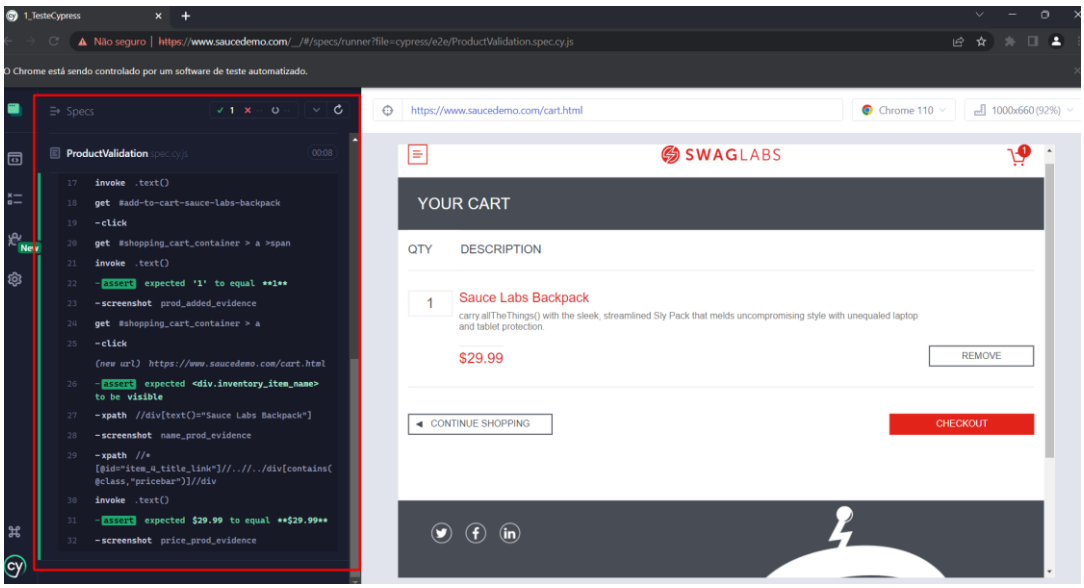
Figura 29 - Geração das imagens pelo Cypress



Fonte: Autor

Outro ponto importante é que durante a execução dos testes é possível acompanhar o resultado das validações conforme forem sendo feitas de forma visual por meio da interface do Cypress. Com essa visão se torna mais fácil e intuitivo identificar possíveis defeitos.

Figura 30 - Resultados das validações durante a execução



Fonte: Autor

4. RESULTADOS E DISCUSSÕES

Após a realização da mesma atividade de testes com a utilização de diferentes ferramentas, é possível ressaltar pontos positivos e negativos para cada uma delas. Para classificar essas tecnologias foi proposto um indicador que categoriza a maturidade das ferramentas com base no nível de esforço necessário para implementação. Este indicador que cataloga essas ferramentas possui três níveis: fácil (1), intermediário (2) e avançado (3). A metodologia para atribuir esses níveis leva em consideração a curva de aprendizado, a facilidade de configuração e tempo de codificação.

O Selenium com Java possui alguns aspectos positivos como o fato de ser independente de sistema operacional e poder acessar diversos navegadores de internet. Essa ferramenta permite que vários recursos adicionais possam ser incorporados ao projeto possibilitando a automação de ações mais complexas. Ela é um recurso validado e bastante utilizado, assim podem ser encontradas em grande quantidade documentações uteis para solução de problemas e estratégias desenvolvidas que podem ser reutilizadas.

Entretanto, para não subutilizar o Selenium é necessário um conhecimento prévio de lógica de programação e da linguagem utilizada. O processo de configuração da ferramenta não é tão simples e demanda atenção. Além disso, a geração de relatórios e imagens das atividades realizadas durante a execução da automação torna-se complicada, pois essas funcionalidades demandam a criação de novos métodos os quais precisam que estruturas externas sejam incluídas ao projeto. Com base nessas informações, conclui-se que o Selenium com o Java possui o indicador 3.

Já o framework Robot com o Python tem sua usabilidade caracterizada por ser fácil e intuitiva, permitindo que pessoas com baixo conhecimento técnico aprendam rapidamente a usar ferramenta. Por ser uma fermenta que utiliza a estrutura de palavras chaves torna o código mais próximo da linguagem humana facilitando seu entendimento.

A documentação oficial da ferramenta é bastante enriquecedora tanto para aprendizagem quanto para consulta durante escrita do código. Outro ponto positivo é

que já existem recursos disponíveis para a criação de documentos e imagens que sintetizam os resultados obtidos pela execução dos fluxos automatizados.

Um aspecto negativo referente ao Robot é o fato de que ele não suporta o uso de elementos como estruturas condicionais e de repetição durante a codificação. Esses itens são bastante utilizados quando os projetos se tornam complexos, pois possibilitam direcionar ações e repetir atividades de maneira mais simples. Em resumo, os elementos discutidos corroboram para que seja atribuído o indicador 2 ao framework Robot com o Python.

O Cypress com o JavaScript conta também com o fato de poder ser utilizado em diversos navegares, fácil configuração e permiti que os resultados das validações possam ser visualizados ao mesmo tempo que a execução da automação é feita. Ele possui não só recursos que facilitam o mapeamento dos elementos das páginas, acelerando o processo de codificação, mas também funcionalidades para a geração de imagens durante a execução das validações estão disponíveis para uso. Além disso, a linguagem utilizada não é difícil de aprender.

Entretanto, em algumas situações o Cypress apresenta algumas limitações com por exemplo quando há a necessidade da abertura de mais de uma aba no navegador. Outro ponto é o fato de que para projetos mais complexos o uso da linguagem JavaScript torna a codificação proporcionalmente mais complicada. Portanto, pode-se afirmar que Cypress com o JavaScript conta com o indicador 1.

Tabela 1 - Resultados da análise entre as ferramentas

Ferramenta/Indicador	Fácil (1)	Intermediário (2)	Avançado (3)
Java c/ Selenuim			x
Python c/ Robot		x	
JavaScript c/ Cypress	x		

5. CONCLUSÃO

Todos os pontos evidenciados, no capítulo anterior, foram significativos para o desenvolvimento e atingimento da meta principal do trabalho. E graças a eles, foi possível realizar a análise das ferramentas de automação de testes apresentadas e obter uma resposta concreta referente a qual ocasião utilizá-las.

Dessa forma, em um contexto em que o conhecimento técnico é baixo e o projeto não apresenta grande complexidade, a ferramenta Cypress com JavaScript poderia ser utilizada para a automação das validações. Fatores como uma configuração fácil e recursos que ajudam na escrita do código tornam ela adequada e eficiente para esse tipo de situação.

Quando já se tem uma compreensão técnica intermediária, o Robot *framework* com o Python se mostra uma opção mais convidativa pois possibilita codificações de novos fluxos previamente não existentes e apresenta recursos nativos para síntese de evidências. Além disso, a linguagem utilizada é bastante intuitiva e fácil de aprender.

Em casos que o sistema já possui certa complexidade e a base técnica está compreendida dentre um nível intermediário ou mais elevado, a escolha de usar o Selenium com o Java é mais pertinente. Alguns pontos que corroboram para isso são a possibilidade de codificação até mesmo de ações mais complicadas e a vasta quantidade de documentações para auxílio durante o desenvolvimento.

Importante ressaltar que para situações em que já existem estruturas codificadas a serem herdadas, é essencial realizar uma avaliação cuidadosa a fim de verificar se é viável a troca de ferramenta de automação.

Desse modo, o estudo para a escolhas de recursos para automação de testes foi importante pois permitiu explicitar os momentos em que determinada ferramenta poderia ser mais apta a fim de tornar mais eficiente a criação e execução das validações.

6. REFERÊNCIAS

SOUZA, Ludmilla. **Indústria de Software e Serviços de TIC cresceu 6,5% em 2021**. Agência Brasil. Disponível em: https://agenciabrasil.ebc.com.br/economia/noticia/2022-07/industria-de-software-e-servicos-de-tic-cresceram-65-em-2021. Acesso em: 25 de julho de 2023.

CLARO, Daniela Barreiro; Sobral, João Bosco Mangueira. **Programação em JAVA**. Disponível em: https://www.faeterj-rio.edu.br/downloads/bbv/0031.pdf. Acesso em: 25 de julho de 2023.

BORGES, Luiz Eduardo. **Python para Desenvolvedores**. Disponível em: https://ark4n.files.wordpress.com/2010/01/python_para_desenvolvedores_2ed.pdf . Acesso em: 26 de julho de 2023.

The Pesticide Paradox as explained by Boris Beizer. Disponível em: https://infiniteundo.com/post/87158389858/pesticide-paradox. Acesso em: 27 de julho de 2023.

GAMMA, ERIC et al. **Padrões de Projeto**. Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

ZARELLI, Guilherme Biff. **Java, JUnit e Selenium WebDriver — Configurando Ambiente de Automação.** Medium, 2020. Disponível em: https://medium.com/luizalabs/pir%C3%A2mide-de-testes-definindo-uma-boa-su%C3%ADte-de-testes-para-seu-software-a6864886f29b#:~:text=A%20Pir%C3%A2mide%20de%20Testes%20%C3%A9,seus%20testes%20em%20diferentes%20camadas. Acesso em: 26 de julho de 2023.

JÚNIOR, Paulo. **Pirâmide de Testes — Definindo uma boa suíte de testes para seu Software.** Medium, 20218. Disponível em: https://medium.com/@paulosajunior/java-junit-e-selenium-webdriver-configurando-ambiente-de-automa%C3%A7%C3%A3o-7468ef1452d1#:~:text=Clicar%20com%20o%20bot%C3%A3o%20direito%20na%20classe%20(arquivo)%20ConfereConfigProjeto.,est%C3%A1%20preparado%20para%20o%20treinamento. Acesso em: 20 de julho de 2023.

KHAN, Hammad Ahmed. **Robot Framework: Creating Custom Keywords in Python**, 2021. Disponível em: https://medium.com/swlh/robot-framework-creating-custom-keywords-78786caa6f89. Acesso em: 20 de julho de 2023.

STAFFEN, Guilherme. **Conheça o Cypress e suas vantagens para automação de testes**, 2021. Disponível em: https://testingcompany.com.br/blog/conheca-o-cypress-e-suas-vantagens-para-automacao-de-testes. Acesso em: 23 de julho de 2023.

HANASHIRO, Akira. **O que se pode fazer com JavaScript hoje em dia?**, 2029. Disponível em: https://www.treinaweb.com.br/blog/o-que-se-pode-fazer-com-javascript-hoje-em-dia. Acesso em: 23 de julho de 2023.

HNZ. **Tipos de testes utilizados em DevOps.** HNZ, 2022. Disponível em: https://hnz.com.br/tipos-de-testes-utilizados-em-devops/. Acesso em: 26 de julho de 2023.

CYPRESS. Cypress. **JavaScript Web Testing and Component Testing Framework | Cypress**. Disponível em: https://www.cypress.io/. Acesso em: 23 de julho de 2023.

ROBOT Framework Foundation. **Robot Framework**. Disponível em: https://robotframework.org/. Acesso em: 20 de julho de 2023.

DEVMEDIA. **Java: história e principais conceitos.** DEVMEDIA, 2012. Disponível em: https://www.devmedia.com.br/java-historia-e-principais-conceitos/25178. Acesso em: 28 de julho de 2023.

BSTQB. **Certified Tester Syllabus. Foundation Level. CTFL 3.1.** Disponível em: https://bcr.bstqb.org.br/docs/syllabus_ctfl_3.1br.pdf. Acesso em: 23 julho 2023.