Estudo dos benefícios da inclusão de IA em testes de software com *Selenium WebDriver* e *Applitools Eyes*

Gabriel C. da Cunha[1], Kassya C. R. de Andrade[1]

[1]Faculdade de Computação e Informática - Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, CEP 01302-000, São Paulo – SP, Brasil

gabrielcoelho.cunha1@mackenzista.com.br,
kassyachristina.andrade@mackenzie.br

Abstract. *This paper is a study about the benefits of visual testing with Applitools Eyes integrated with Selenium WebDriver's functional testing. The objective is to explore and demonstrate some benefits of using tools that may better the software development process and, in this way, encourage research about artificial intelligence applied to software testing.*

Keywords: *Software Testing Tools; Artificial Intelligence; Selenium; Applitools Eyes.*

Resumo. *Este artigo é um estudo dos benefícios de testes visuais com Applitools Eyes integrados a testes funcionais da Selenium WebDriver. O objetivo é explorar e demonstrar alguns benefícios da utilização de ferramentas que possam contribuir para a melhoria do processo de desenvolvimento de software e, assim, incentivar a pesquisa de inteligência artificial aplicada a testes de software.*

Palavras-chave: *Ferramentas de Teste de Software; Inteligência Artificial; Selenium; Applitools Eyes.*

1 Introdução

Teste de Software tem alta relevância no processo de desenvolvimento de software, já que cerca de 50% do tempo gasto (MYERS, BADGETT e SANDLER, 2011) e cerca de 23% do orçamento de tecnologia da informação de empresas é alocado para testes (CAPGEMINI, 2020).

Tendo em vista a importância de testes durante o processo de desenvolvimento de software, ao aplicar Inteligência Artificial (IA) na testagem de software, é possível a obtenção dos seguintes benefícios indicados por Battina (2019):

- Precisão melhorada;
- Testagem além das restrições manuais;
- Facilitadores para desenvolvedores e testadores;
- Aumento do número total de testes realizados;
- Redução de tempo e dinheiro consumidos, o que acelera a disponibilização do software ao mercado.

Um dos benefícios citados, aumento do número total de testes realizados, é ilustrado na figura 1, que mostra quanto as técnicas de teste com IA podem melhor acompanhar a complexidade de um software quando comparado às técnicas tradicionais de testagem automática.

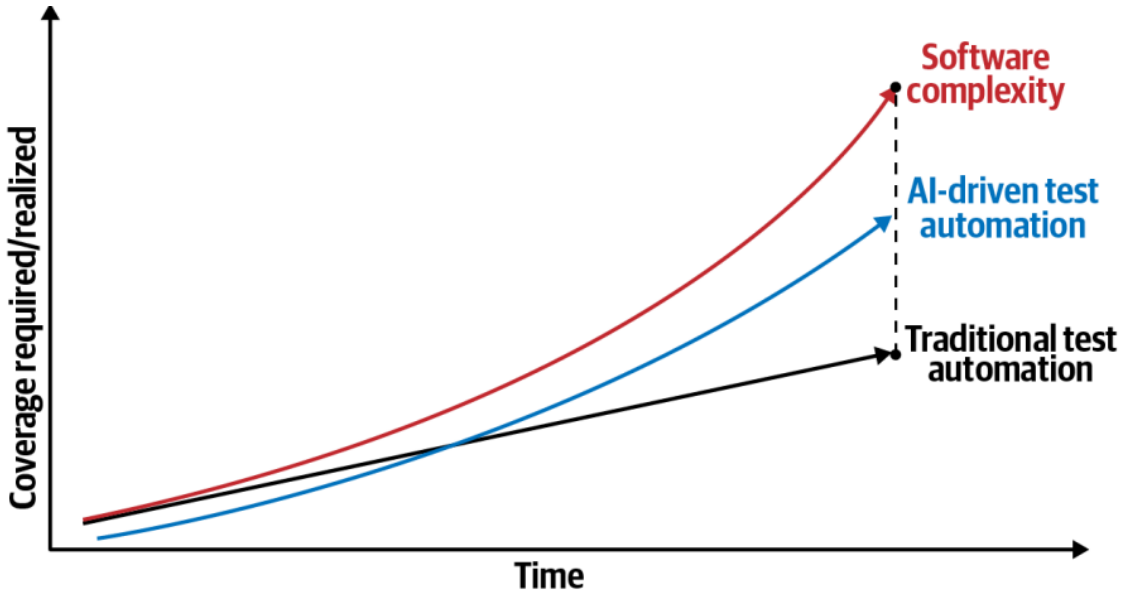

Figura 1. Aumentando cobertura de testes com automação baseada em IA

Fonte: King (2021, n.p.)

Diante da importância de testes e dos benefícios que IA aplicada a testes de software podem trazer, como disseminar e incentivar o uso de ferramentas de automação de testes de software que utilizam IA?

De forma a responder essa pergunta, o objetivo geral deste trabalho é prover uma demonstração de benefícios ao incluir uma ferramenta que utiliza IA para realizar testes visuais em uma aplicação web. Para atender tal objetivo geral, os objetivos específicos foram:

- Estudar Teste de Software, Inteligência Artificial e ferramentas de automação de teste de software;
- Indicar possíveis benefícios da utilização de ferramentas de automação de testes de software que usam IA;
- Disseminar ferramentas de automação de testes de software que utilizam IA;
- Demonstrar benefícios de incluir IA para realizar verificações adicionais;
- Realizar experimentos com ferramentas de automação de teste de software.

2 Referencial Teórico

Para entender Teste de Software, primeiro é necessário entender o que é qualidade de software, que de acordo com Pressman e Maxim (2021) é a gestão de qualidade com o objetivo de gerar um produto útil. É importante para o desenvolvimento de software pois busca prevenção de defeitos, avaliação das implementações e redução de custos provenientes de falhas.

Um importante método para atingir qualidade é através da utilização de Teste de Software, a forma de mostrar que o programa funciona de acordo com seus requisitos e identificar defeitos antes da entrega ao cliente (SOMMERVILLE, 2019).

Testes fazem parte do processo de verificação e validação (V&V) (SOMMERVILLE, 2019). Tal processo analisa o produto que está sendo construído, verificando se é a forma correta de construí-lo e validando se é o produto certo (BOEHM, 1979).

Segundo Sommerville (2019), testes estão presentes em todos os modelos genéricos de processos de desenvolvimento de software.

Primeiro, o modelo em cascata, que os utiliza durante as fases de implementação e teste de unidade e integração e teste de sistema.

As fases de teste do modelo de desenvolvimento incremental são intercaladas com o desenvolvimento e são chamadas de atividades de validação. Esse modelo é utilizado como base pela metodologia ágil.

Por fim, as abordagens que utilizam o modelo de integração e configuração, utilizam testes nas fases de configuração da aplicação para uma aplicação já existente e para casos em que alguns componentes estão disponíveis, aplica testes nas fases de adaptação dos componentes, desenvolvimento de componentes e integração do sistema.

Algumas das ferramentas de automação de teste de software aplicam os conceitos de Inteligência Artificial através de algoritmos de Aprendizagem de Máquina e Aprendizagem Profunda.

Segundo Norvig e Russel (2013), IA é o campo de estudo de agentes que tem capacidade de funcionar de forma autônoma, aprender com mudanças e criar e perseguir metas para alcançar o melhor resultado esperado.

Uma forma de aplicar o conceito de IA é Aprendizagem de Máquina (AM), que de acordo com Murphy (2012) é um conjunto de algoritmos utilizados para detectar padrões de forma automatizada e utilizar tais padrões para tentativas de predizer dados futuros e auxiliar em decisões.

Tais algoritmos computacionais aprendem ao realizar uma determinada tarefa diversas vezes, seus resultados melhoram quanto mais realizam o mesmo tipo de tarefa (MITCHELL, 1997).

De acordo com Bengio, Goodfellow e Courville (2015), uma categorização comum desses algoritmos é aprendizagem não supervisionada e aprendizagem supervisionada, que indicam o tipo de experiência utilizada durante o aprendizado.

Algoritmos de aprendizagem não supervisionada exploram um conjunto de dados contendo diversas características para aprender propriedades úteis da estrutura do conjunto de dados. Essa abordagem é geralmente utilizada para resolver tarefas do tipo estimação de densidade com o papel de ajudar em outras tarefas (BENGIO; GOODFELLOW; COURVILLE, 2015).

Já algoritmos de aprendizagem supervisionada servem para explorar um conjunto de dados contendo diversas características com cada exemplo do conjunto associado a um rótulo. As tarefas geralmente relacionadas com essa categorização envolvem regressão, classificação e problemas de saída estruturada (BENGIO; GOODFELLOW; COURVILLE, 2015).

Uma abordagem muito poderosa para aplicar AM é Aprendizagem Profunda (AP),

> [...] que permite computadores aprender por experiências e entender o mundo como uma hierarquia de conceitos, com cada conceito definido a partir de conceitos simples. Pelo ganho de conhecimento através de experiências, essa abordagem evita a necessidade de operadores humanos de formalmente especificar todo o conhecimento que o computador necessita. A hierarquia de conceitos permite que o computador aprenda conceitos complicados construindo-os usando conceitos mais simples. Se desenharmos um gráfico mostrando como esses conceitos são construídos um a partir dos outros, o gráfico é profundo, com diversas camadas (BENGIO; GOODFELLOW; COURVILLE, 2015, tradução nossa).

A figura 2 ilustra o relacionamento entre Inteligência Artificial, Aprendizagem de Máquina e Aprendizagem Profunda.

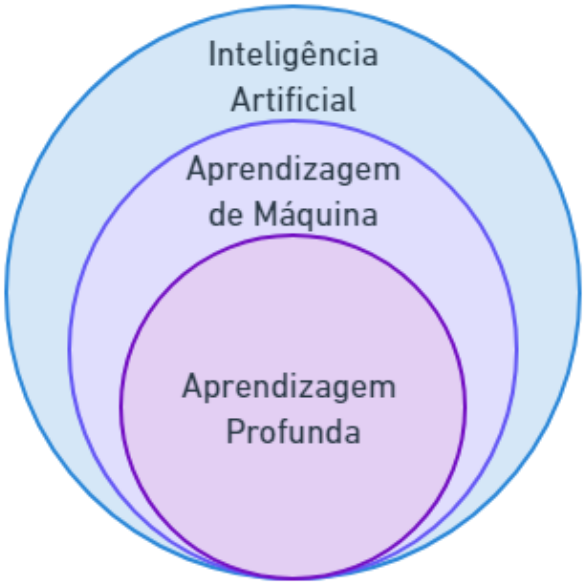

Figura 2. Diagrama de Venn mostrando como Aprendizagem Profunda é um tipo de Aprendizagem de Máquina que por fim é um conjunto de algoritmos de Inteligência Artificial

Fonte: autoria própria baseado em Bengio, Goodfellow e Courville (2015, p. 9)

Existem diversas ferramentas com o papel de auxiliar a testagem de software através da automação de tal processo. As duas ferramentas com maior presença no mercado, de acordo com *Enlyft* (c2023), uma empresa que agrega dados em tempo real de empresas que usam diversas soluções tecnológicas, são:

a) *Selenium*

É uma ferramenta de automação de navegadores de internet. É usada principalmente para automação de aplicações web com propósitos de testagem (SOFTWARE FREENDOM CONSERVANCY, c2023). Uma das formas de utilizar tal ferramenta é através do *Selenium WebDriver*, que se refere a dois assuntos, o primeiro sendo o código que possibilita as várias linguagens de programação utilizar *Selenium* e o

segundo é a implementação do código de controle dos navegadores (SOFTWARE FREENDOM CONSERVANCY, c2023).

b) *Apache JMeter*

É um software *open source* feito 100% em Java para testar capacidade de carga do comportamento funcional das aplicações, e medir performance. Foi originalmente designado para testar aplicações *web*, mas foi complementado com outras funções de teste (APACHE SOFTWARE FOUNDATION, c2022).

Além de *Selenium* e *Apache JMeter*, existem ferramentas que utilizam inteligência artificial para melhorar o processo de teste de software. A seguir são apresentadas algumas:

a) *Katalon*

É uma plataforma de gerenciamento de qualidade que utiliza IA para sugerir métodos alternativos de localização de elementos quando a opção padrão falha; acelerar verificações manuais de interfaces de usuário e comparações de pixel propensas a erros com métodos baseados no conteúdo ou na estrutura; e torna análise de causa raiz mais eficiente encontrando falhas parecidas dos resultados das execuções anteriores (KATALON, c2022).

Dos três planos oferecidos, *Free*, *Premium* e *Ultimate*, apenas o último permite utilização das funcionalidades da IA.

Os benefícios prometidos incluem:

- Visibilidade: provê meios para entender a cobertura e efetividade de qualidade de software sobre toda a organização;
- Escalabilidade: possibilita o ajuste as necessidades do negócio;
- Velocidade: torna mais rápida a escalabilidade, reduz o tempo para os produtos serem liberados no mercado e aumenta a qualidade de cada versão disponibilizada;
- Inovação: utilização de IA e AM para ajudar a testar mais rapidamente e de forma mais eficiente;
- Acessibilidade: planos gratuitos e planos pagos flexíveis para atender necessidades do negócio.

b) *Launchable*

É uma plataforma de desenvolvimento de software focada em integração contínua que utiliza AM para predizer qual a chance de falha de cada teste baseado em execuções anteriores e mudanças no código (LAUNCHABLE, s.d.).

A plataforma disponibiliza duas modalidades de planos, *Moon* e *Mars*, mas apenas o *Mars* habilita a utilização da funcionalidade com IA.

Os benefícios que a *Launchable* promete incluem:

- Redução do tempo de execução dos testes através da utilização da seleção preditiva de testes que identifica e executa apenas os 20% de testes que mais impactam o processo de desenvolvimento;
- Eficiência por toda a organização, visível do painel de controle da ferramenta;
- Melhor visibilidade para projetos e times;

- Visualização de dados de tendência de como o projeto tem evoluído.

c) *Applitools*

É uma plataforma de automação de testes que utiliza uma rede de algoritmos de computação visual, composta de diversas abordagens incluindo AP, para analisar páginas de aplicações e identificar diferenças (APPLITOOLS, c2022).

A plataforma disponibiliza quatro modalidades de planos, *Free*, *Starter*, *Enterprise* e *Ultrafast*, sendo que todos permitem utilização da funcionalidade com IA. A principal diferença é a quantidade de testes com a IA por mês, sendo que o único gratuito (*Free*) está limitado a cem testes de página por mês.

Os benefícios que a *Applitools* promete incluem:

- Validar páginas inteiras com uma única captura de tela;
- Gerenciar conteúdo dinâmico ou que mudam de forma contínua;
- Testar todos os navegadores e dispositivos em segundos;
- Automatizar revisões e manutenções;
- Corrigir erros mais rapidamente com análise da causa raiz;
- Testar para acessibilidade visual.

3 Metodologia

A metodologia foi dividida em pesquisa bibliográfica e desenvolvimento. A seguir estão seus detalhamentos.

3.1 Pesquisa bibliográfica

Essa etapa foi realizada com intuito de construir um referencial teórico com os principais temas do trabalho, assim expandindo sobre o que é o processo de Teste de Software e entendendo onde está posicionado dentro do processo de desenvolvimento de um software. Além disso se definiu o que é Inteligência Artificial e como Aprendizado de Máquina e Aprendizagem Profunda fazem parte de tal tema. Explicou-se a função de ferramentas de automação de testes de software, exemplificando com algumas amplamente utilizadas e outras que utilizam Inteligência Artificial.

Para a seção de Teste de Software, os materiais utilizados foram livros sobre Engenharia de Software de autores renomados como Pressman, Maxim e Sommerville. Ao falar de Inteligência Artificial, os materiais utilizados foram livros dos autores Norvig, Russel, Murphy, Bengio, Goodfellow e Courville. A escolha de livros foi realizada pelo fato de serem obras publicadas formalmente e expandirem sobre partes necessárias para o compreendimento deste artigo. Já sobre ferramentas de automação de testes de software, os materiais escolhidos foram websites, um de uma empresa que agrega dados em tempo real de empresas que usam diversas soluções tecnológicas e o restante foram das ferramentas e de suas documentações. O primeiro foi escolhido por mostrar quais ferramentas são as mais utilizadas no mercado e os últimos foram escolhidos pois detalham muito bem o que são e como utilizá-las.

3.2 Desenvolvimento

Após a exploração dos principais temas, buscou-se as ferramentas que oferecem a disponibilidade das funcionalidades que usam IA em planos gratuitos e não temporários, o que resultou na escolha da *Applitools Eyes*.

Com a *Applitools Eyes* escolhida, sua documentação foi explorada e encontrou-se testes visuais integrados com testes funcionais da *Selenium WebDriver*. *Selenium* foi escolhida pois concentra uma grande parcela (27.35%) do mercado de ferramentas de testagem de software de acordo com *Enlyft* (c2023).

Para aprender a utilizar *Applitools Eyes* em conjunto com a *Selenium WebDriver* seguiu-se um tutorial, disponibilizado pela própria *Applitools*, que ensina a realizar testes *Selenium WebDriver* em conjunto com o *framework Jest*, que utiliza *JavaScript* para testagem (APPLITOOLS, c2022).

Durante o período de aprendizado foi notada a existência de quatro algoritmos, chamados de *match levels*, disponíveis para realizar verificações visuais, sendo:

- ***Exact***: compara as imagens *pixel* a *pixel*, sendo sensível a anomalias de renderização não visíveis aos olhos humanos;
- **S*trict***: detecta mudanças em texto, fontes, cor, gráficos, e posição dos elementos;
- **I*gnore colors***: similar ao *strict*, mas ignora mudanças de cores;
- ***Layout***: identifica os elementos da página, como texto, imagens, botões, e colunas e verifica se a posição relativa dos elementos é consistente.

Após o aprendizado das ferramentas de automação de testes, foi escolhida a aplicação web, *TMDB Web App*, desenvolvida anteriormente durante um curso *online*.

TMDB Web App é uma aplicação web que utiliza a *API* de *The Movie Database* (*TMDB*), um banco de dados de filmes e séries alimentado por voluntários (THE MOVIE DATABASE, s.d.), para gerenciar listas de filmes. As tecnologias utilizadas para implementar tal aplicação foram *HTML*, *CSS*, *TypeScript*, *ReactJS*, *Axios*, *Styled Componentes* e *Redux Toolkit* (CUNHA, 2022).

Sobre suas funcionalidades, o quadro 1 as apresenta, detalhando o estado inicial necessário para utilizá-las e os possíveis estados finais que podem ocorrer.

Quadro 1. Funcionalidades da aplicação web

Funcionalidade	Estado inicial	Possíveis estados finais
Entrar na conta *TMDB*	Usuário na página *Login*.	A. Se usuário insere credenciais válidas é redirecionado para página *Search*; B. Se usuário insere credenciais inválidas, um alerta aparece avisando das credenciais inválidas e permanece na página *Login*.
Sair da conta *TMDB*	Usuário nas páginas *Search* ou *Lists*.	C. Usuário redirecionado para página *Login*.
Criar lista	Usuário na página *Lists*.	D. Se usuário insere nome e descrição, uma tabela aparece com a lista criada com nome e descrição; E. Se usuário insere só nome, uma tabela aparece com a lista criada com nome; F. Se usuário não insere nome, um alerta aparece avisando que o campo *Name* é obrigatório; G. Se usuário insere nome de uma lista já criada, um alerta aparece avisando que já existe uma lista com o mesmo nome.
Remover lista	Usuário na página *Lists* com uma lista criada.	H. Lista é removida da tabela de listas e se for a única lista, a tabela também desaparece.
Pesquisar por filme	Usuário na página *Search*.	I. Se o filme estiver presente na base de dados, mostra tabela com resultados; J. Se o filme não estiver na base de dados, a página não sofre alteração.
Adicionar filme em uma lista criada	Usuário na página *Search* com uma lista criada.	K. Filme é adicionado na lista selecionada, habilitando o botão *Remove* e desabilitando o botão *Add* da página *Search* para o filme e aparecendo na lista de filmes da lista na página *Lists*.
Remover filme de uma lista criada	Usuário nas páginas *Search* ou *Lists* com uma lista criada e com um filme adicionado nela.	L. Filme é removido da lista selecionada, habilitando o botão *Add* e desabilitando o botão *Remove* da página *Search* e desaparecendo da lista de filmes na lista na página *Lists*.

Fonte: autoria própria

Após escolher a aplicação web, os testes a serem realizados foram planejados com o intuito de cobrir a maioria dos estados finais das funcionalidades da aplicação, resultando no quadro 2. As duas últimas colunas deste quadro fazem referência a primeira e última colunas, respectivamente, do quadro 1.

Quadro 2. Detalhamento dos cenários de teste

#	**Cenário**	**Funcionalidade**	**Estado final esperado**
1	Com credenciais válidas consegue acessar a aplicação	Entrar na conta *TMDB*	A
2	Sem credenciais válidas não consegue acessar a aplicação	Entrar na conta *TMDB*	B
3	Consegue criar uma lista com nome e descrição	Criar lista	D
4	Consegue criar uma lista somente com nome	Criar lista	E
5	Não consegue criar uma lista sem nome	Criar lista	F
6	Não consegue criar uma lista com mesmo nome de lista existente	Criar lista	G
7	Busca por um filme presente na base de dados	Pesquisar por filmes	I
8	Busca por um filme não existente na base de dados	Pesquisar por filmes	J
9	Adiciona um filme numa lista	Adicionar filme em uma lista criada	K
10	Na página *Search*, remove um filme de uma lista	Remover filme de uma lista criada	L
11	Na página *Lists*, remove um filme de uma lista	Remover filme de uma lista criada	L

Fonte: autoria própria

Para todos os testes foram implementados passos anteriores e posteriores com o objetivo de manter o estado inicial e final da aplicação sempre iguais e facilitar a implementação dos testes. Com esse objetivo em mente, as funcionalidades **sair da conta *TMDB*** e **remover lista** não têm cenários porque foram utilizadas como passos posteriores de grande parte dos testes.

Com os testes planejados, vivenciou-se a limitação do plano gratuito da *Applitools Eyes* de somente permitir cem capturas de tela durante o período de um mês.

Como cada teste planejado necessita de duas capturas, para registar os estados inicial e final da página de cada cenário, o autor deste artigo decidiu escolher uma amostra dos testes para permitir mais enganos durante a implementação. Os testes escolhidos, detalhados no quadro 2, foram 1, 3, 7, 9, 10 e 11, já que demonstram o estado de sucesso de quase todas as funcionalidades.

4 Resultados

A seguir estão detalhados o ambiente computacional utilizado (quadro 3), sua preparação e a realização dos experimentos.

Quadro 3. Ambiente computacional

Sistema Operacional	*Linux Ubuntu* (22.04.2 *LTS*)
Processador	*Intel Core i5-8250U*
RAM	4 GB
Navegador	*Google Chrome* (113.0.5672.63)

Fonte: autoria própria

Durante a preparação do ambiente para a realização dos experimentos, foram implementados testes de duas formas. Para a primeira forma, somente com código *Selenium*, foram implementados todos os testes do quadro 2. Para a segunda forma, incluindo *Applitools Eyes* nos testes com *Selenium* de forma a capturar os estados inicial

e final de cada teste, foram implementados apenas os testes escolhidos (1, 3, 7, 9, 10 e 11).

Parte das implementações envolvem algumas configurações, essas implementadas utilizando como base o código do tutorial seguido para aprender a utilizar *Applitools Eyes* com *Selenium*, que resultaram nas configurações finais representadas pelas figuras 3 e 4.

```
let driver;
const getDriver = () => driver; // Permite utilizacao de mesma instancia em outros arquivos

// Configura passos anteriores e posteriores dos testes Selenium
function configureSeleniumSetupAndTeardown() {
    beforeEach(async () => { // Antes de cada teste
        driver = await configureChromeWebDriver();
        await getPage();
    });
    afterEach(async () => { // Apos cada teste
        await driver.quit();// Fecha a instancia do Chrome Web Driver
    });
}

// Cria e configura instancia de Chrome Web Driver.
async function configureChromeWebDriver() {
    const driver = new Builder()                    // Cria instancia
        .withCapabilities({
            browserName: 'chrome',                  // define navegador
            'goog:chromeOptions': { args: [], },    // habilita configuracoes adicionais
        })
        .build();
    // Configura o tempo de espera para cada passo da instancia em 1 segundo
    await driver.manage().setTimeouts({ implicit: 1000 });
    // Configura a janela da instancia
    await driver.manage().window().setRect({
        x: 0, y: 0,                                 // topo da pagina
        width: SCREEN.WIDTH, height: SCREEN.HEIGHT, // resolucao de 1024×768
    });
    return driver;
}

// Acessa localhost na porta 3000 na pagina 'page'
async function getPage(page = '') {
    await getDriver().get(`http://localhost:3000/${page}`);
}
```

Figura 3. Configurações *Selenium*

Fonte: autoria própria baseado em *Applitools* (c2022)

```
let eyesConfig;
let eyes;
// Permite utilizacao de mesma instancia em outros arquivos
const getEyesConfig = () => eyesConfig;
const getEyes = () => eyes;

// Configura passos anteriores e posteriores dos testes com Applitools Eyes
function configureApplitoolsEyesSetupAndTeardown() {
    let runner;
    beforeAll(async () => {              // Antes de todos os testes
        eyesConfig = new Configuration();// Cria instancia de configuracao
        // Carrega a chave da API das variaveis de ambiente
        eyesConfig.setApiKey(process.env.APPLITOOLS_API_KEY);
        runner = new ClassicRunner();    // Cria instancia para gerenciar os testes localmente
    });
    beforeEach(async () => {             // Antes de cada teste
        eyes = new Eyes(runner);         // Cria instancia do Eyes, que realiza as comparacoes
        eyes.setConfiguration(eyesConfig);// Define que a instancia Eyes utilize a configuracao atual
        await eyes.open(                                  // Abre o Eyes para aceitar testes visuais
            getDriver(),                                  // Objeto driver a ser observado
            APPLITOOLS_APP,                               // Nome da aplicacao a ser testada
            expect.getState().currentTestName,            // Nome do teste atual
            new RectangleSize(SCREEN.WIDTH, SCREEN.HEIGHT)// Resolucao a ser testada de 1024×768
        );
    });
    afterEach(async () => {// Apos cada teste
        // Fecha instancia do Eyes para que os testes fiquem disponiveis no painel da Applitools
        await eyes.closeAsync();
    });
    afterAll(async () => {// Apos todos os testes
        // Recupera todos os resultados dos testes
        await runner.getAllTestResults();
    });
}

/*
Para cada pagina da aplicacao a ser testada,
    utilizei a seguinte funcao na secao de passos anteriores a todos os testes da pagina
*/
// Define a qual conjunto de testes o teste atual pertence de acordo com a pagina 'page'
async function setBatchInfoForPage(page) {
    await getEyesConfig().setBatch(new BatchInfo(`${APPLITOOLS_APP} @ ${page}`));
}
```

Figura 4. Configurações *Applitools Eyes*

Fonte: autoria própria baseado em *Applitools* (c2022)

Além das configurações demonstradas nas figuras 3 e 4, foram necessárias configurações para que o *framework* de testes *Jest* não rejeitasse os testes por tempo de execução. Assim o tempo definido para os testes *Selenium WebDriver* foi de 10 segundos e para os testes com *Applitools Eyes* foi de 120 segundos.

A aplicação em conjunto com as configurações e os testes estão disponíveis no repositório *GitHub* (CUNHA, 2023).

Após a preparação do ambiente, os testes foram executados, primeiro com somente *Selenium WebDriver* e logo em seguida os mesmos testes com a inclusão da *Applitools Eyes*.

Após a execução dos testes, registrou-se os dados obtidos e observações sobre algumas características escolhidas para analisar a implementação e execução de testes com *Selenium WebDriver* e com *Selenium WebDriver* em conjunto com *Applitools Eyes*. Tais características, detalhadas no quadro 4, foram escolhidas ao considerar alguns estudos sobre ferramentas de automação de teste de software (SINGH e TARIKA, 2014) (GAMIDO e GAMIDO, 2019).

Quadro 4. Características, das implementações e execuções dos testes de software com as ferramentas, a serem analisadas

Característica	Detalhamento
Requisitos para implementação e execução	Quais os conhecimentos técnicos necessários para implementar e executar os testes.
Função	O que a execução dos testes com as ferramentas válida.
Tempos de execução	Quanto tempo cada teste levou (em segundos) e a porcentagem do aumento do tempo de execução ao incluir *Applitools Eyes*.
Limites de execução	Quais os limites ao executar os testes de acordo com limitações de cada ferramenta.

Fonte: autoria própria

a) Requisitos para implementação e execução

Para utilizar *Selenium WebDriver*, seguindo as tecnologias utilizadas neste experimento, é necessário **familiaridade com escrita e depuração de código *JavaScript*, construção de conjuntos de testes com *Jest*** e ***SDK Selenium WebDriver***.

Já para incluir *Applitools Eyes* nos testes com *Selenium WebDriver*, além dos conhecimentos anteriores é necessário conhecer o ***SDK Applitools Eyes***.

Considerando tais conhecimentos, para uma pessoa que não esteja inserida ou iniciando na área de desenvolvimento e qualidade de software, a utilização da *Applitools Eyes* acrescenta dificuldade. Mas em contraponto, quem já tem contato com os conhecimentos técnicos anteriores ao *SDK Applitools Eyes* terá um caminho facilitado para conseguir desfrutar dos benefícios que a *Applitools Eyes* propõe.

b) Função

A função dos testes, implementados com somente Selenium WebDriver é de verificar se é possível interagir da forma esperada com os elementos da página da aplicação. Já ao testar a aplicação incluindo *Applitools Eyes* a função expande a verificação da *Selenium WebDriver* também testando mudanças na aparência da interface da aplicação.

c) Tempos de execução

A seguir estão os dados obtidos dos tempos de execução dos testes.

Tabela 1. Tempos de execução (em segundos)

#	***Selenium WebDriver***	***Selenium WebDriver* com *Applitools Eyes***	**Aumento percentual da inclusão da *Applitools Eyes***
1	4,06	36,87	808,13%
3	4,10	41,65	915,85%
7	4,31	37,26	764,50%
9	7,27	46,45	538,93%
10	7,13	49,08	588,36%
11	7,64	44,73	485,47%
Total	**34,51**	**256,04**	**641,93%**
Média	**5,75**	**42,67**	**641,93%**

Fonte: autoria própria

Como é possível observar na tabela 1, como a inclusão da Applitools Eyes adiciona a função de realizar verificações visuais nos testes, seus tempos de execução aumentam consideravelmente, em média 641,93% por teste.

Ao levar em consideração o ambiente computacional descrito no quadro 3, a aplicação web testada (*TMDB Web App*), as tecnologias utilizadas (*JavaScript* e *Jest*) e o plano gratuito da *Applitools*, estes resultados podem mudar.

d) Limites de execução

Sobre limites de execução, ao utilizar a ferramenta *Selenium WebDriver* não há limite de execução, já que se trata de uma ferramenta que tem apenas execução local e por se tratar de uma ferramenta *open source*.

Applitools Eyes necessita de uma chave de autenticação atrelada à limites de execução, isso se dá ao fato de que é uma ferramenta comercial que oferece planos para utilizar suas funcionalidades. Considerando o plano utilizado no experimento, o gratuito, o limite de cem capturas de tela por mês é bem restrito.

Para configurar a inclusão da *Applitools Eyes* e realizar uma execução completa dos testes planejados, 80% da cota mensal foi atingida.

5 Conclusão

Além das características analisadas durante a realização dos experimentos, pôde-se perceber alguns dos benefícios prometidos pela *Applitools* (c2022).

Ao utilizar *Applitools Eyes* é possível **validar páginas inteiras com uma única captura de tela**, isso significa visualizar o estado da aplicação para cada captura de tela realizada nos testes automatizados, verificando se as funcionalidades e os testes com *Selenium* estão realizando o esperado. Além disso é possível visualizar prévias dos outros algoritmos disponíveis para verificar o estado visual da aplicação. Estes sendo *exact*, *strict*, *ignore colors* e *layout*. A figura 5 exemplifica tal benefício. Os destaques em rosa são as diferenças identificadas entre as duas versões da página web, que foram causados pelo tamanho do conteúdo do seletor ser maior na versão mais recente (captura de tela da aplicação à direita), assim movendo os botões para a direita. O algoritmo utilizado foi o *strict*, então considerou o desalinhamento dos elementos. Ao aplicar a prévia do algoritmo *layout*, como exemplificado na figura 6, as diferenças não são mais encontradas já que a interface possui os mesmos elementos.

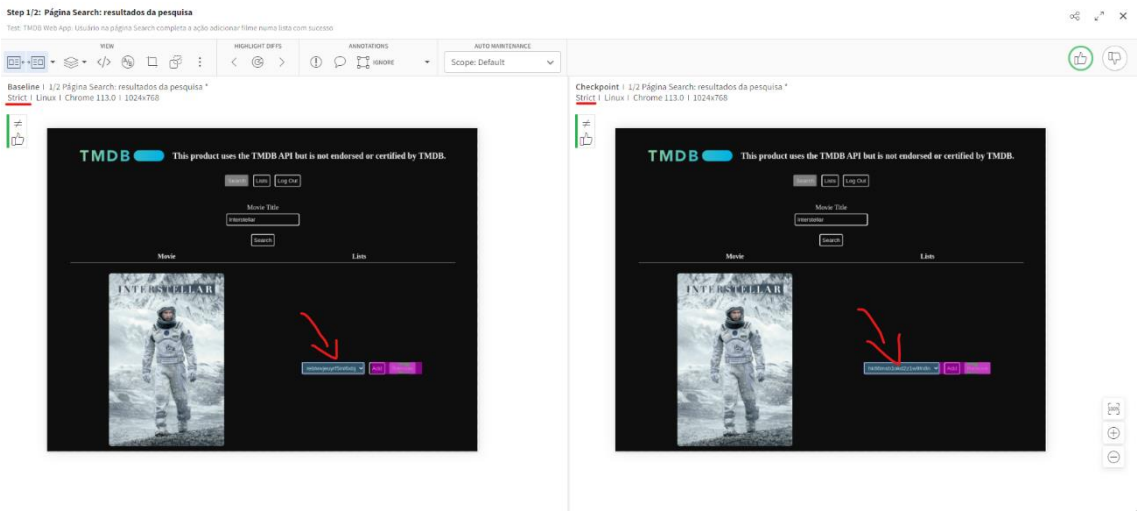

Figura 5. Exemplo da validação de páginas inteiras com uma única captura de tela

Fonte: autoria própria

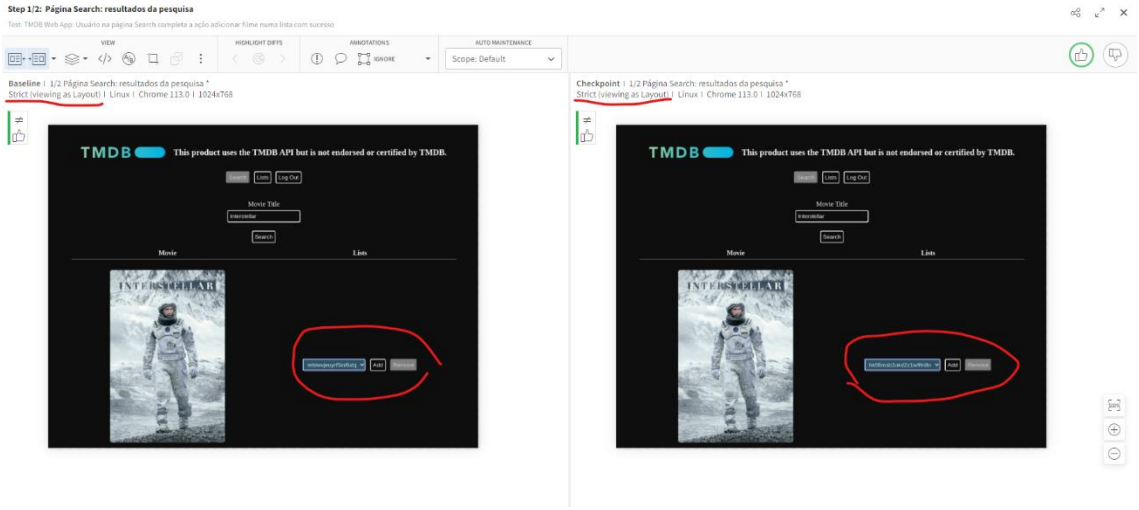

Figura 6. Exemplo da prévia de algoritmos diferentes para validação de páginas inteiras com uma única captura de tela

Fonte: autoria própria

Os mesmos algoritmos servem para **gerenciar conteúdo dinâmico ou que mudam de forma contínua**, principalmente utilizando o algoritmo *layout*, que verifica a estrutura da página. Com as anotações, tanto por código ou na visualização dos resultados dos testes, também é possível utilizar múltiplos algoritmos para seções diferentes da página. A figura 7 exemplifica esse benefício, mostrando como é possível adicionar algoritmos diferentes para realizar a verificação de seções da página. A seção em cinza, cobrindo o seletor da aplicação web mostra uma anotação para ignorar o elemento e a região com pontos que permitem o redimensionamento demonstram a adição do algoritmo *layout* a uma verificação com o algoritmo *strict*.

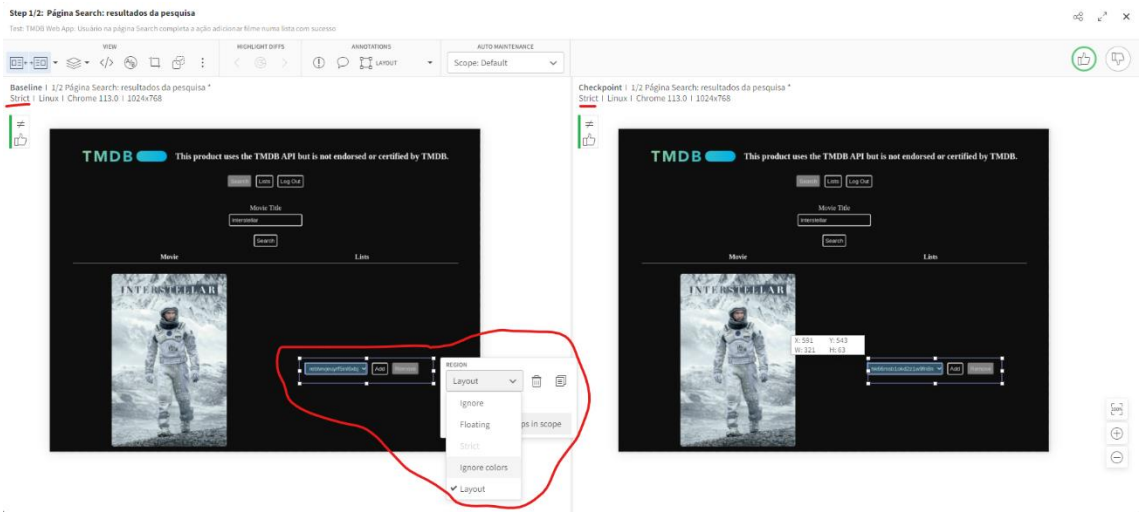

Figura 7. Exemplo do gerenciamento de conteúdo dinâmico ou que mudam de forma contínua

Fonte: autoria própria

Caso seja necessário, é possível utilizar as anotações mencionadas para **automatizar revisões e manutenções dos testes**. Isso quer dizer que existe a possibilidade de se fazer um teste via código e necessitando mudanças, realizar alterações durante a visualização dos resultados e permitir que tais modificações sejam consideradas para todos os testes futuros. Ao clicar no botão *Apply to other steps in scope* da figura 8, é possível o aproveitamento deste benefício.

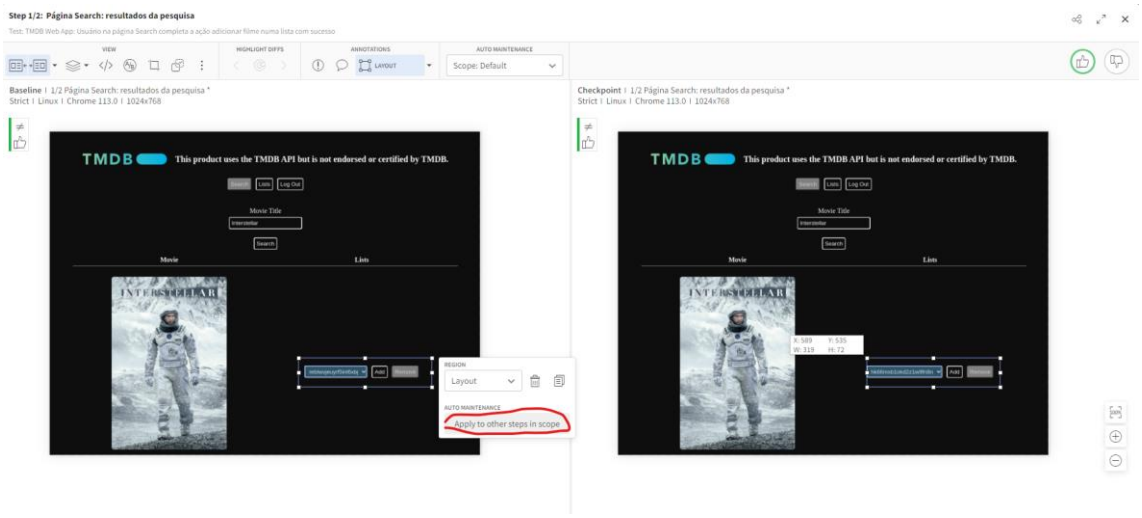

Figura 8. Exemplo da automatização de revisões e manutenções dos testes

Fonte: autoria própria

Ao visualizar o resultado de um teste é possível **corrigir erros mais rapidamente com análise da causa raiz**, que mostra quais as razões dos testes terem falhado de acordo com o algoritmo utilizado para verificar o estado visual da página. A figura 9 exemplifica este benefício, ao focar na seção *Root cause analysis*, é possível a visualização da causa dos destaques em rosa.

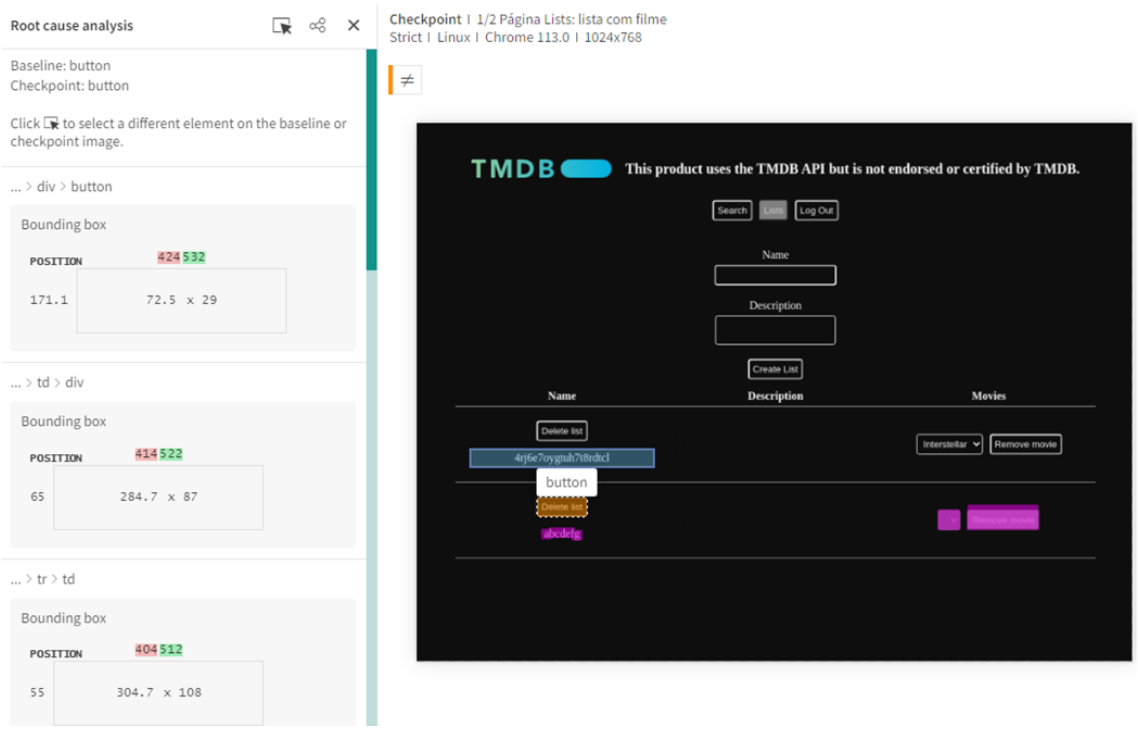

Figura 9. Exemplo da correção de erros mais rapidamente com análise da causa raiz

Fonte: autoria própria

Durante o desenvolvimento da pesquisa, encontrou-se alguns obstáculos. O primeiro foi o fato de que os planos de utilização da primeira escolha de ferramenta com IA a ser testada, *Launchable*, foram alterados e impossibilitaram o uso da funcionalidade com IA. Tal obstáculo foi superado pela escolha e utilização de outra ferramenta previamente pesquisada, *Applitools Eyes*, em conjunto com *Selenium WebDriver*.

O segundo obstáculo foi que surgiram dificuldades na alteração da primeira escolha de aplicação web a ser testada pelas ferramentas, *Sharetribe Flex* (SHARETRIBE, c2023). Esse obstáculo foi superado pela escolha da aplicação web *TMDB Web App*.

O terceiro e último obstáculo foi o limite de utilização da ferramenta *Applitools Eyes*, que restringiu quais testes realizou-se sobre a aplicação web escolhida.

As contribuições esperadas ao produzir este trabalhão são incentivar a pesquisa de Inteligência Artificial aplicada a Teste de Software e informar sobre ferramentas que possam melhorar o processo de desenvolvimento de software.

Sugere-se como trabalhos futuros, realizar todos os testes propostos e experimentar com todos os algoritmos disponíveis para realizar as verificações visuais. Ainda, ao considerar a limitação do plano gratuito da *Applitools Eyes*, realizar testes com os diferentes planos pagos. Por fim, ao considerar todas as ferramentas de automação de testes de software que utilizam inteligência artificial, realizar testes com seus diferentes planos.

Referências

APACHE SOFTWARE FOUNDATION. *Apache JMeter*. c2022. Disponivel em: https://jmeter.apache.org/. Acesso em: 15 maio 2023.

APPLITOOLS. *Applitools*. c2022a. Disponivel em: https://applitools.com/. Acesso em: 7 nov. 2022.

BATTINA, Dhaya Sindhu. Artificial Intelligence in Software Test Automation: A Systematic Literature Review. *International Journal of Emerging Technologies and Innovative Research*, v. 6, n. 12, p. 1329-1332, dez. 2019. Disponivel em: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4004324. Acesso em: 21 nov. 2022.

BENGIO, Yoshua; GOODFELLOW, Ian; COURVILLE, Aaron. *Deep Learning*. Cambridge: MIT press, 2015.

BOEHM, Barry W. Software Engineering as it is. *Proceedings of the 4th International Conference on Software Engineering*. Munich: IEEE Press. 1979. p. 11-21. Disponivel em: https://www.inf.ed.ac.uk/teaching/courses/seoc1/2005_2006/resources/bullet03.pdf. Acesso em: 4 nov. 2022.

CAPGEMINI. *World Quality Report*, 2020. Disponível em: https://www.capgemini.com/insights/research-library/world-quality-report-2019-20/. Acesso em: 14 jun. 2023.

CUNHA, Gabriel C. da. *GitHub*: repositório TMDB Web App. 2022. Disponivel em: https://github.com/gccunha015-dio/tmdb_web_app. Acesso em: 18 maio 2023.

CUNHA, Gabriel C. da. *GitHub*: repositório TMDB Web App com testes. 2023. Disponivel em: https://github.com/gccunha015-upm/tmdb_web_app. Acesso em: 18 maio 2023.

ENLYFT. *Enlyft*: Software Testing Tools products. c2023. Disponivel em: https://enlyft.com/tech/software-testing-tools. Acesso em: 15 maio 2023.

GAMIDO, Heidilyn V.; GAMIDO, Marlon V. Comparative review of the features of automated software testing tools. *International Journal of Electrical and Computer Engineering*, v. 9, n. 5, p. 4473-4478, out. 2019. Disponivel em: http://download.garuda.kemdikbud.go.id/article.php?article=1304601&val=146&title=Comparative%20Review%20of%20the%20Features%20of%20Automated%20Software%20Testing%20Tools. Acesso em: 23 maio 2023.

KATALON. *Katalon*. c2022. Disponivel em: https://katalon.com/. Acesso em: 19 nov. 2022.

KING, Tariq. *AI-Driven Testing*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2021. Disponivel em: https://www.oreilly.com/library/view/ai-driven-testing/9781098105983/. Acesso em: 2 nov. 2022.

LAUNCHABLE. *Launchable*. s.d. Disponivel em: https://www.launchableinc.com/. Acesso em: 4 nov. 2022.

MITCHELL, Tom M. *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

MURPHY, Kevin P. *Machine Learning*: A Probabilistic Perspective. Cambridge, MA, USA: MIT Press, 2012.

MYERS, Glenford J.; BADGETT, Tom; SANDLER, Corey. *The Art of Software Testing*. 3ª ed. New Jersey: Wiley & Sons, 2011.

NORVIG, Peter; RUSSEL, Stuart. *Inteligência Artificial*. 3ª ed. Rio de Janeiro: Elsevier Editora, 2013.

PRESSMAN, Roger S.; MAXIM, Bruce R. *Engenharia de Software*. 9ª ed. [S.l.]: Grupo A, 2021.

SHARETRIBE. *Sharetribe*. c2023. Disponivel em: https://www.sharetribe.com/. Acesso em: 7 abr. 2023.

SINGH, Inderjeet; TARIKA, Bindia. Comparative analysis of open source automated software testing tools: Selenium, sikuli and watir. *International Journal of Information & Computation Technology*, v. 4, n. 15, p. 1507-1518, 2014. Disponivel em: http://www.ripublication.com/irph/ijict_spl/ijictv4n15spl_04.pdf. Acesso em: 9 maio 2023.

SOFTWARE FREENDOM CONSERVANCY. *Selenium*. c2023. Disponivel em: https://www.selenium.dev/. Acesso em: 22 fev. 2023.

SOMMERVILLE, Ian. *Engenharia de Software*. 10ª ed. São Paulo: Pearson Education do Brasil, 2019.

THE MOVIE DATABASE. *The Movie Database (TMDB)*. s.d. Disponivel em: https://www.themoviedb.org/. Acesso em: 30 abr. 2023.