# UMA ANÁLISE COMPARATIVA ENTRE TESTES DE SOFTWARE MANUAL E AUTOMATIZADO

*Ian Victor Gomes Aguiar*
i.victorgomes21@gmail.com
*Stéfano Schwenck Borges Vale Vita*
stefano.vita@uniube.br

**RESUMO**

*Softwares* se tornaram indispensáveis ao desenvolvimento da humanidade nesse sentido, seu perfeito funcionamento é essencial para a realização das atividades ao qual foi proposto. Durante o seu desenvolvimento é importante que a cada etapa o sistema seja submetido a testes a fim de assegurar mais produtividade, uma vez que os erros podem ser detectados e corrigidos a cada etapa, pois testar o *software* apenas quando esse já estiver concluído consiste em uma abordagem ineficaz. Assim, essa pesquisa terá como objetivos criar casos de testes para demonstrar qual o passo a passo a ser realizados nos testes; Realizar o teste manual conforme os casos de teste; Criar o script automatizado para minimizar problemas da abordagem manual; Avaliar qual teste obteve um melhor desempenho.

**Palavras-chave:** *Softwares*. Testes. Manual. Automatizado. Desempenho.

# SOFTWARE TESTING: A COMPARATIVE ANALYSIS BETWEN MANUAL TEST AND AUTOMATED TEST

**ABSTRACT**

Software has become indispensable to the development of humanity in this sense, its perfect functioning is essential for carrying out the activities for which it was proposed. During its development, it is important that the system is submitted to tests at each stage in order to ensure more productivity, since errors can be detected and corrected at each stage, since testing the software only when it is already completed consists of a ineffective approach. Thus, this research will have as objectives to create test cases to demonstrate the step by step to be carried out in the tests; Perform manual testing according to test cases; Create the

automated script to minimize problems with the manual approach; Evaluate which test performed better.

## 1 INTRODUÇÃO

Atualmente a busca por *softwares*, aplicativos e sites tiveram uma enorme alavancada devido a população utilizá-los para quase todas as finalidades de seu dia a dia. Devido a esta grande dependência tecnológica dos últimos anos, está cada vez mais complexo o desenvolvimento correto de um *software* que tenha qualidade e pronto para ser utilizado.

O *software* é o elemento central que realiza estruturas complexas e flexíveis, trazem utilidades, funções e valor ao sistema em si. O seu desenvolvimento faz parte do ciclo de vida da engenharia de *software*, que informa que o seu progresso é feito através de um projeto e todo seu esquema representa a execução de um processo. Uma parte importante da engenharia de *software* é a qualidade. Geralmente a qualidade de um produto é diretamente proporcional ao nível do processo utilizado em sua produção. (PAULA FILHO, 2000).

Criar um *software* requer um grande raciocínio lógico para instruir a máquina a executar processos de cálculos numéricos, sendo esta uma tarefa que exige muita atenção, foco e concentração ao implementar várias regras que possuem grande complexidade. Devido a tais complicações, o código criado pode conter alguma falha de lógica que passe despercebida. (SPIRLANDELI, 2019; ROLAND, 2019).

Para assegurar tais características é feita utilização de testes manuais dos sistemas. Antes de disponibilizar o *software* em produção, é realizado umas avaliações para comprovar a sua qualidade, geralmente é realizado por equipes especializadas em testes, desenvolvedores e usuários do sistema através dos testes manuais, tendo o objetivo de verificar possíveis falhas em relação aos requisitos iniciais (CRISPIN; HOUSE, 2003).

Atualmente, as empresas que trabalham com desenvolvimento precisam validar seus produtos de forma mais rápida devido ao (*Continuous Delivery* – CD), no qual o objetivo é a realizar qualquer tipo de melhorias no ambiente de produção de forma segura, rápida e sustentável. Assim, devido as alterações que ocorrem no *software* ao decorrer do seu ciclo de vida é necessário testar o que já estava desenvolvido e as novas funcionalidades para que o usuário tenha a melhor experiência, deste modo, as empresas querendo mais agilidade e eficácia

em seus testes, estão adotando a automação dos testes manuais para reduzir o custo e o tempo de execução comparado a forma manual. (TRINDADE; 2021).

Levando em consideração a importância dos testes, será realizado uma comparação entre os testes manuais e os automatizados, com o intuito de demonstrar que a automação possui grandes benefícios.

O objetivo de automatizar os testes é a melhoria da qualidade do *software* como um todo, pois podem prover uma melhor cobertura, busca validar se defeitos antigos que já foram resolvidos não reapareceram no *software* e ajuda a executar muitos casos de testes de forma consistente e repetidamente em várias versões do ambiente ou sistema.

A automação ira fazer com que a máquina execute os testes, o mesmo poderá ser executado repetidas vezes, terá um tempo menor de execução, melhora o custo final, maior eficiência e um feedback mais rápido. Para conseguir que a máquina execute esses testes será usado ferramentas *Open Source* com linguagem de programação Java e será usado o *Selenium WebDriver* para realizar a comunicação com o *browser*.

Os objetivos específicos irão demonstrar como é a fase de planejamento para ser realizado um teste em um *software*. Os passos são:

- Criar cenários para demonstrar qual o passo a passo a ser realizados nos testes.
- Realizar o teste manual conforme os cenários.
- Criar o *script* automatizado para minimizar problemas da abordagem manual.
- Avaliar qual teste obteve um melhor desempenho.

# 2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção é apresentada a revisão da literatura, dando exemplos de métodos para testes de software até chegar nos testes automatizados.

## 2.1 Testes de *software*

Os testes de *softwares* consistem em uma sequência de atividades empregadas metodicamente, antes do desenvolvimento dos softwares ser finalizado, a fim de analisar se o produto está funcionando conforme a expectativa, desde sua interface e layout até o código (PRESSMAN, 2011).

De acordo com Sommerville (2011) o objetivo do teste é demostrar se o programa está executando o que foi proposto a fazer e conhecer os possíveis defeitos ainda antes do uso. Durante os testes de *softwares* são usados dados fictícios. Os resultados dos testes são analisados em busca de erros ou informações dos atributos disfuncionais do programa.

Trindade (2021) explica que no decorrer da história do desenvolvimento de *softwares*, foram definidos os princípios de teste que fornecem quesitos de orientações gerais e comuns a todo o tipo de teste, conforme relatado no Quadro 1.

Quadro 1 - Princípios na realização de testes

| | |
|---|---|
| **Os testes mostram a presença de defeitos e não a sua ausência** | Os testes podem mostrar a presença de defeitos, mas não provam a sua inexistência. Os testes reduzem a probabilidade de os defeitos não detectados permanecerem no *software*. |
| **Testes exaustivos são impossíveis** | Testar tudo e considerar todas as combinações possíveis não é viável. Em vez de testes exaustivos, a análise de risco e as prioridades devem ser utilizadas para focar os esforços de teste. |
| **Testar cedo salva tempo e dinheiro** | Para tentar evitar que os defeitos sejam mantidos até uma fase de desenvolvimento mais avançada, as atividades de teste devem ser iniciadas o mais cedo possível no ciclo de desenvolvimento do *software* ou sistema e devem estar focadas nos objetivos definidos. |
| **Agrupamento de defeitos** | O esforço de testes deve ser proporcional à densidade de defeitos esperada por módulo. Normalmente, um número reduzido de módulos apresenta a maioria dos defeitos deletados, ou é responsável pela maioria das falhas. Portanto, deve-se perceber quais os módulos do sistema que são mais críticos ou propícios a defeitos ou falhas para se investir com mais rigor na qualidade desses módulos. |
| **Paradoxo do pesticida** | A repetição exaustiva dos mesmos casos de teste pode levar à não deteção de novos defeitos. Para superar este "paradoxo do pesticida", os casos de teste devem ser regularmente analisados, revistos e mudados para executar diferentes partes do *software* ou sistema. |
| **Os testes dependem do seu contexto** | Os testes são efetuados de forma diferente em diferentes contextos. Por exemplo, o *software* crítico em segurança (*safety-critical software*) é testado de forma diferente de um website de comércio eletrônico, pois exige diferentes tipos de cuidados. |
| **Falácia da ausência de erros** | Detectar e corrigir defeitos por si só não chega se o sistema construído não satisfizer as necessidades e expectativas dos seus utilizadores. |

Fonte: Adaptado de Trindade, 2021, p. 8-9

O teste de *software* se tornou essencial no processo de desenvolvimento, integrado a esse processo (BASTOS, 2007). Koscianski e Soares (2007) complementam que quando a etapa de teste é planejada de modo sistemático e rigoroso, pode ser empregue para estimular a confiança e qualidade do projeto em desenvolvimento.

## 2.2 Níveis de testes de *softwares*

Testar o *software* somente quando esse já estiver concluído, consiste em uma abordagem ineficaz. Uma metodologia de teste deve seguir uma abordagem incremental, iniciando com o teste das unidades, seguido de um teste de integração e findando, com os testes do sistema final (PRESSMAN, 2006). Seguindo essa abordagem, Lima (2014) classifica em sua pesquisa os testes em quatro etapas: unidade, integração, sistema e aceitação.

Conforme Bernardo (2011) os testes de unidade analisam a menor unidade do projeto de *software*, tendo como objetivo verificar a lógica interna de processamento e estruturas de dados de um componente. Geralmente, é relacionado a funções de linguagens proceduráis e metodologias de linguagens orientadas a objetos. Os testes nessa etapa devem ser realizados durante a implementação, pelos próprios desenvolvedores, com intuito de assegurar que o código foi criado de acordo com o que foi definido ante de sua integração com as demais unidades.

Os testes de integração visam verificar a interface entre os componentes e acontece em paralelo à atividade de integração do sistema. Esses testes são realizados em sistemas completos ou subsistemas, formados por componentes integrados, em que unidades de *hardware* e *software* devem ser testados com conjunto a fim de avaliar a integração entre eles, permitindo conhecer os problemas nas interfaces das unidades (SOFTEX, 2011).

Os testes de sistema objetivam a verificação do comportamento do sistema final, e precisam ser executados em um ambiente correspondendo ao ambiente em que o sistema será implementado. Deve ser verificado o sistema por completo, focando em analisar as conformidades com os requisitos (LIMA, 2014).

O teste de aceitação é empregue para garantir ao usuário que o sistema irá atender as suas expectativas. Para realizar esse teste é preciso estabelecer critérios de aceitação a partir dos requisitos do *software*, definindo como o teste será conduzido com base nos critérios estabelecidos (LIMA, 2014).

## 2.3 Tipos de testes de *softwares*

Durante o desenvolvimento de um *software*, até sua conclusão, diversos testes são realizados a fim de assegurar que o produto está em conformidade com os requisitos pré-estabelecidos (LUÍS, 2018). Braga (2019) destaca em sua pesquisa os seguintes testes: teste funcional, teste de carga, teste de instalação, teste de segurança, teste de volume, teste de stress,

teste de regressão, teste de manutenção, teste de usabilidade e *smoke test*, conforme apresentados no Quadro 2.

Quadro 2 – Tipos de testes

| | |
|---|---|
| **Teste funcional** | Este teste é caracterizado por possuir o objetivo de validar se as funções disponíveis pelo *software* estão de acordo com o especificado pelo cliente na documentação do produto, como por exemplo requisitos de negócios e requisitos técnicos. |
| **Teste de carga** | É o teste que tem como objetivo simular a carga em condições de normais de uso. |
| **Teste de instalação** | Tipo de teste que verifica se o *software* é instalado corretamente em diferentes hardwares sob condições adversas como pouca memória, internet ruim e interrupções durante a instalação. |
| **Teste de segurança** | São testes focados em verificar se o sistema funciona de maneira segura e se os dados estão protegidos |
| **Teste de volume** | É o teste que visa simular o comportamento do sistema com em períodos de maior tráfico de dados esperado, e com isso monitorar os impactos na rede. |
| **Teste de *stress*** | Teste que tem como objetivo verificar o comportamento do *software* quando se atinge o máximo (ou além disso) do tráfico de dados suportados. |
| **Teste de regressão** | Um dos testes considerados mais importantes, é o reteste de funcionalidades anteriormente desenvolvidas para verificar se alguma modificação ou nova funcionalidade impactou nas mesmas. |
| **Teste de manutenção** | É o teste que verifica se a mudança de ambiente, exemplo ambiente de pré-produção para ambiente de produção, gerou impacto no funcionamento do sistema |
| **Teste de usabilidade** | É o teste que foca na utilização da aplicação pelo usuário, se o layout está correto, se a interface está de acordo com as necessidades do cliente, entre outros. |
| ***Smoke test*** | É o teste que verifica os pontos mais críticos do *software*, é executado após a aplicação estar pronta no ciclo de desenvolvimento, para ser encaminhada para o ciclo de testes. |

Fonte: Adaptado de Braga, 2019, p. 26

## 2.4 Automação de testes

Lima (2014) explica que os testes podem ser realizados por meios manuais ou automáticos. Conforme Trindade (2021) os testes manuais consistem em reproduzir as atividades por parte de uma pessoa definida, por sua vez, os testes automatizados consistem na automação do processo manual, com auxílio de *software* que realiza a interação humana com o sistema, sendo então criados *scripts* que simulam os testes manuais e que são executados de maneira automática.

> Teste automatizado é realizado por um programa que testa outro programa para localizar eventuais erros. Ao criar um programa somente para testes, é possível executar os testes constantemente, várias vezes, de forma rápida e com baixo custo. Uma das formas de se desenvolver *softwares* dirigido por testes é com a implementação de trechos de código, chamados de Testes Unitários ou Testes Funcionais, nos quais se testa a menor parte passível de verificação de um programa. Por exemplo se a programação for orientada a objetos testam-se os métodos criados nas classes de objetos. Com a utilização de testes unitários as mensagens de retorno (*feedback*) indicam se algo parou de funcionar a partir de alterações realizadas numa funcionalidade, gerando algum erro ocorrido em algum trecho do sistema (SPIRLANDELI, 2019, p. 3).

Durante os testes automatizados, os testes e seus canários são codificados em um programa e devem ser realizados todas as vezes que houver mudança na aplicação. O emprego de desse tipo de teste vem crescendo no mercado, pois permite um gasto de tempo menor na atividade de teste, enquanto o software realiza o teste, os profissionais podem trabalhar em outras atividades (SOMMERVILLE, 2011).

A automação permite a criação de testes manuais mais complexos e completos a serem realizados pela equipe em qualquer instante do projeto, transmitindo mais confiança para a equipe que pode realizar as alterações no código do *software* (BERNARDO e KON, 2008).

# 3 DESENVOLVIMENTO

Nesta seção serão apresentadas as ferramentas utilizadas no desenvolvimento do projeto e os resultados obtidos nas execuções manuais e automatizadas, seguindo os *steps* do cenário proposto.

## 3.1 Ferramentas utilizadas

Para realizar a automação de teste do projeto proposto, foram utilizadas ferramentas *open source*, técnicas de desenvolvimento e *ide* para compilação dos códigos, onde a seguir terá uma breve explicação.

A linguagem utilizada é o *java*, uma linguagem de programação orientada a objetos amplamente utilizada e uma plataforma *software* executada em bilhões de dispositivos, incluindo *notebooks*, dispositivos móveis, consoles de jogos, dispositivos médicos e muitos outros. As regras e a sintaxe do *java* são baseadas nas linguagens C e C++.

O banco de dados utilizado é o *MySQL Workbench* que é uma ferramenta de design de banco de dados visual que integra desenvolvimento *SQL*, administração, design de banco de dados, criação e manutenção em um único ambiente de desenvolvimento integrado para o sistema de banco de dados *MySQL*.

Para simular a interação do usuário com o *browser*, foi utilizado o *selenium*, que é um conjunto de ferramentas de código aberto multiplataforma, usado para testar aplicações web pelo browser de forma automatizada. Ele executa testes de funcionalidades da aplicação web e testes de compatibilidade entre browser e plataformas diferentes. O *Selenium* suporta diversas linguagens de programação, como por exemplo C#, Java e *Python*, e vários navegadores *web* como o *Chrome* e o *Firefox*.

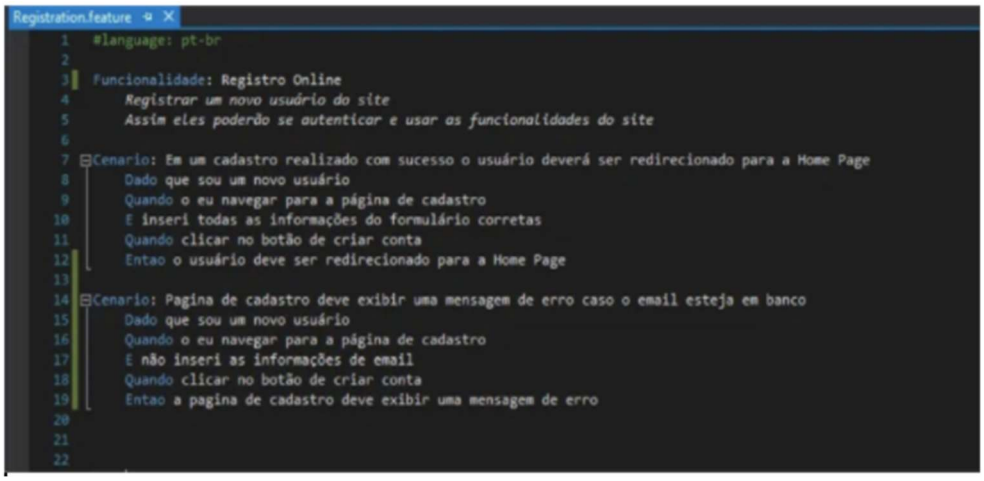
Entre as técnicas de desenvolvimento, foi utilizado o BDD *Behavior Driven Development, que é* uma técnica utilizada para integrar regras de negócios e linguagem de programação, caracterizando-se por um vocabulário bem específico e pequeno, que minimiza as dificuldades de comunicação e possibilita todos os membros do time utilizarem uma mesma linguagem para realizar o trabalho.

Para criar os cenários de teste utilizando BDD, foi utilizado as palavras-chaves: Dado (*Given*), Quando (*When*), Então (*Then*).

- Dado: Define as pré-condições verdadeiras para executar o seu teste
- Quando: Define a ação que será executada
- Então: Seguindo a ação descrita no QUANDO define o resultado esperado para o seu teste
- E: adiciona uma sentença positiva no Dado, no Quando ou no Então.

Para conseguir realizar a utilização do BDD, utilizamos o *cucumber*. O c*ucumber* é uma ferramenta usada para executar testes de aceitação automatizados que foram criados em um formato BDD. Ele que torna possível especificar o comportamento esperado do software durante a codificação e posteriormente usar esses cenários de testes para averiguar o comportamento obtido. A Figura 1 que está abaixo, ilustra como é a utilização do BDD com o *cucumber*.

Figura 1 – Exemplo de escrita BDD com *cucumber*




A *ide* escolhida para rodar os *scripts* foi o *IntelliJ IDEA Community Edition*, que é uma edição gratuita e de código aberto do aplicativo *IDEA Ultimate Edition* de nível profissional da *JetBrains*. Ele fornece todas as ferramentas necessárias para que desenvolver seu código.

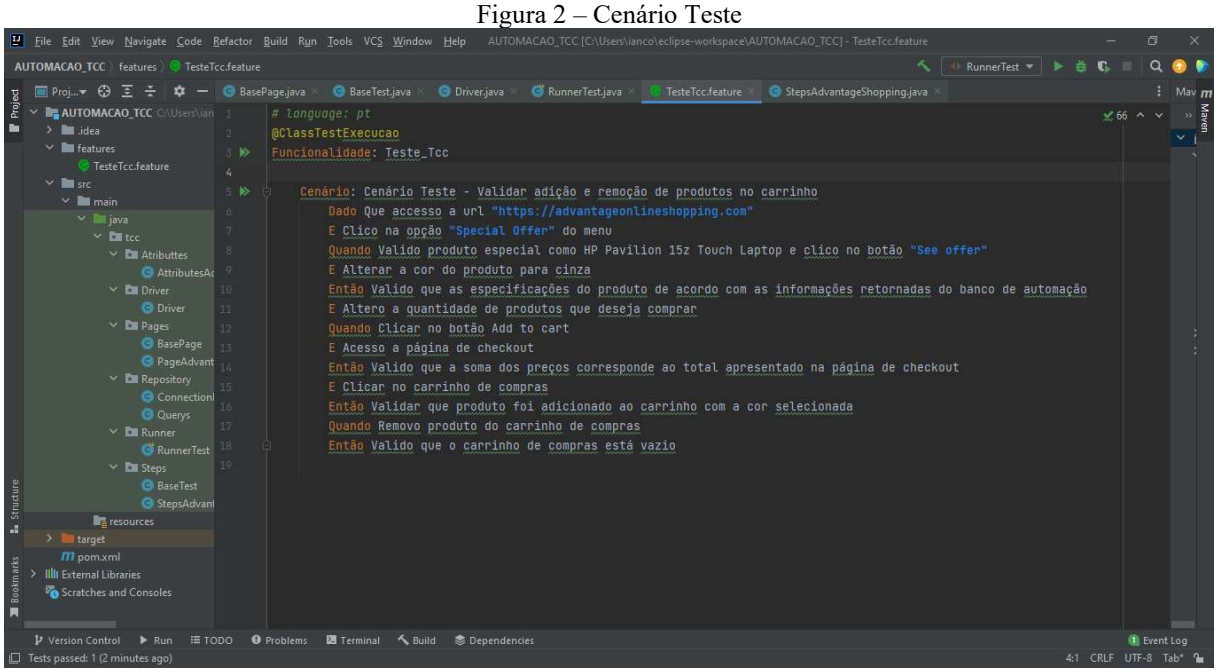
## 4 RESULTADOS

Nesta seção serão apresentados os resultados obtidos na realização dos testes manuais e automatizados.

### 4.1 Testes manuais

Para a execução dos testes manuais é necessário seguir a estória e montar os cenários para seguir o passo a passo. Para essa montagem normalmente é utilizado um sistema para gerenciamento de testes como *ALM*, *TestLink*, *qTest*, entre outros. Para os cenários hipotéticos que vamos testar, utilizaremos escrita feita em BDD com *cucumber*. Conforme mencionado no desenvolvimento, cada DADO, QUANDO, ENTÃO e E representa o passo a passo que será realizado, conforme demonstrado na Figura 2.

Figura 2 – Cenário Teste




Fonte: Autor

O teste manual consiste no analista ir manualmente no *browser* e seguir o passo a passo definido nos cenários para verificar o comportamento do *software* conforme os requisitos que constam na estória.

### *4.1.1 Resultados manuais*

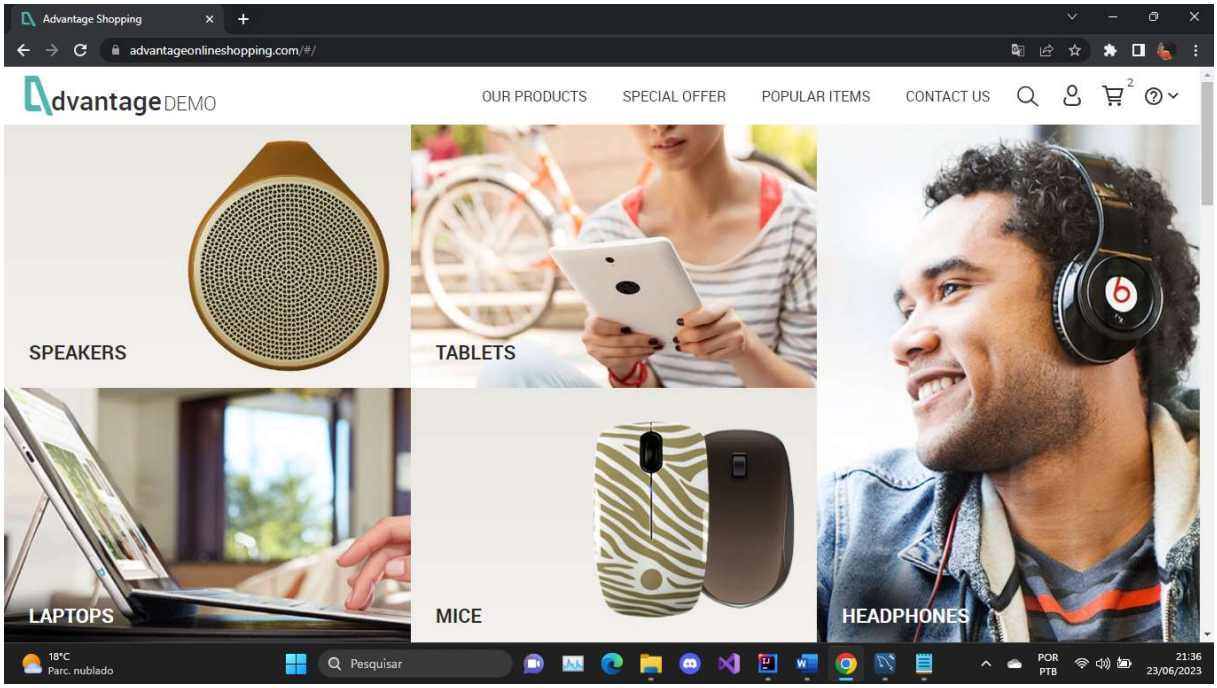
Os resultados dos testes manuais são adicionados nas ferramentas de gerenciamentos de testes mencionadas acima como um print em cada *step* ou um arquivo contendo todos as capturas de imagem que demonstram o passo a passo, onde é necessário a evidenciação dos *steps* para validar que o *software* obteve o comportamento esperado ou mostrar que há um *bug* que precisa ser corrigido.

Como a maioria desses *softwares* são pagos, será apresentado com evidências de imagem a execução manual do cenário hipotético proposto. Abaixo é ilustrado os resultados obtidos para o cenário executado manualmente.

Cenário Teste - Validar adição e remoção de produtos no carrinho

Abaixo, a Figura 3 identifica a abertura do navegador no site que será realizado o teste.

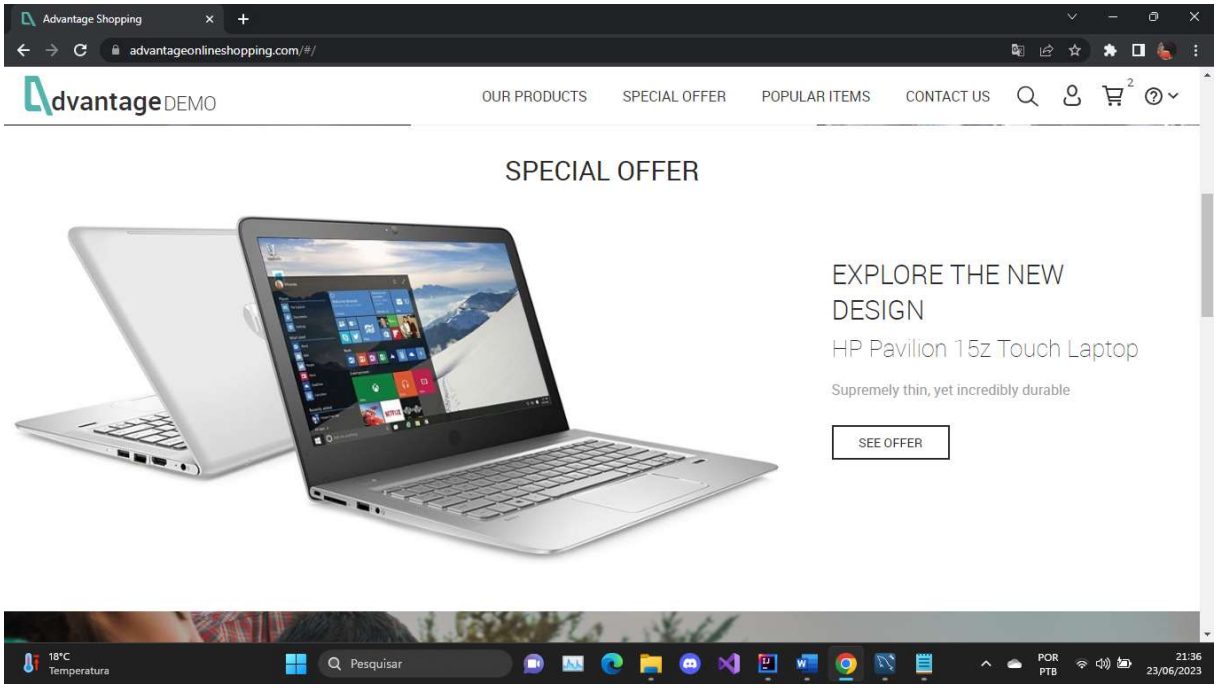
Figura 3 - *Step* 1: Que acesso a *url* https://*advantageonlineshopping*.com




Fonte: Autor

Ao clicar na opção *special offer* disponível no menu superior do site, ele é encaminhado para o produto que está em oferta, conforme demonstrado na Figura 4.

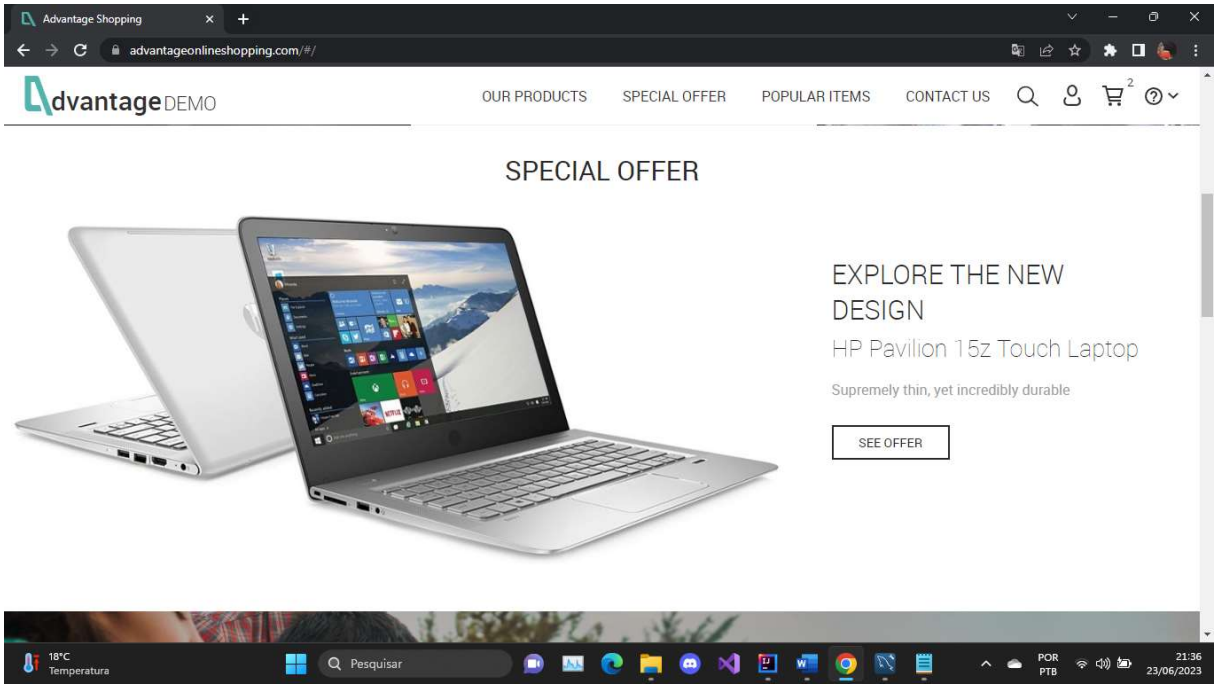
Figura 4 - *Step* 2: Clico na opção *Special Offer*




Fonte: Autor

Ao ser direcionado para a oferta especial, é validado que o produto em oferta é o HP *Pavillon* 15z *Touch Laptop*, conforme demonstrado na Figura 5.

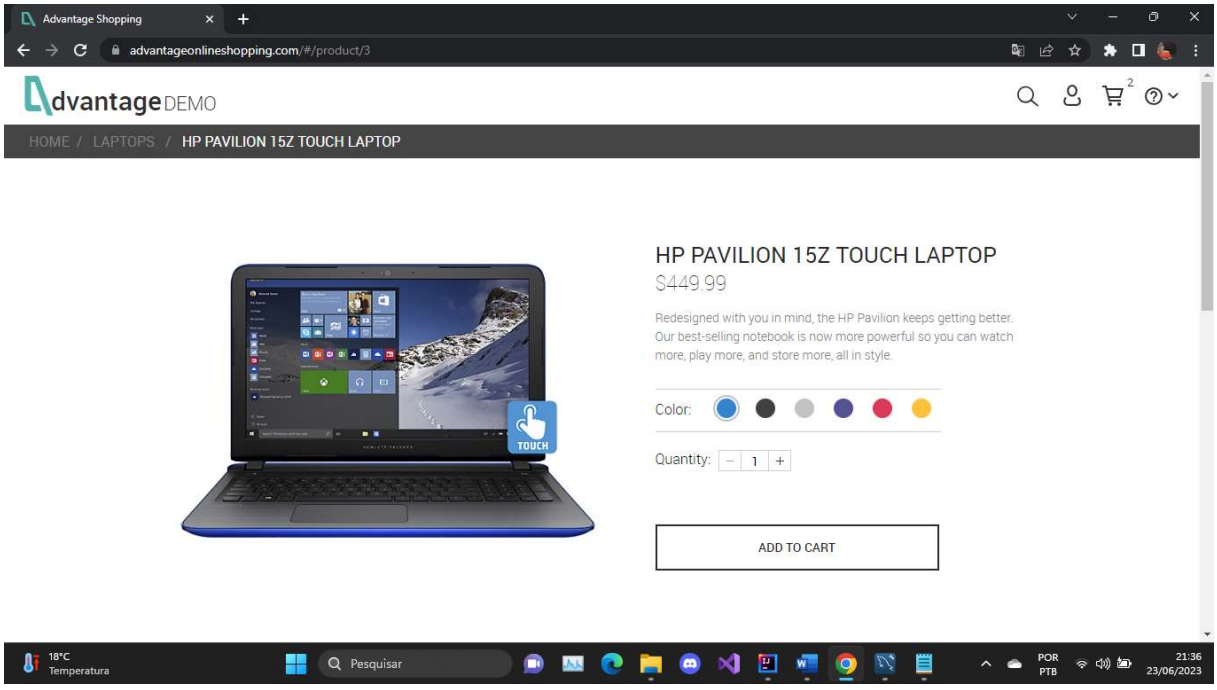
Figura 5 - *Step* 3: Valida produto *special offer* como HP *Pavilion* 15z *Touch Laptop*




Fonte: Autor

A Figura 6 demonstra que ao clicar no botão *see offer* no produto especial, é feito o direcionamento para a tela de atributos do produto.

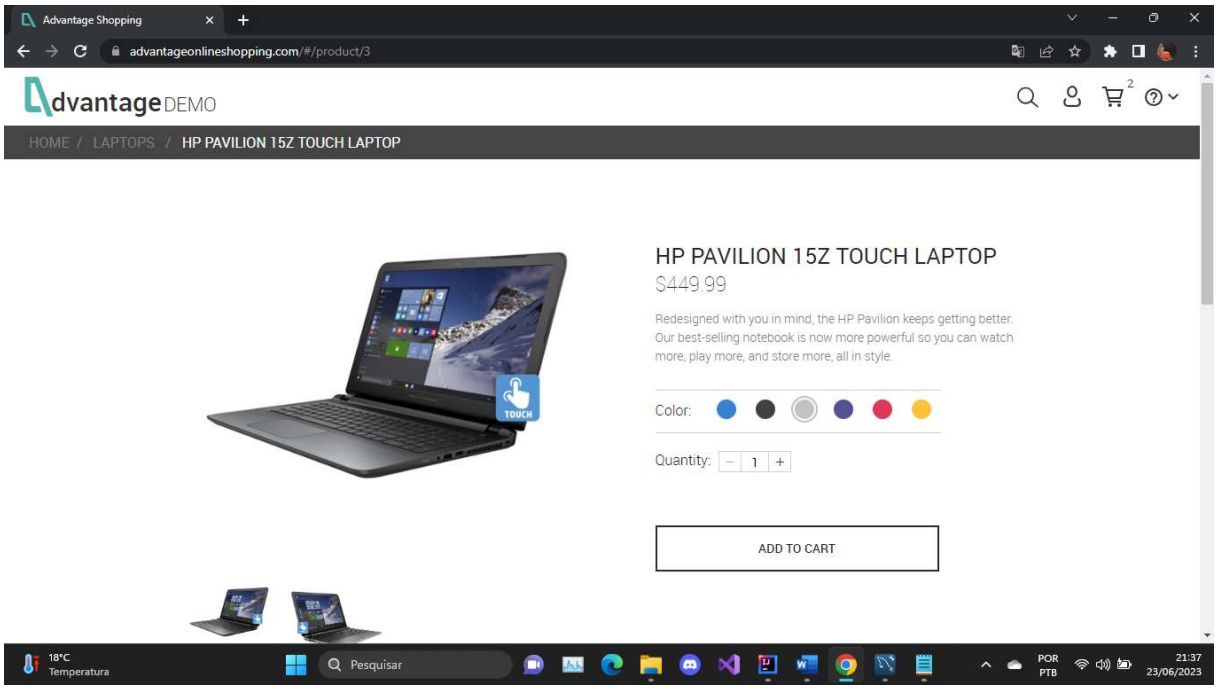
Figura 6 - *Step* 3: Clico no botão *See Offer*




Fonte: Autor

A escolha da cor cinza que é diferente da pré-selecionada é ilustrada pela Figura 7.

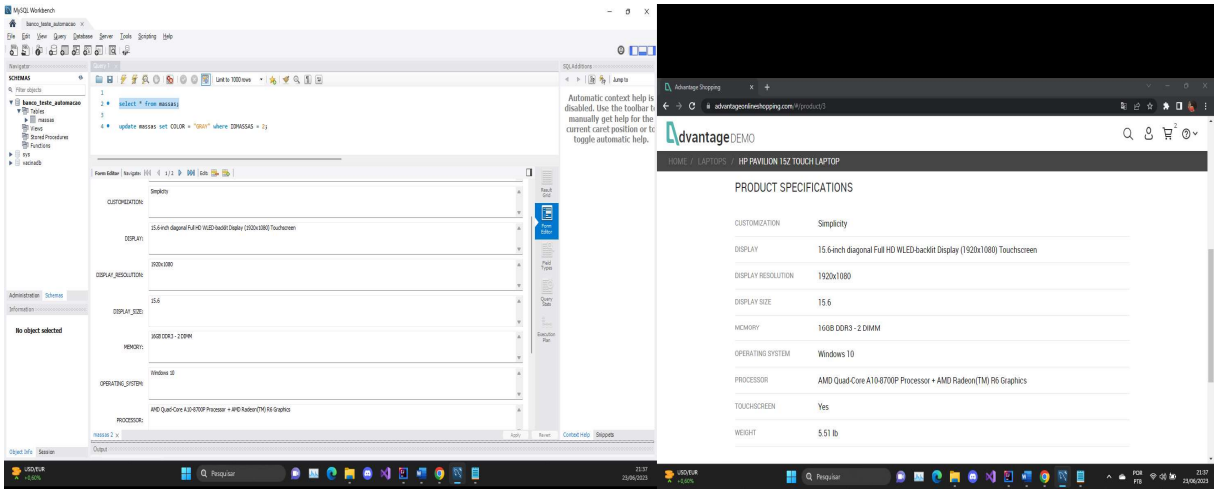
Figura 7 - *Step* 4: Alterar a cor para cinza




Fonte: Autor

A Figura 8 demonstra a validação das especificações do produto como a customização, resolução, tamanho da tela, memória, sistema operacional, processador, se é touchscreen e o peso, conforme está cadastrado no banco de dados.

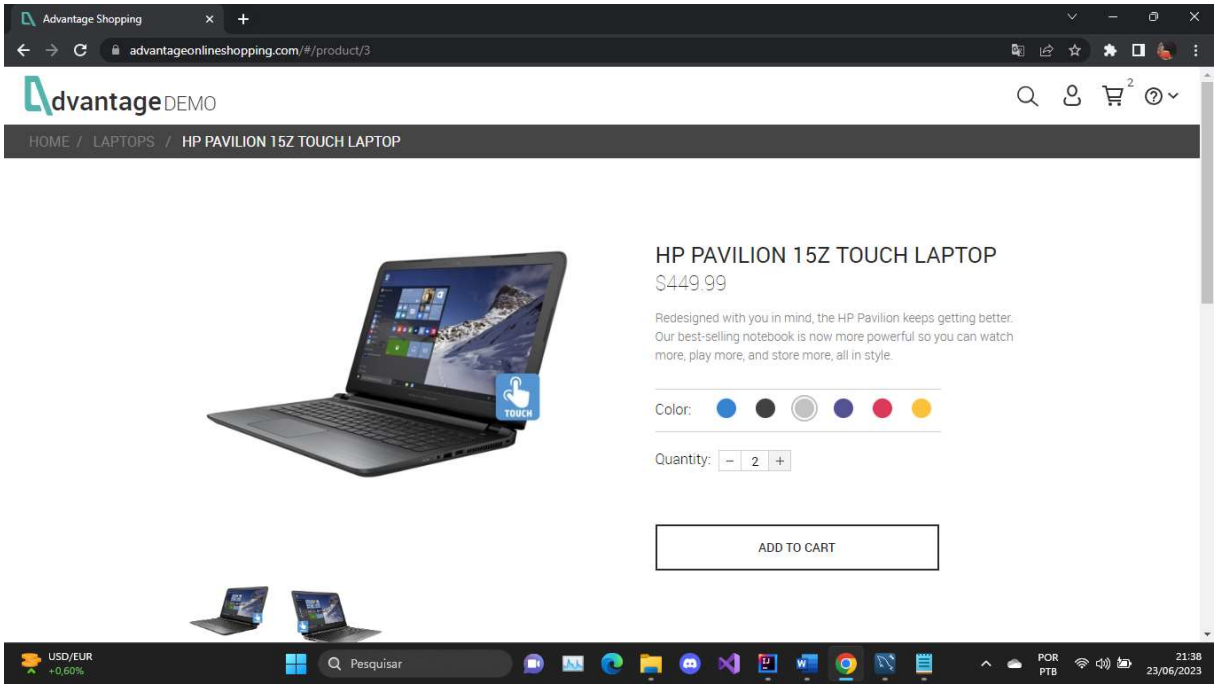
Figura 8 - *Step* 5: Valida especificação do produto conforme o cadastro no banco




Fonte: Autor

Após as especificações do produto, é demonstrado pela Figura 9 a opção de escolher a quantidade de produtos que deseja comprar.

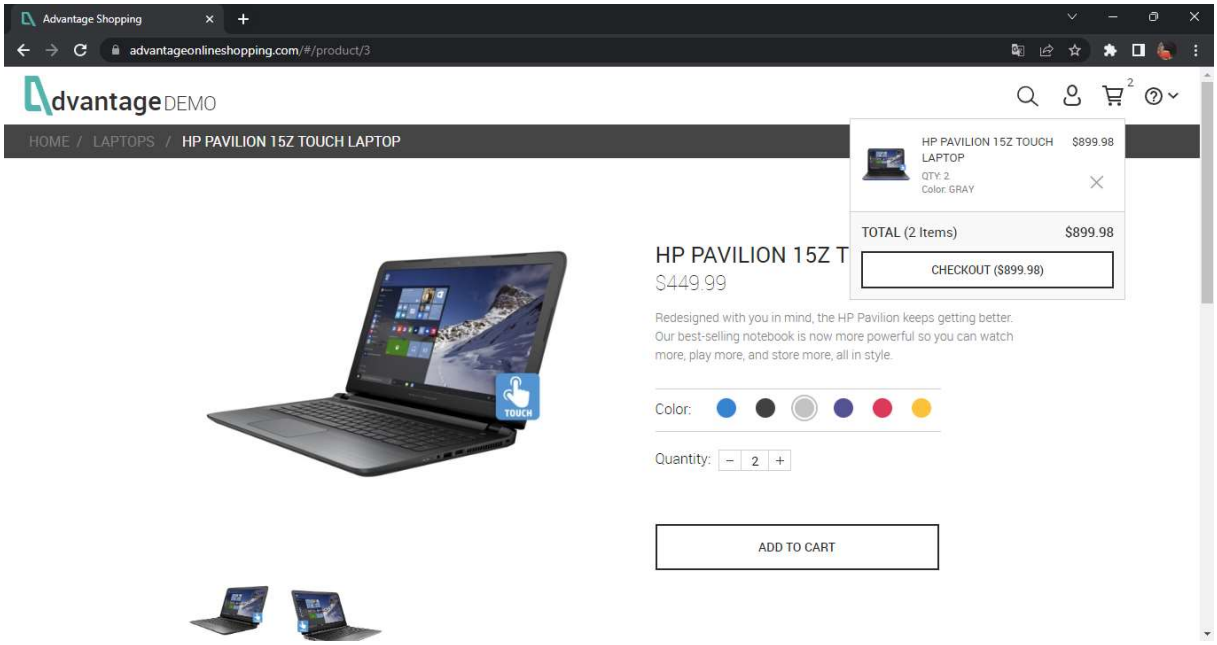
Figura 9 - *Step* 6: Altera a quantidade do produto que deseja compra




Fonte: Autor

Ao escolher a quantidade de produtos, é selecionado a opção adicionar ao carrinho conforme demonstrador pela Figura 10.

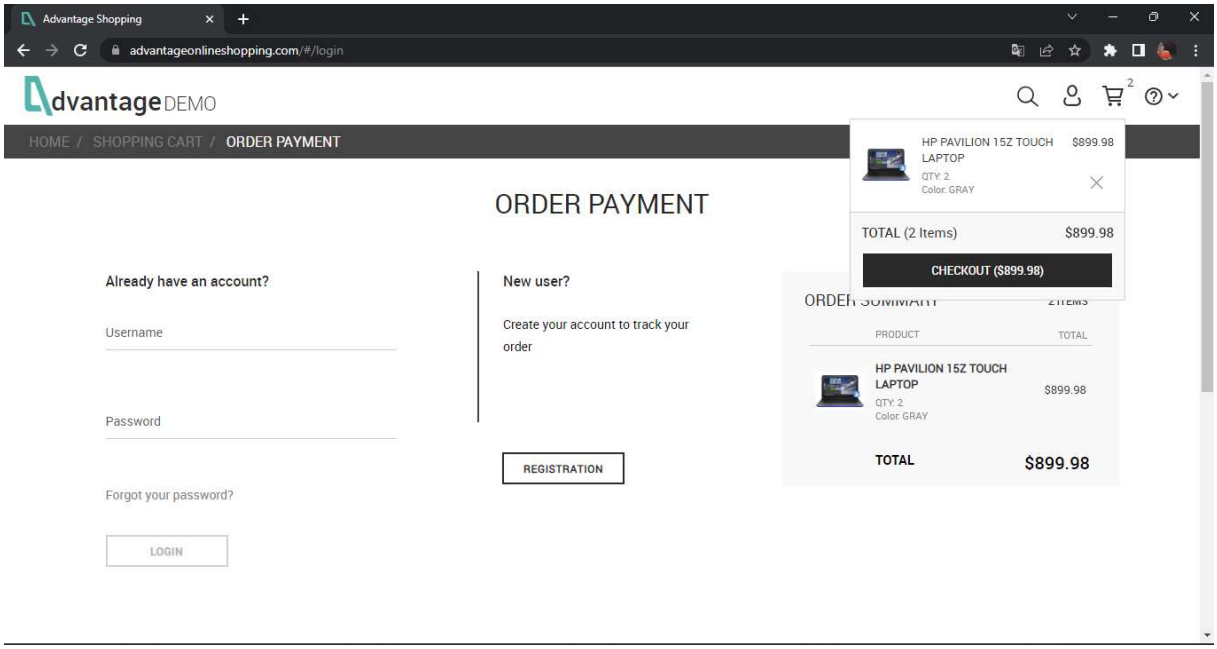
Figura 10 - *Step* 7: Clica no botão *Add To Cart*




Fonte: Autor

Quando é adicionado ao carrinho, um pop-up é apresentando com informações do produto e é selecionado o botão *checkout* conforme demonstrador pela Figura 11.

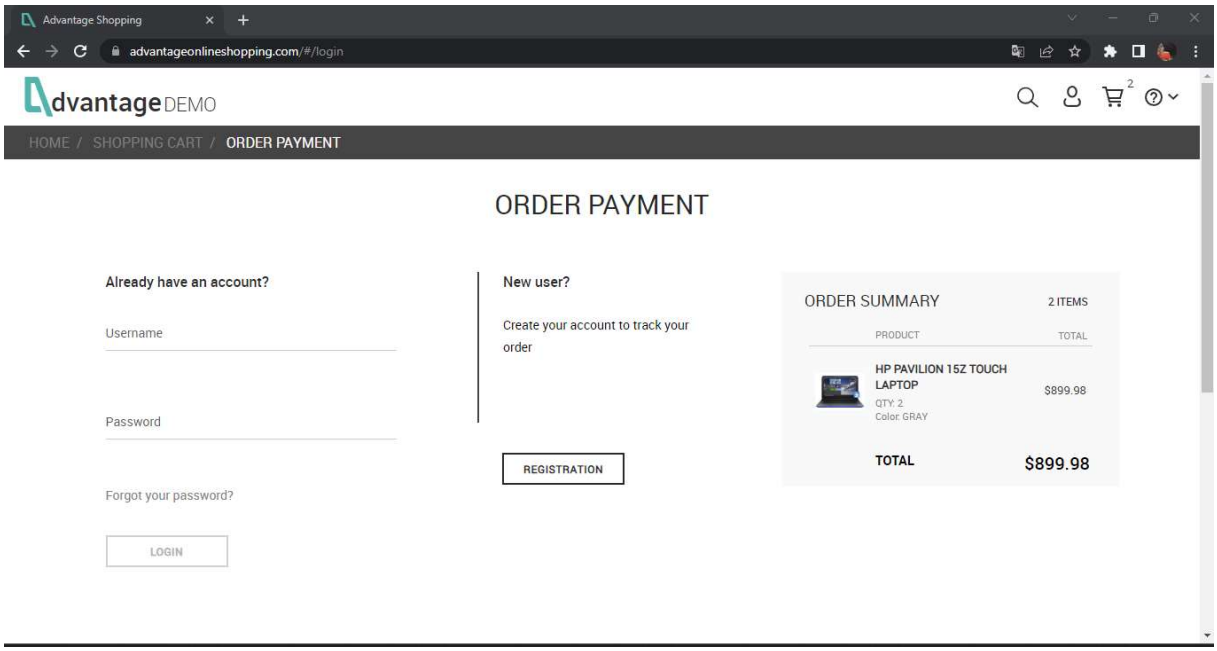
Figura 11 - *Step* 8: Acessa a página *Checkout*




Fonte: Autor

Ao ser direcionado para a tela de *checkout*, é feito a validação do valor total dos produtos escolhidos, essa validação é identificada pela Figura 12.

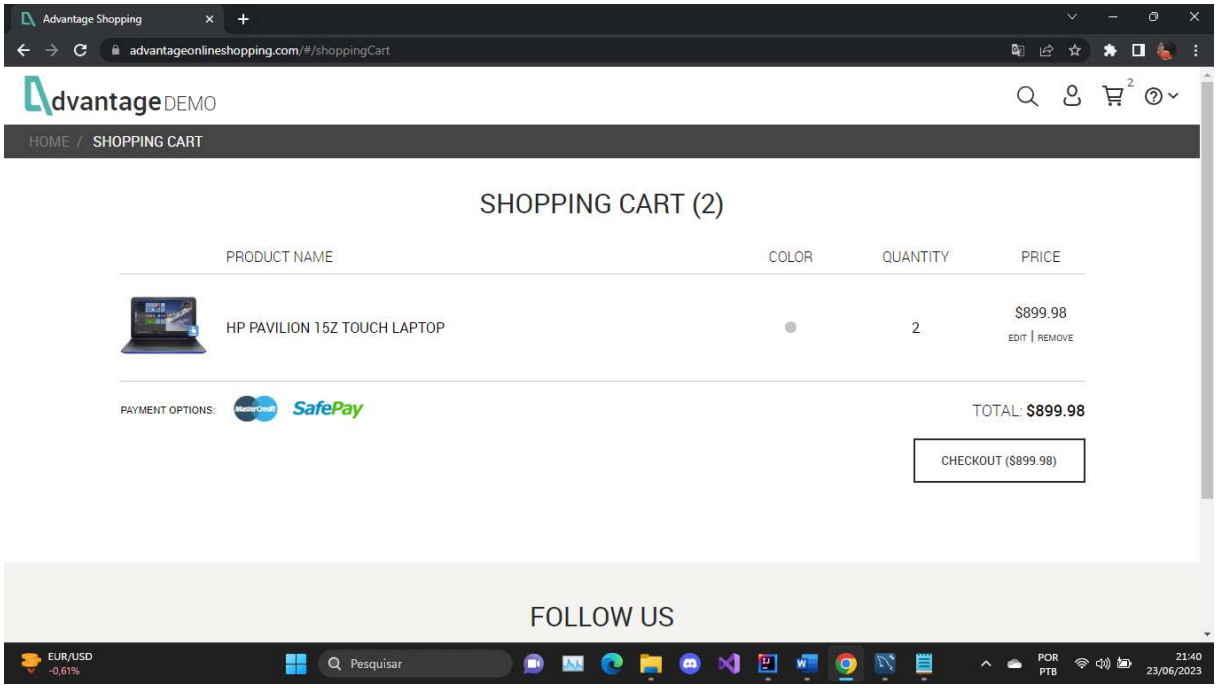
Figura 12 - *Step* 9: Valida o preço está correspondente a quantidade de produtos




Fonte: Autor

Ao ser feito a validação do valor total dos produtos, é acessado a opção do carrinho de compras como ilustrado pela Figura 13.

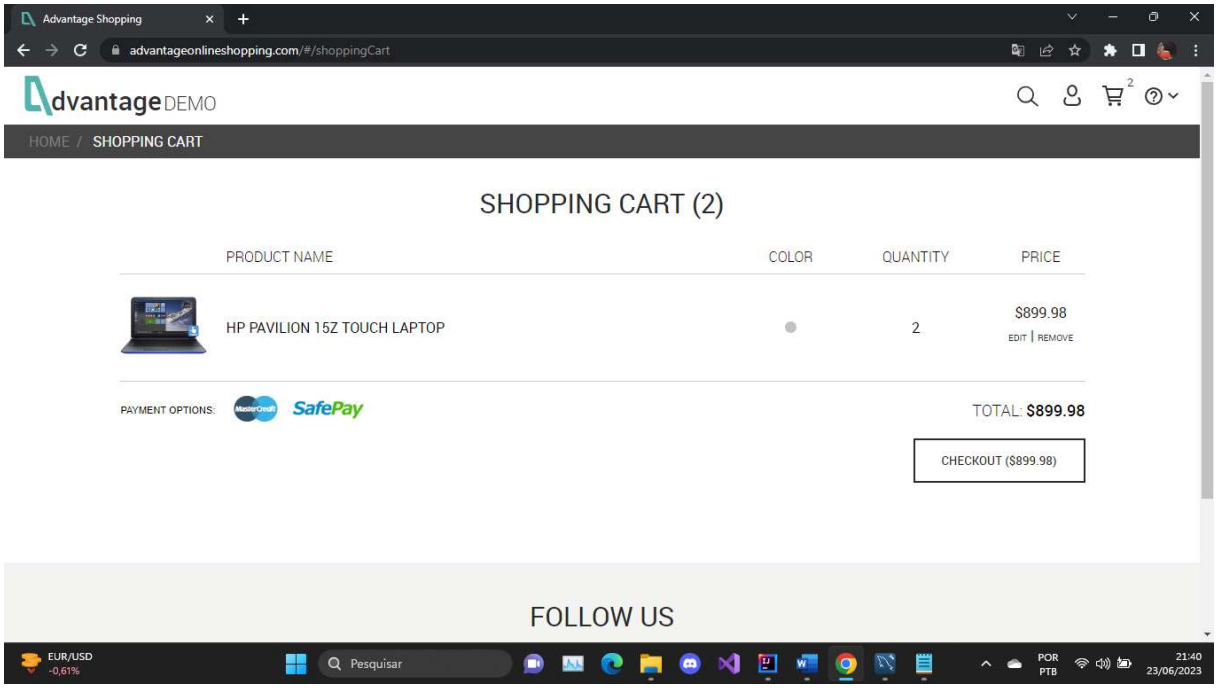
Figura 13 - *Step* 10: Clicar no carrinho de compras




Fonte: Autor

Na Figura 14, é apresentando que os produtos escolhidos foram adicionados com a cor correta.

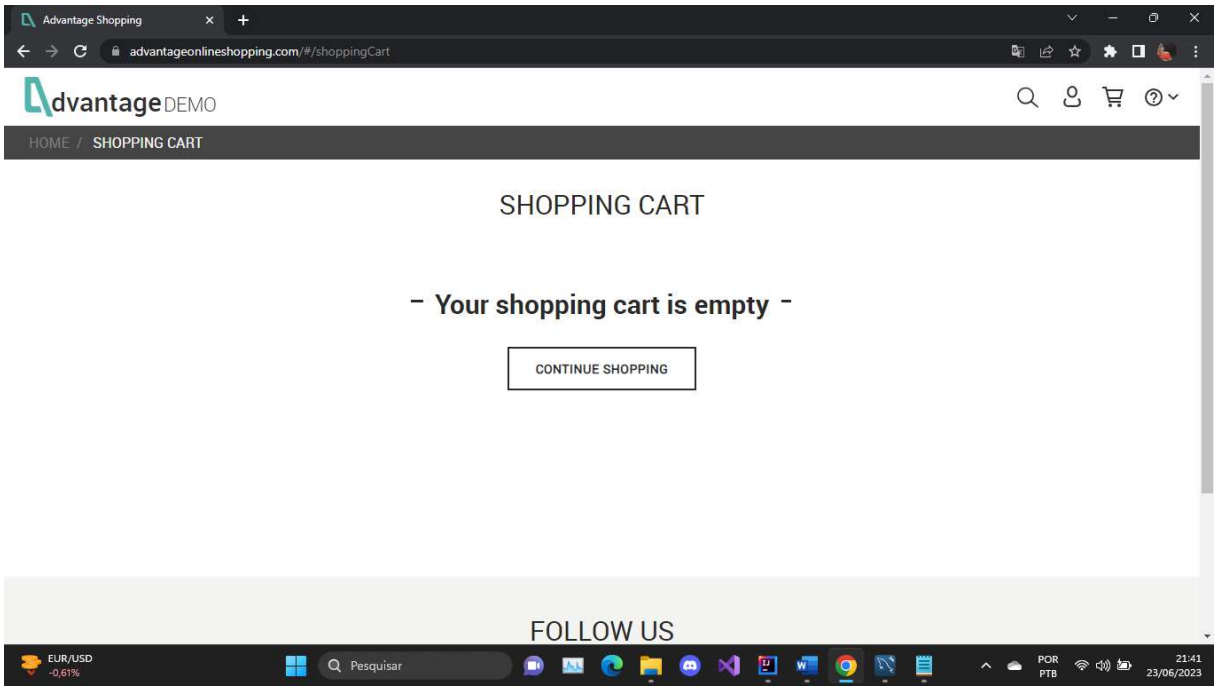
Figura 14 - *Step* 11: Valida que produto foi adicionado com a cor correta




Fonte: Autor

Quando é validado que o produto foi adicionado com a cor correta, é feito um clique na opção *remove* para remover os produtos do carrinho de compra conforme identificado pela Figura 15.

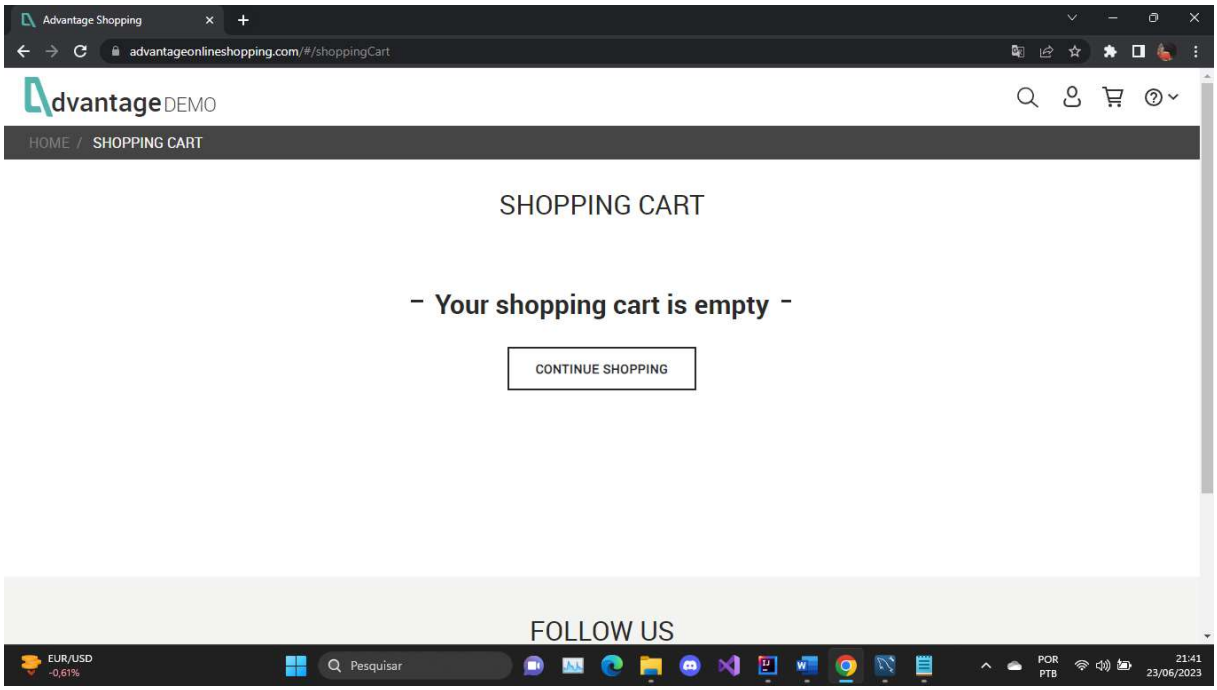
Figura 15 – *Step* 12: Remove os produtos do carrinho




Fonte: Autor

Por fim, a Figura 16 demonstra que ao ser removido os produtos, o carrinho de compras se encontra vazio.

Figura 16 - *Step* 13: Valida que o carrinho de compras está vazio




Fonte: Autor

O cenário de teste executado manualmente possui um total de 13 *steps*, no qual iniciou-se as 21 horas e 36 minutos e teve o fim de suas validações as 21 horas e 41 minutos. Portanto, para a realização do fluxo levou em torno de 5 minutos.

## 4.2 Testes automatizados

Para a execução do teste automatizado usamos a classe *RunnerTest*, com o auxílio de *puglins* do *cucumber* é possível selecionar qual cenário executar caso possua mais de um ou realizar a execução de todos junto. A automação irá seguir o mesmo fluxo realizado no teste manual, portanto o passo a passo é o mesmo, a única diferença é que a máquina que realizará a execução.

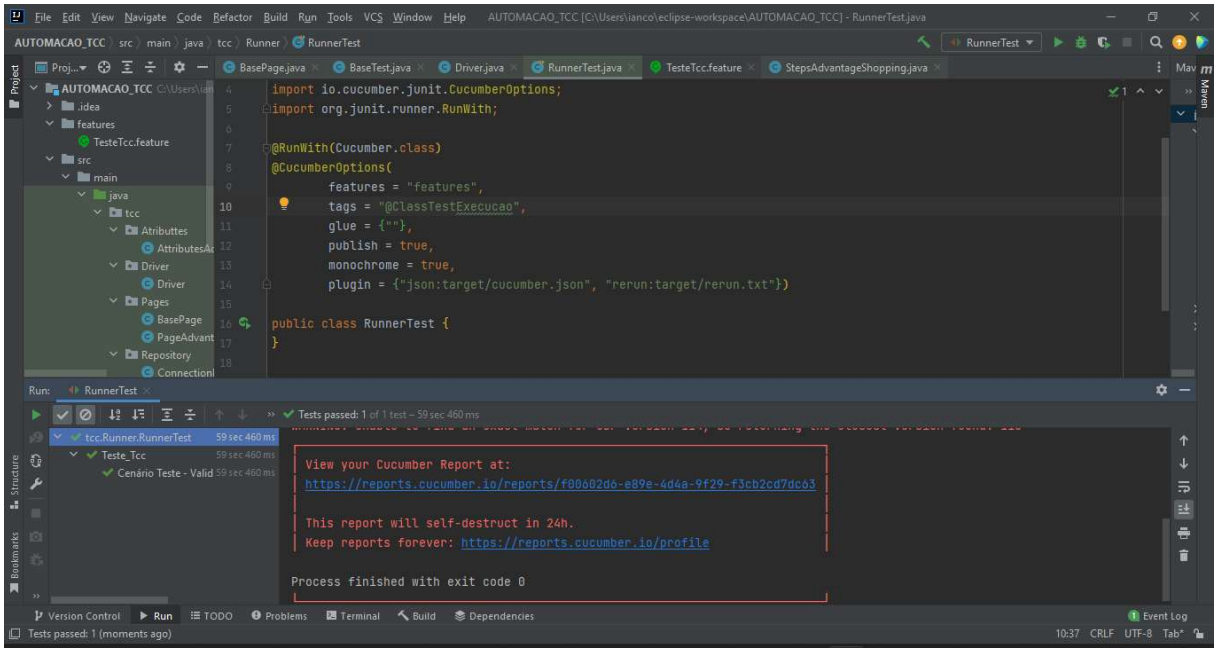
### *4.2.1 Resultados automatizados*

Abaixo serão apresentados os resultados obtidos e o desempenho do teste ao realizar a execução de forma automatizada.

Cenário Teste - Validar adição e remoção de produtos no carrinho

A Figura 17 apresenta a classe de execução do teste automatizado *RunnerTest*, é identificado também que a execução de classe obteve êxito exibindo o nome do cenário executado com um *check* verde e o link para acessar o reporte da execução.

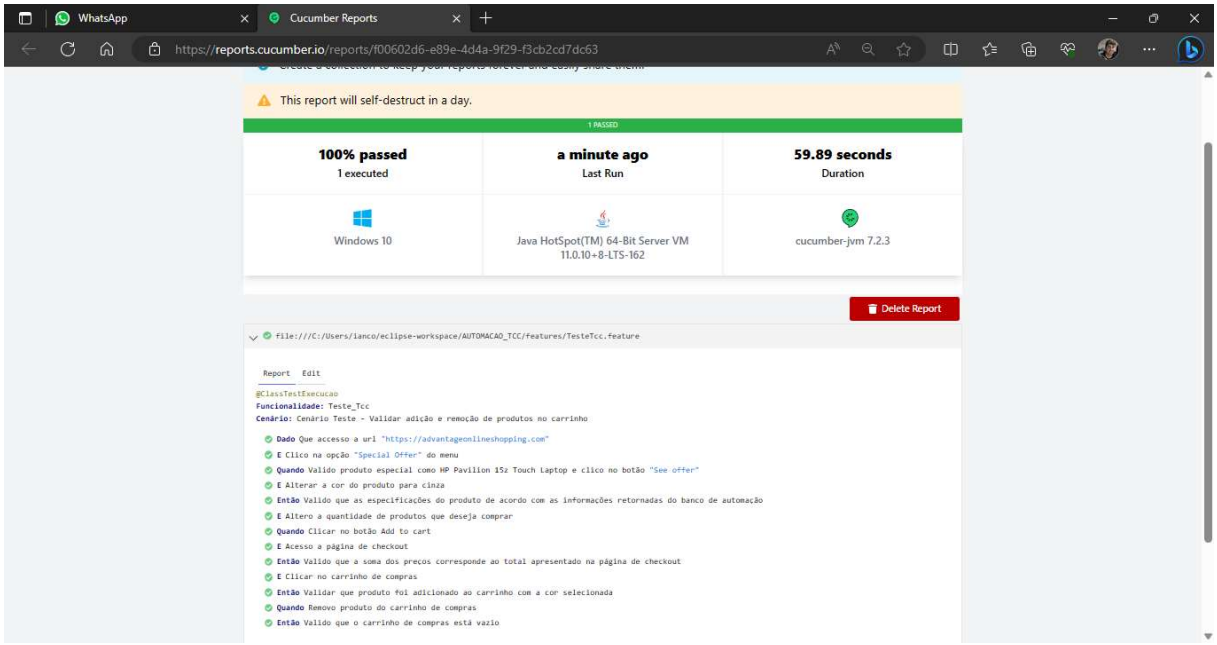
Figura 17 – *RunnerTest* do cenário




Fonte: Autor

A execução automatizada do cenário teste obteve sucesso, com tempo estimado em 59.89 segundos para finalizar todas as validações e é exibido todo o passo a passo realizado pela máquina ao acessar o link de reporte disponibilizado ao final da execução do teste, que pode ser identificado pela Figura 18.

Figura 18 – Reporte cenário teste




Fonte: Autor

Realizando a comparação das execuções, notamos que a automação obteve um desempenho quase 5 vezes melhor em relação ao teste manual. Em algumas ocasiões é necessário executar o mesmo teste várias vezes, assim, tornando a automação que tem um ganho de tempo bem maior em relação ao manual a melhor escolha para a realização dos testes, onde se pode executar os scripts quantas vezes forem necessárias sem que haja algum erro humano na execução ou perda de desempenho ocasionadas por repetição ou cansaço.

## 5 CONCLUSÕES

A globalização fez com que os s*oftwares* se tornassem essenciais para o desenvolvimento de todas as atividades em todos os setores, tanto industriais como residenciais e comerciais. Assim, seu perfeito funcionamento é essencial para produtividade e desempenho das organizações, além do lazer e entretenimento humano.

Nesse sentido, os testes de s*oftwares* configuram uma etapa essencial durante seu desenvolvimento, a fim de garantir o funcionamento perfeito à atividade a qual foi proposto. Do ponto de vista técnico, os testes atuam ainda para garantir mais produtividade ao processo de desenvolvimento desses sistemas, uma vez que os erros podem ser detectados e corrigidos a cada etapa, pois testar o *software* apenas quando esse já estiver concluído consiste em uma abordagem ineficaz. Assim é importante que os profissionais conheçam cada etapa do teste de s*oftwares*.

Levando em consideração esses pontos, este trabalho buscou realizar os testes manuais e automatizados, onde foi apresentado uma diferença de tempo considerável entre a execução manual e a automatizada, provando que a automação de testes é mais eficiente. É fato que a automação garante melhores resultados, segurança e agilidade, pois ela permite a execução e principalmente, a repetição de cenários de teste, assim, otimizando o tempo e aumentando a velocidade e confiabilidade nas aplicações.

Entretanto, um dos principais desafios em relação a automação é o bom desenvolvimento, a sua manutenção e atualização dos scripts de teste. Pois, quando estão desatualizados ou incorretos, geram falhas e acabam perdendo a eficácia. Vale lembrar que até a automação precisa da interferência humana, sendo assim, os testes automatizados não excluem a necessidade de um teste manual de *software*, onde para se ter uma boa automação é necessário partir de um processo de teste manual estabelecido e maduro.

Portanto, o teste manual continua sendo uma parte importante para se ter uma boa automatização de processos, pois, os *softwares* são utilizados por pessoas e ninguém melhor que *testers* manuais para simular as diversas situações reais em que os usuários finais vão utilizar nas aplicações.

Para um trabalho futuro, poderia utilizar a automação para realizar validações via *APIs* - Interface de Programação de Aplicativos, que possuem o importante papel de possibilitar a comunicação entre dois ou mais sistemas, facilitando o manuseio de dados de outras aplicações. Os testes de *APIs* servem para verificar as dependências das aplicações com outros servidores e certificar que a comunicação entre os aplicativos, sistemas, banco de dados estão funcionando corretamente.

## 6 REFERÊNCIAS

BASTOS, A. et al. **Base de conhecimento em teste de software**. 2 ed. São Paulo: Martins Fontes, 2007.

BERNARDO, P. C. **Padrões de Testes Automatizados**. 2011. Dissertação (Mestrado em Engenharia Industrial) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2011.

BERNARDO, P. C; KON, F. **A Importância dos Teste Automatizados**. 2008. Engenharia de Software Magazine, nº 3.

BRAGA, F. A. **Qualidade de software: proposta de automação de testes e de um processo ágil para uma empresa de software.** Monografia (Bacharel em Sistemas de Informação). Universidade do Sul de Santa Catarina. Florianópolis. 2019.

KOSCIANSKI, A., SOARES, M. S. **Qualidade de Software**. 2.ed. São Paulo: Novatec, 2007.

LIMA, A. G. S. **Automação de testes funcionais em ambientes web: um estudo de caso no laboratório de sistema e bancos de dados da universidade federal do ceará.** Monografia (Bacharel em Engenharia de Software). Universidade Federal do Ceará. Quixadá. 2014.

LUÍS, B. **Melhoria de Conhecimentos em Garantia de Qualidade no Software (Tipos de Teste)**. Disponível em: https://slidex.tips/download/melhoria-de-conhecimentos-em-garantia-de-qualidade-no-software-tipos-de-teste. Acesso em: 16 nov. 2022.

PRESSMAN, R. S. **Engenharia de Software**, 6ª Edição. Porto Alegre: McGraw-Hill, 2006.

PRESSMAN, R. S. **Engenharia de Software: uma abordagem profissional**. 7. ed. São Paulo: Pearson Makron Books, 2011.

SOFTEX, RECIFE. **Fundamentos do Teste de Software**. Recife, 2011. Disponível em: http://ava.nac.softex.br/pluginfile.php/602/mod_resource/content/2/Aula%202.pdf. Acesso em: 15 nov. 2022.

SOMMERVILLE, I. **Engenharia de software**. 9ª edição. São Paulo: Pearson Prentice Hall, 2011.

TRINDADE, S. I. M. **Caso de estudo sobre automação de testes de software.** Tese (Doutor em Sistemas e Tecnologias da Informação para as Organizações). Instituto Politécnico de Viseu. 2021.