CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Vitor Campos e Silva

SELENIUM WEBDRIVER: AVALIAÇÃO DA FERRAMENTA DE AUTOMATIZAÇÃO DE TESTE FUNCIONAL

Timóteo

2018

Vitor Campos e Silva

SELENIUM WEBDRIVER: AVALIAÇÃO DA FERRAMENTA DE AUTOMATIZAÇÃO DE TESTE FUNCIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Orientador: Deisymar Botega Tavares
Coorientador: Marcelo de Sousa Balbino

Timóteo

2018

Vitor Campos e Silva

SELENIUM WEBDRIVER: AVALIAÇÃO DA FERRAMENTA DE AUTOMATIZAÇÃO DE TESTE FUNCIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 06 de julho de 2018:

Prof. Me. Deisymar Botega Tavares
Orientadora

Prof. Me. Marcelo de Sousa Balbino
Coorientador

Prof. Me. Aléssio Miranda Júnior
Professor Convidado

Timóteo
2018

.

.

.

Dedico a todos aqueles que sempre me apoiaram e incentivaram,
sem eles nada disso seria possível.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado força e condições de chegar aonde cheguei, aos meus pais, minha irmã e demais familiares pelo apoio e incentivo e a minha namorada pela paciência, sempre estando ao meu lado e me apoiando.

Agradeço também aos amigos de caminhada, que compartilharam comigo momentos de preocupações e alegrias no decorrer do curso. Amizades essas que irei levar para o resto da vida.

E por ultimo e não menos importante, agradeço a minha orientadora Deisymar e meu coorientador Marcelo que foram pessoas excepcionais e de extrema importância para a conclusão do trabalho, sempre me auxiliando e incentivando desde o início até a conclusão do mesmo.

“Nada poderá me abalar
Nada poderá me derrotar
Pois minha força e vitória
Tem um nome
É Jesus"
Eliana Ribeiro

Resumo

Com o passar do tempo, cada vez mais são produzidos softwares sofisticados para serem utilizados no dia a dia da população. Em algumas situações, tais softwares são responsáveis por controlar negócios de alto valor de mercado ou por garantir a vida humana, fazendo com que falha nos mesmos ocasione grandes perdas. Com isso, é de extrema importância a busca pela qualidade do software, com o intuito de tentar minimizar ao máximo a possibilidade de falhas, entregando ao usuário final um produto seguro e de qualidade. Neste cenário surgem no mercado técnicas e ferramentas para auxiliar e/ou executar testes de software, garantindo maior rapidez e qualidade nos testes realizados. Apesar dos testes de software atuarem como um importante recurso e grande aliado para a obtenção da qualidade, ainda são poucos os livros que se aprofundam em tal assunto, sendo que a maioria destes livros faz apenas uma breve descrição, apresentando o conteúdo de uma forma mais conceitual. Quanto à automatização de software, também é um assunto pouco abordado em livros e artigos acadêmicos, onde, na maioria das vezes, não é demonstrado de forma prática o uso deste recurso. Com isso, o presente trabalho dedica-se a explorar uma ferramenta de automatização de teste funcional, com o intuito de verificar se a mesma possui recursos necessários para a automatização dos diversos tipos de testes referentes à técnica de teste funcional descritos na literatura. No decorrer do desenvolvimento do trabalho, passou-se por um critério de seleção onde foi escolhido o Selenium WebDriver como ferramenta a ser utilizada. Com os testes realizados utilizando o Selenium WebDriver, foi possível elencar recursos e limitações de sua utilização na automatização dos testes de software. A utilização da ferramenta é vantajosa nos cenários onde é necessário executar repetidas vezes o mesmo teste, visto que a repetição de tais testes manualmente é um trabalho exaustivo. Por outro lado, o esforço exigido para desenvolvimento dos scripts de teste pode não ser vantajoso em cenários onde os testes manuais sejam simples ou que não seja necessário realizar repetições dos mesmos. Foi possível utilizar o Selenium WebDriver para automatizar os testes de requisitos, testes de tratamento de erro, testes de regressão, testes de interconexão com outros softwares e testes paralelos, pertencentes à técnica de teste funcional. Os resultados obtidos auxiliam e são um apoio aos professores e estudantes que procuram se aprofundar no assunto.

Palavras-chave: teste de software, teste funcional,automatização de teste de software, Selenium WebDriver.

Abstract

Over time, more sophisticated software is produced for the population's everyday use. In some situations, such software is responsible for control in high value markets or guaranteeing human survival, making even small failures cause big losses. Then, it is extremelly important to try to improve software quality, aiming to minimize the possibility of failure, delivering software with quality and safety to the end user. In this scenario, some techniques have appeared on the market to aid and execute software tests, bringing higher quality and speed to the testing. Even though software testing is a great resource and valuable ally in achieving quality, there are few books going deep on the subject, as most books only do a quick description, presenting the basic concepts. Software automatization is also sparsely covered by books and academic papers, and in the instances where they are covered it usually does not involve any practical application. This work devotes itself to the exploring a functional test automatization tool, aiming to verify whether it has all the necessary resources to automate the diverse range of tests described by the literature on the functional testing technique. During the development of this work, a selection process opted for Selenium WebDriver as the tool to be used. With the tests done with Selenium WebDriver, it was possible to enumerate the resources and the limitations of its use in the software test automatization. The use of tools is advantageous in the scenarions where it is necessary to run the test repeatedly, as manual repetition of such tests is an tiresome job. On the other hand, the necessary effort for developing the test scripts can make it a disavanteage in scenarios where the manual tests are simple or there is no need for their repetition. It was possible to use Selenium WebDriver to automate the requisits tests, error treating tests, regression tests, interconnection tests with other softwares and parallel tests pertaining to the functional testing technique. The results obtained can be a resource to professors and students that seek to deepen themselves on the subject.

Keywords: software test, functional test, automation of software testing, Selenium WebDriver

Lista de ilustrações

Figura 1 – Resultados dos testes por suíte 27
Figura 2 – Abordagem do APOGEN para criação de objetos de páginas da Web 28
Figura 3 – Métodos 30
Figura 4 – Procedimentos Adotados 34
Figura 5 – Procedimentos do Cenário 1 35
Figura 6 – Criando novo evento 38
Figura 7 – Verificando a existência do evento criado 38
Figura 8 – Verificando as alterações no evento 39
Figura 9 – Cenário 1 executado com sucesso 39
Figura 10 – Procedimentos do Cenário 2 40
Figura 11 – Parte do script de teste para comparação de valores 42
Figura 12 – Procedimentos do Cenário 3 43
Figura 13 – Verificação das mensagens de erro 46
Figura 14 – Exibição das mensagens de erro do Sympla 46
Figura 15 – Execução do Cenário 3 47
Figura 16 – Procedimentos do Cenário 4 48
Figura 17 – Comparação dos dados 50
Figura 18 – Exibição das mensagens de erro das Casas Bahia 51
Figura 19 – Procedimentos do Cenário 5 53
Figura 20 – Teste executado após implementação de nova funcionalidade 55
Figura 21 – Teste executado após implementação de nova funcionalidade 56
Figura 22 – Procedimentos do Cenário 6 57
Figura 23 – Comentário no Twitter realizado com sucesso 58
Figura 24 – Comentário no Facebook verificado com sucesso 59
Figura 25 – Cenário 6 executado com sucesso 59
Figura 26 – Login no Gmail 65
Figura 27 – Cenário 1 - Primeira parte do script 73
Figura 28 – Cenário 1 - Segunda parte do script 73
Figura 29 – Cenário 1 - Terceita parte do script 74
Figura 30 – Cenário 2 - Primeira parte do script 75
Figura 31 – Cenário 2 - Segunda parte do script 75
Figura 32 – Cenário 2 - Terceita parte do script 76
Figura 33 – Cenário 3 - Primeira parte do script 77
Figura 34 – Cenário 3 - Segunda parte do script 78
Figura 35 – Cenário 4 - Primeira parte do script 79
Figura 36 – Cenário 4 - Segunda parte do script 80
Figura 37 – Cenário 4 - Terceita parte do script 80
Figura 38 – Cenário 5 - Primeira parte do script 81
Figura 39 – Cenário 5 - Segunda parte do script 81

Figura 40 – Cenário 6 - Primeira parte do script 82
Figura 41 – Cenário 6 - Segunda parte do script 83

Lista de quadros

Quadro 1 – Exemplo de um Procedimento de Teste 20
Quadro 2 – Exemplo de um Caso de Teste 21
Quadro 3 – Comandos do Selenium Webdriver 23
Quadro 4 – Palavras Chaves para Pesquisas Bibliográficas 24
Quadro 5 – Critérios de seleção 32
Quadro 6 – Critérios atendidos 32
Quadro 7 – Procedimento de Teste 01 36
Quadro 8 – Caso de Teste 01 36
Quadro 9 – Procedimento de Teste 02 37
Quadro 10 – Caso de Teste 02 37
Quadro 11 – Procedimento de Teste 03 41
Quadro 12 – Caso de Teste 03 41
Quadro 13 – Procedimento de Teste 04 43
Quadro 14 – Caso de Teste 04 44
Quadro 15 – Procedimento de Teste 05 44
Quadro 16 – Caso de Teste 05 45
Quadro 17 – Procedimento de Teste 06 48
Quadro 18 – Caso de Teste 06 49
Quadro 19 – Procedimento de Teste 07 53
Quadro 20 – Caso de Teste 07 54
Quadro 21 – Procedimento de Teste 08 57
Quadro 22 – Caso de Teste 08 58
Quadro 23 – Selenium WebDriver - Vantagens x Limitações 67

Sumário

1 INTRODUÇÃO 12
1.1 Motivação 13
1.2 Objetivo 13
1.3 Organização do Trabalho 14

2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 Teste de Software 15
2.2 Teste Funcional 15
2.2.1 Teste de Requisitos 16
2.2.2 Teste de Regressão 17
2.2.3 Teste de Tratamento de Erros 17
2.2.4 Teste de Suporte Manual 18
2.2.5 Teste de Interconexão com outros Softwares 18
2.2.6 Teste de Controle 19
2.2.7 Teste Paralelo 19
2.3 Preparação dos Testes 20
2.4 Automatização de Teste de Software 21
2.5 Selenium IDE 21
2.6 Selenium WebDriver 22

3 ESTADO DA ARTE 24
3.1 Ferramenta de Suporte a Automatização de Teste Funcional: Levantamento Bibliográfico 24
3.2 Avaliação de Qualidade no Sistema de Locações da Faculdade de Balsas Através de Testes Funcionais Automatizados 26
3.3 APOGEN: automatic page object generator for web testing 27

4 MATERIAIS E MÉTODOS 29

5 ANÁLISE E DISCUSSÃO DOS RESULTADOS 32
5.1 Teste de Requisitos 34
5.1.1 Cenário 1 - Teste Automatizado de Execução e Confirmação de um Cadastro 34
5.1.2 Cenário 2 - Teste Automatizado em um Carrinho de Compras 40
5.2 Teste de Tratamento de Erros 42
5.2.1 Cenário 3 - Teste Automatizado para Verificar o Tratamento de Erro 42
5.2.2 Cenário 4 - Teste Automatizado para Tratamento de Erro de uma Tela de Cadastro 47
5.3 Teste de Regressão 52
5.3.1 Cenário 5 – Teste Automatizado no Sistema Clubes Online 52

5.4 Teste de Interconexão com outros Softwares 55
5.4.1 Cenário 6 - Teste de Conexão entre Twitter e Facebook 55
5.5 Teste Paralelo 60
5.6 Vantagens encontradas 61
5.7 Limitações encontradas 62
5.7.1 Automatização de páginas que possuem Captcha 62
5.7.2 Automatização em determinados sites responsivos 63
5.7.3 Automatização em páginas que não possuem elementos estáticos 63
5.7.4 Reexecução do script de teste 64
5.7.5 Automatização em páginas que possuem o código html complexo 65
5.7.6 Localizar elemento pelo layout 66
5.8 Vantagens x Limitações 66

6 CONCLUSÃO 68

REFERÊNCIAS 71

A SCRIPT DE TESTE – CENÁRIO 1 73

B SCRIPT DE TESTE – CENÁRIO 2 75

C SCRIPT DE TESTE – CENÁRIO 3 77

D SCRIPT DE TESTE – CENÁRIO 4 79

E SCRIPT DE TESTE – CENÁRIO 5 81

F SCRIPT DE TESTE – CENÁRIO 6 82

1 Introdução

Os softwares estão presentes no nosso dia-a-dia e são responsáveis por controlar desde equipamentos mais simples, como um micro-ondas, até equipamentos mais robustos e sofisticados, como um avião. Existem também os tipos de software que são essenciais no controle dos negócios dos mais diversos tipos de instituições, além dos mais populares, que são utilizados constantemente pelas pessoas no decorrer do dia, como os aplicativos comerciais e sistemas operacionais.

A busca pela qualidade passou a ser algo fundamental e de extrema importância para os desenvolvedores, visto que falhas nesses softwares podem resultar em grandes perdas financeiras e de tempo, chegando em alguns casos a ocasionar perdas de vidas humanas.

Segundo Bernardo e Kon (2008), o controle da qualidade dos sistemas de software tem se tornado, cada vez mais, um enorme desafio devido à grande complexidade dos produtos desenvolvidos e às inúmeras dificuldades encontradas no processo de desenvolvimento (questões humanas, burocráticas, técnicas, de negócios e políticas).

Em um cenário ideal, os sistemas deveriam ir além da preocupação em fazer corretamente o que o cliente necessita, se preocupando também em fazê-lo de forma segura e eficiente. Deve-se ter como produto final sistemas flexíveis, de fácil evolução e manutenção.

> Neste sentido, o tema Teste de Software vem sendo muito discutido. Atualmente existem várias abordagens de teste e cada vez mais surgem outras novas. A principal abordagem utilizada atualmente em muitas empresas de desenvolvimento, principalmente as de médio para pequeno porte, é a utilização dos testes manuais. Nela é atribuída a função de testar as aplicações desenvolvidas a um membro da equipe de forma manual. No entanto, essa abordagem é muito ineficaz e demorada. As melhores técnicas de teste são baseadas em algum processo automatizado, onde é possível executar uma maior quantidade de testes, buscando assim testar ao máximo os requisitos de um software. (DAILBERT; TOLEDO; ARAUJO, 2008, p. 53)

Caetano (2008) diz que a automatização de testes é uma área em expansão, mas que ainda se mostra muito imatura, onde grande parte do sucesso obtido nos projetos de automatização de testes é decorrente de processos empíricos de tentativa e erro.

Segundo Neto (2008b), um dos motivos de tal imaturidade se dá pelo fato de que nos livros tradicionais de Engenharia de Software, geralmente encontra-se apenas algum capítulo ou seção com uma descrição breve e superficial sobre testes de software, sempre de forma básica e conceitual. Porém, quando se trata do desenvolvimento/utilização dos testes em algum cenário real, tais informações apresentadas não são suficientes para dar suporte no desenvolvimento dos testes.

No trabalho de conclusão de curso “Ferramentas de Suporte a Automação de Teste Funcional: Levantamento Bibliográfico” desenvolvido pela aluna Jessica Vieira (SOARES, 2017) é realizado um levantamento bibliográfico com o intuito de verificar a existência de ferramentas

de automatização de teste funcional, apresentando características das mesmas e analisando-as com base nas características relatadas nas referências bibliográficas coletadas.

O presente trabalho propõe a dar continuidade ao trabalho desenvolvido por Soares, escolhendo uma determinada ferramenta de automatização de teste funcional para verificar se a mesma atende aos requisitos do teste funcional. Sendo assim, o mesmo visa ser um apoio para os leitores que pretendem se aprofundar na automatização de testes, dando um enfoque na automatização de testes funcionais.

1.1 Motivação

No trabalho desenvolvido por Soares (2017), são apresentadas informações pontuais acerca das ferramentas abordadas. Para tais ferramentas, por não ter sido realizado uma analise mais detalhada, a autora apresenta informações que não possibilitam que os leitores se aprofundem no estudo das mesmas e nem conheçam grande parte de seus recursos e funcionalidades.

Matieli e Oliveira de Araujo (2015) afirmam que há uma grande carência no que se diz respeito à existência de documentos técnico-científicos referentes à área de testes, gerando uma fragilidade às organizações e profissionais que trabalham diretamente com teste de software.

Percebe-se então a necessidade e importância de realizar um estudo sobre uma das ferramentas citadas por Soares em seu trabalho, verificando o comportamento da mesma diante de diferentes cenários que abordem a automatização de testes funcionais, assim como de seus recursos e limitações quando se trata da automatização desse tipo de teste.

Dessa forma deve-se ser capaz de explorar a ferramenta escolhida, de forma a apresentar os diversos recursos e limitações que a mesma possui, visando contribuir com a automatização dos testes.

1.2 Objetivo

O presente trabalho tem como objetivo explorar uma ferramenta dentre todas citadas por Soares (2017) em seu trabalho e realizar uma análise da mesma, com o intuito de verificar se tal ferramenta possui recursos necessários para a realização dos diversos tipos de testes que atendem a técnica de teste funcional descritos na literatura.

Espera-se desenvolver uma documentação com o objetivo de contribuir com a área de testes de software, além de ser um material de auxílio a docentes e discentes que queiram aprofundar seus conhecimentos sobre a utilização de ferramentas de testes para o desenvolvimento de testes automatizados.

Como objetivo específico, o trabalho em questão se propõe a:

1. Elencar as vantagens observadas com a utilização da ferramenta escolhida no cenário de automatização dos testes funcionais;
2. Relatar as limitações encontradas ao explorar a ferramenta, onde a mesma não seja capaz de cobrir determinado tipo de teste submetido;
3. Produzir testes automatizados capazes de contemplar cenários que abordem a automatização de cada tipo de teste funcional, relatando se a ferramenta utilizada foi capaz de cobrir ou não tais testes.

1.3 Organização do Trabalho

O trabalho em questão encontra-se organizado em capítulos que abordam todo o conteúdo necessário para um bom entendimento do que se foi desenvolvido.

O Capítulo 1 contém a “Introdução” onde se é apresentado resumidamente uma visão geral de todo o conteúdo do trabalho.

O Capítulo 2 apresenta a “Fundamentação Teórica”, com os principais conceitos e tecnologias utilizadas para o desenvolvimento do trabalho, e que fazem parte dos conhecimentos necessários para um bom entendimento do mesmo.

No Capítulo 3 é apresentado o “Estado da Arte”. Nessa seção são apresentados os artigos relacionados ao tema do trabalho desenvolvido, que foram encontrados durante as pesquisas, mas que se diferem em determinados aspectos.

No Capítulo 4 são descritos os “Materiais e Métodos”. Seção que relata todas as etapas percorridas desde o início da execução do projeto até a conclusão do mesmo.

O Capítulo 5 apresenta a “Analise e Discussão dos Resultados”. Nessa seção são apresentados os resultados obtidos a partir da exploração da ferramenta escolhida, no que se diz respeito à automatização de testes funcionais.

O Capítulo 6 é sobre a “Conclusão”. Nesse capítulo são apresentadas as conclusões obtidas por meio dos dados gerados na seção “Analise e Discussão dos Resultados” e apresenta sugestões de trabalhos futuros.

2 Fundamentação Teórica

Os assuntos abordados nesse capítulo visam descrever conceitos relacionados aos testes de software, dando enfoque nos testes funcionais, na automatização de teste de software e na ferramenta utilizada para o desenvolvimento do presente trabalho.

2.1 Teste de Software

Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado (NETO, 2008a). Testar um software tem por objetivo identificar falhas em um produto, a fim de que as mesmas sejam corrigidas pela equipe de desenvolvimento antes da entrega ao usuário final.

Neto (2008a) ainda diz que, de uma forma simples, testar um software significa verificar através de uma execução controlada se o seu comportamento corre de acordo com o que foi especificado inicialmente.

O objetivo principal dessa tarefa consiste em revelar o maior número de falhas utilizando o menor esforço possível. Ou seja, demonstrar aos desenvolvedores se os resultados encontrados estão ou não dentro do esperado.

Bastos et al. (2012) dizem que a realização dos testes deve ser capaz de descobrir erros na codificação das funcionalidades definidas nas especificações do programa além dos erros cometidos durante a codificação.

Esses autores separam os testes de software em testes estruturais e testes funcionais. Os testes estruturais garantem uma estruturação sólida do software, assegurando que o mesmo funcione no contexto técnico onde será instalado, enquanto os testes funcionais garante que os requisitos e as especificações do sistema estejam conforme o esperado.

Para Maldonado et al. (2004) as atividades de VVT – Verificação, Validação e Teste, têm como objetivo minimizar a ocorrência de tais erros e riscos associados. Segundo o autor, dentre as técnicas de verificação e validação, o teste de software é uma das mais utilizadas, por se tratar de um dos elementos que vizam fornecer evidências de confiabilidade do sistema testado.

2.2 Teste Funcional

Teste funcional representa uma categoria de técnicas de teste em que o componente de software a ser testado é abordado como se fosse uma caixa-preta, ou seja, não se considera o comportamento interno do mesmo (NETO, 2010). Nesse tipo de teste são fornecidos

os dados de entrada e tem-se um resultado esperado previamente conhecido. Haverá sucesso no teste se, ao inserir dados previamente conhecidos como entrada, o resultado obtido com a execução for igual ao resultado esperado.

Segundo Neto (2010) o componente de software a ser testado pode ser um método, uma função interna, um programa, um componente, um conjunto de programas e/ ou componentes ou mesmo uma funcionalidade.

Bastos et al. (2012) afirmam que esse tipo de teste visa garantir que os requisitos e as especificações do sistema sejam atendidos. Segundo os autores, os testes de requisitos, testes de regressão, testes de tratamento de erros, testes de suporte manual, testes de interconexão com outros softwares, testes de controle e testes paralelos são os tipos de testes que atendem a técnica de teste funcional. As definições dos autores acerca de cada um desses tipos de teste estão detalhadas da seção 2.2.1 até a seção 2.2.7.

2.2.1 Teste de Requisitos

O teste de requisitos tem como objetivo verificar a correta execução das funcionalidades do sistema. Esse tipo de teste normalmente é realizado após o programa se tornar operacional e é basicamente realizado por meio de checklists das funcionalidades do sistema, assim como da criação de condições de testes. Segundo os autores, os principais objetivos do teste de requisitos são verificar se:

- Os requisitos dos usuários estão implementados;
- A correção é mantida por períodos prolongados de tempo;
- O processamento da aplicação está em conformidade com as políticas e os procedimentos da organização.

Inicialmente, as condições de testes devem ser preparadas de forma genérica na fase de requisitos. Com o decorrer do ciclo de vida do software, tais condições de teste são detalhadas, até a preparação dos dados que serão utilizados nos testes do sistema. Toda a aplicação deve ter seus requisitos testados.

Nesse senário, Bastos et al. (2012) recomendam que as condições de testes sejam elaboradas a partir dos requisitos, visto que se as mesmas forem elaboradas por meio das documentações do sistema, poderão apresentar defeitos não identificados nesses documentos, afetando assim a qualidade dos testes.

Segundo Sommerville (2011), os testes de requisitos têm como intuito realizar a validação de determinado teste ao invés de procurar por defeitos, onde deve ser demonstrado que o sistema implementou corretamente seus requisitos.

O autor também afirma que deve ser mantido os registros de rastreabilidade dos testes de requisitos, pois estes ligam os testes aos requisitos específicos que estão sendo testados.

2.2.2 Teste de Regressão

Os testes de regressão visam testar, após a implementação de uma mudança no software, segmentos implementados que já foram testados anteriormente.

Segundo Bastos et al. (2012), sempre que surgem mudanças no código do sistema, podem ocorrer problemas em segmentos já testados. Dessa forma, o teste de regressão tem como principais objetivos:

- Determinar se a documentação do sistema permanece atual;
- Determinar se os dados e as condições de teste permanecem atuais;
- Determinar se as funções previamente testadas continuam funcionando corretamente após a introdução de mudanças no sistema.

Sempre que o software sofrer qualquer tipo de alteração, deve-se realizar o teste de regressão. Deve-se realizar o teste na parte alterada, assim como nos outros segmentos do sistema, com o intuito de verificar se não houve alterações, após as mudanças, nos resultados obtidos anteriormente.

Um dos problemas gerados ao se realizar tal tipo de teste é quanto ao excessivo gasto de tempo, assim como ao cansativo trabalho de repetições das operações realizadas, contribuindo para que este tipo de teste seja realizado parcialmente ou não seja realizado. Tal problema pode ser resolvido com a utilização da automatização de testes por parte da organização.

Segundo Paula Filho (2012), o teste de regressão consiste em repetidos testes de um sistema ou componente com o intuito de verificar se as alterações realizadas no mesmo não causaram efeitos indesejáveis, garantindo que o sistema ou componente testado mantenha a conformidade com os requisitos especificados.

O autor também diz que a automatização de testes é considerada praticamente indispensável para os testes de regressão. Tais testes, feitos com base nas baterias de testes de sistema, são particularmente importantes durante a manutenção do software, visto que nessa fase é mais comum que as alterações realizadas afetem outras porções do código.

2.2.3 Teste de Tratamento de Erros

Este tipo de teste tem como finalidade verificar se o sistema é capaz de tratar transições incorretas. Bastos et al. (2012) afirmam que em alguns softwares, é dedicado aproximadamente 50% do esforço de programação para o tratamento das condições de erro.

São objetivos específicos do Teste de Tratamento de Erros:

- Determinar se todas as condições de erro esperadas são reconhecidas pelo sistema;
- Determinar se foi atribuída responsabilidade para processar os erros identificados;

- Determinar se é mantido um controle razoável sobre os erros durante o processo de correção.

Deve-se fazer uma análise para identificar o que pode dar de errado no sistema. Com os possíveis erros obtidos, pode-se separar por funcionalidades do sistema, a fim de criar conjuntos de teste para cada uma dessas funcionalidades.

Nesse tipo de teste deve-se realizar um ciclo, onde primeiramente os erros são introduzidos no sistema, identificados e corrigidos. Logo após os mesmos erros devem ser novamente introduzidos, para que o ciclo se complete.

2.2.4 Teste de Suporte Manual

Segundo Bastos et al. (2012), a parte manual do sistema exige a mesma atenção que a parte automatizada processada no computador. Para os autores, embora os momentos e os métodos utilizados para a realização destes dois tipos de testes possam ser diferentes, os objetivos do Teste de Suporte Manual são:

- Verificar se os procedimentos de suporte manual estão documentados e completos;
- Determinar se as responsabilidades pelo suporte manual foram estabelecidas;
- Determinar se o pessoal que dará o suporte manual está adequadamente treinado;
- Determinar se o suporte manual e o segmento automatizado estão interligados apropriadamente.

Esse tipo de teste pode ser realizado pelos desenvolvedores do sistema, assim como por pessoas que não desenvolveram o sistema, desde que essas sejam treinadas e aprendam os procedimentos a serem executados.

2.2.5 Teste de Interconexão com outros Softwares

Os testes de interconexão com outros softwares visam garantir que funcione corretamente a interconexão entre os diversos tipos de software que uma aplicação possa ter. Tal interconexão pode se dar por meio do recebimento e fornecimento de dados.

Segundo Bastos et al. (2012), os softwares da aplicação costumam estar conectados com outros softwares similares. Os autores ainda afirmam que testar a conexão de vários softwares, além de ser uma atividade demorada, pode ser muito custosa.

O processo nesse tipo de teste consiste basicamente em realizar transições de dados entre os softwares envolvidos, verificando se a transferência foi realizada com sucesso. Com isso, tem-se como objetivo desse tipo de teste:

- Determinar se os parâmetros e dados são transferidos corretamente entre os softwares;

- Garantir o momento certo de execução e a existência de coordenação das funções entre os softwares;
- Determinar se a documentação pertinente é correta e completa.

2.2.6 Teste de Controle

A integridade de arquivo, a validação de dados, as trilhas de auditoria, o backup, entre outros aspectos do sistema relacionados à integridade são referentes ao controle.

Para Bastos et al. (2012), esse tipo de teste é a ferramenta de gestão necessária para assegurar que o processamento do software seja realizado conforme o esperado.

Para os autores, o teste de controle pode ser considerado um sistema dentro do outro, sendo os objetivos desse tipo de teste garantir que:

- Os dados estejam completos e corretos;
- A manutenção das informações da trilha de auditoria seja realizada;
- Os processamentos sejam eficientes, eficazes e econômicos;
- O processamento atenda às necessidades dos usuários.

Os controles são desenhados para reduzir ao máximo eventuais riscos, e, para isso, tais riscos devem ser identificados.

2.2.7 Teste Paralelo

Para Bastos et al. (2012), o teste paralelo é um tipo de teste que serve para determinar se os resultados obtidos com o desenvolvimento de uma nova versão do software são consistentes com o processamento do sistema antigo.

Esse tipo de teste pode ser realizado com o software inteiro ou com segmentos do mesmo. Para tal, devem-se ser utilizados os mesmos valores de entradas nas duas versões distintas da mesma aplicação. Se a nova versão modificar o formato da entrada dos dados, então os testes devem ser adaptados para atender a tal modificação.

Tem-se como objetivos dos testes paralelos:

- Assegurar que a nova versão do software de aplicação execute corretamente;
- Demonstrar consistências e inconsistências entre as duas versões do mesmo software de aplicação.

Este tipo de teste deve ser utilizado quando houver incertezas com relação aos resultados obtidos na execução da nova versão do software em relação a anterior, caso as duas versões sejam similares.

2.3 Preparação dos Testes

Segundo Filho (2001), um Processo de Testes consiste basicamente em duas etapas, sendo elas a preparação e a realização. A etapa de preparação envolve realizar o Plano de Teste e o Projeto de Teste, enquanto a etapa de realização é responsável pela execução dos testes e pela documentação dos resultados obtidos. Realizar tais atividades é o ideal nos processos de teste, porém não é o que geralmente ocorre.

Esta seção da um enfoque na etapa de preparação dos testes, se propondo a explicar alguns conceitos referentes ao Plano de Teste. Conceitos estes que serão utilizados no decorrer do trabalho.

O Plano de Teste consiste em uma atividade de identificação dos itens a serem testados. Nessa etapa é definido o conjunto de itens a serem testados e então são elaborados os Procedimentos de Teste e seus devidos Casos de Teste.

Para Filho (2012) o Procedimento de Teste consiste em uma descrição breve e em sequencia de todos os passos necessários para executar um caso (ou um grupo de casos) de teste.

Filho (2001) diz que geralmente, cada procedimento de teste corresponde a um roteiro importante de um caso de uso, contendo a sequência de ações a serem executadas, conforme é apresentado no Quadro 1.

Quadro 1 – Exemplo de um Procedimento de Teste

objetivo	Verificar se a inclusão de um usuário é feita corretamente no Merci.
Fluxo	1. Abrir interface **Tela de Usuários.** 2. Acionar Novo. 3. Inserir Nome, Login, Senha. 4. Acionar Salvar. 5. Inserir Login. 6. Acionar Pesquisar

Fonte: (FILHO, 2001)

Já o Caso de Teste consiste em descrever uma condição particular a ser testada, definindo os valores de entrada, as restrições para a sua execução e o resultado esperado.

Segundo Filho (2001), os Casos de Teste tem como objetivo detectar defeitos ainda não descobertos e devem cobrir tanto as entradas válidas como as entradas inválidas. O Quadro 2 apresenta um exemplo de Caso de Teste.

Quadro 2 – Exemplo de um Caso de Teste

Entradas	Campo	Valor
	Nome	Caixeiro_01
	Login	Caixeiro_01
	Senha	www
Saídas Esperadas	Campo	Valor
	Nome	Caixeiro_01
	Grupos do Usuário	Caixeiro

Fonte: (FILHO, 2001)

2.4 Automatização de Teste de Software

Molinari (2010) diz que há uma visível diferença entre os testes de software e a automação desses testes. Para ele, teste de software se resume a ação de testar determinado software. Já a automatização de testes, segundo Kalowa e Huizinga citado por Neto (2010), consiste na utilização de alguma ferramenta ou apoio computacional para controlar a execução dos testes, a comparação dos resultados e comportamentos obtidos com a execução dos testes em relação ao que era esperado, a configuração das pré-condições dos testes entre outras atividades de testes e seus resultados.

Normalmente, tal processo de automatização se resume a automatizar um processo manual já estabelecido em uma empresa ou organização que utiliza um processo de testes formalizado.

Existem algumas ferramentas que se propõem a executar automaticamente os mais diversos tipos de teste de software. Dentre essas ferramentas é destacado o Selenium IDE e o Selenium WebDriver, que são ferramentas utilizadas para a automatização de teste funcional. Um conhecimento superficial do funcionamento de ambas as ferramentas se faz necessário para um bom entendimento do que foi realizado no presente trabalho.

2.5 Selenium IDE

Selenium IDE é uma ferramenta gráfica utilizada para desenvolver casos de teste automatizados. Tal ferramenta tem como objetivo realizar a automatização de diversos cenários de teste funcional.

A ferramenta é implementada como uma extensão do Firefox e permite que usuários possam rapidamente gravar, editar e reproduzir os testes no ambiente real que será executado.

Campos (2011) afirma que além de automatizar os testes funcionais, a ferramenta também contribui com a realização do teste de regressão, visto que a qualquer momento pode-se realizar o mesmo teste nas novas versões do sistema.

Segundo Selenium (2017) que é a documentação presente no site oficial, a ferramenta é um plug-in do Firefox fácil de instalar e utilizar, sendo uma maneira eficiente de desenvolver

casos de teste.

Por padrão, o script desenvolvido no caso de teste é o HTML, embora possa ser traduzido para uma linguagem de programação, como Java e C, ou uma linguagem de script como Python. Durante a execução dos testes, o Selenium IDE traduz as palavras-chaves em ações para interagir com os elementos da interface gráfica do website.

2.6 Selenium WebDriver

Selenium WebDriver é um *framework open source* (código aberto), que é utilizado na automatização de testes funcionais em aplicações web (SELENIUM, 2017). Ao utilizar o Selenium WebDriver, os testes podem ser executados em diferentes navegadores (Firefox, IE, Opera e Chrome) e podem ser escritos em diversas linguagens de programação, tais como: Java, C, Python, PHP, entre outras.

Assim como acontece com o Selenium IDE, a ferramenta Selenium WebDriver tem como objetivo possibilitar aos usuários realizar a automatização de teste funcional.

De forma objetiva, Heinen (2014) descreve o Selenium WebDriver como sendo um conjunto de bibliotecas que são utilizadas em várias linguagens de programação e que servem para integrar o código-fonte com a tela do sistema.

Heinen (2014) também afirma que, diferente do IDE, o WebDriver possibilita a interação com banco de dados, criar *page objects* (padrões de projeto para organização de testes funcionais) e, através de *drivers* externos, fazer a execução em outros navegadores.

O Selenium WebDriver disponibiliza uma série de comandos que são utilizados para o desenvolvimento dos testes. No Quadro 3 são apresentados alguns desses comandos, assim como a explicação de cada um deles.

Quadro 3 – Comandos do Selenium Webdriver

Comando	Descrição
Elemento.clear();	Possibilita limpar o conteúdo de determinado elemento.
Elemento.click();	Possibilita simular o click em determinado elemento.
close();	Possibilita fechar a janela atual.
equals();	Possibilita comparar se o conteúdo de dois objetos são iguais.
findElement();	Possibilita buscar determinado elemento da página por className, cssColector, id, linkText, name, partialLinkText, tagName ou xpath.
findElements();	Possibilita buscar uma lista de elementos da página.
Elemento.getAttribute();	Possibilita retornar o valor do atributo passado como parâmetro de determinado elemento.
Elemento.getClass();	Possibilita obter a classe a qual determinado objeto pertence.
getCurrentUrl();	Possibilita obter a URL da página atual.
Elemento.getText();	Possibilita obter o texto presente em determinado elemento.
getTitle();	Possibilita obter o título da página atual.
Elemento.isDisplayed();	Possibilita verificar se determinado elemento está visível na tela.
Elemento.isEnabled();	Possibilita verificar se determinado elemento está ativo na tela.
Elemento.isSelected();	Possibilita verificar se determinado elemento está selecionado.
quit();	Possibilita parar a execução do teste, fechando a janela.
Elemento.sendKeys();	Possibilita inserir dados em determinado elemento da tela.
setSize();	Possibilita definir o tamanho da janela.
Elemento.submit();	Possibilita enviar dados para o servidor, se o elemento em questão for um formulário.
wait();	Possibilita esperar por um elemento estar visível ou presente na tela.

Fonte: Elaborado pelo autor

Tais comandos são responsáveis por realizar a interação do script de teste do Selenium WebDriver com a página a ser testada. Sem a utilização dos mesmos, os testes não poderiam ser realizados.

Segundo Selenium (2017), caso o usuário necessite de realizar testes de forma remota ou em máquinas virtuais é necessário a utilização do Selenium Server, visto que tais procedimentos não podem ser realizados utilizando apenas o Selenium WebDriver.

O download da ferramenta está disponível na página oficial da mesma e a sua utilização se fez possível por meio da integração do Selenium WebDriver com o Eclipse (IDE para desenvolvimento Java). Para tal, é realizado o download das bibliotecas necessárias para a utilização do Selenium e a importação das mesmas no eclipse, por meio da adição dos arquivos .JARs baixados.

3 Estado da Arte

Nesse capítulo são apresentados projetos já desenvolvidos, que possuem objetivos semelhantes aos desenvolvidos no projeto em questão, mas que se diferem em determinados requisitos.

As pesquisas bibliográficas foram realizadas nas bases de consultas que são referências em repositórios de trabalhos nas áreas de ciências exatas e de tecnologia. As bases pesquisadas foram: ACM Digital Library, IEEE Xplore Digital Library e Science Direct. Além disso, a revista técnica Engenharia de Software Magazine e o Google Acadêmico também foram utilizados para realizar as pesquisas.

Com o propósito de filtrar os artigos, obtendo assim uma melhor qualidade no resultado, foram adotados os seguintes critérios de busca:

- Artigos publicados nos últimos cinco anos;
- Artigos escritos no idioma inglês ou português.

Para realizar as pesquisas, foram adotadas palavras chaves de busca conforme demonstrado no Quadro 4.

Quadro 4 – Palavras Chaves para Pesquisas Bibliográficas

Número	**Palavra Chave**
1	Selenium WebDriver
2	Overview Selenium WebDriver
3	Functional Test with Selenium WebDriver
4	Depth Study about Selenium WebDriver
5	Automatização de Teste Funcional com Selenium WebDriver

Fonte: Elaborado pelo autor

Nessa seção também consta o Trabalho de Conclusão de Curso desenvolvido pela ex-aluna Jessica Vieira Soares (SOARES, 2017) no CEFET-MG campus Timóteo, visto que o mesmo é tido como base para o presente trabalho.

3.1 Ferramenta de Suporte a Automatização de Teste Funcional: Levantamento Bibliográfico

“Ferramenta de Suporte a Automatização de Teste Funcional: Levantamento Bibliográfico” é o trabalho desenvolvido pela aluna Jessica Vieira Soares - Faculdade CEFET-MG (campus Timóteo), em 2017 - como requisito parcial para a obtenção do título de Engenheira de Computação.

Soares (2017) durante seu trabalho faz um levantamento bibliográfico de diferentes sistemas utilizados para automatizar e/ou dar suporte na automatização de teste funcional.

Em seu trabalho, a autora separa suas buscas em dois subgrupos, sendo o primeiro para relatar sobre as técnicas e ferramentas de geração de casos de teste encontradas e o segundo para relatar sobre o uso de ferramentas para a execução de teste funcional.

No primeiro subgrupo, Soares (2017) apresenta as seguintes técnicas e ferramentas que são utilizadas para a geração de casos de teste:

- AutoBlackTest
- EEOCP
- Framework para geração de dados de teste a partir da especificação de negócios
- LBTest
- MoMuT::UML
- NNBBT
- TAXI
- Teste baseado em modelo para aplicações WEB

No segundo subgrupo relatado, são apresentadas as seguintes ferramentas que são utilizadas para a execução de teste funcional:

- Abbot Framework
- Marathon
- Mobile Test
- QF-Test
- Selenium IDE
- SoapUI
- TestLink

A partir dos resultados obtidos por seus estudos, observou-se a existência de uma documentação superficial, com poucos documentos técnico-científicos voltados para a área de teste de software, não possuindo informações suficientes para serem aplicadas em um ambiente complexo de teste.

3.2 Avaliação de Qualidade no Sistema de Locações da Faculdade de Balsas Através de Testes Funcionais Automatizados

O trabalho “Avaliação de Qualidade no Sistema de Locações da Faculdade de Balsas Através de Testes Funcionais Automatizados” foi desenvolvido pelo aluno Israel de Oliveira Barbosa - Faculdade de Balsas – MA.

O trabalho de Barbosa (2013) descreve um estudo realizado na Faculdade de Balsas, onde se utilizou a ferramenta TesLink para gerir e planejar os testes e a ferramenta Selenium IDE para a execução de tais testes, como forma de viabilizar o processo de teste bem como possibilitar seu reaproveitamento em trabalhos futuros.

O autor utilizou de técnicas de teste funcional, auxiliado pelos testes automatizados realizados com o Selenium IDE, com o intuito de avaliar as funcionalidades do Sistema de Locações da Faculdade de Balsas.

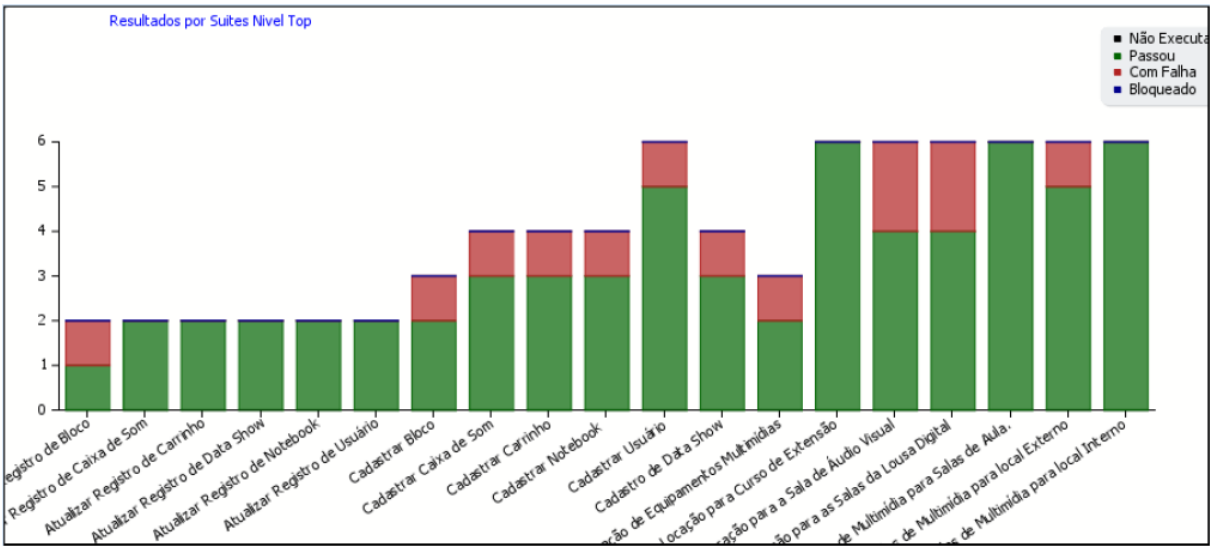
Os testes elaborados por Barbosa (2013) em seu trabalho, foram realizados com o intuito de avaliar o sistema SILMA (Sistema de Locações Multimídias) da faculdade. Para a execução dos testes, o autor utilizou o teste de requisitos, teste de tratamento de erro e teste de interface a fim de verificar o pleno funcionamento do sistema.

No decorrer do trabalho, o autor aborda superficialmente a ferramenta utilizada para realizar a automatização dos testes elaborados, não apresentando detalhadamente os comandos, possibilidades e limitações das mesmas.

A execução dos testes foi realizada utilizando o tipo de usuário “Visão Geral”, que possui acesso a todas as funcionalidades do sistema. Para testar as funcionalidades do sistema, o mesmo foi submetido a diversos testes de cadastro, atualização e cancelamento de locações.

A Figura 1 demonstra a quantidade de erros encontrados pelo autor com a execução de cada Suíte de Teste.

Figura 1 – Resultados dos testes por suíte



Fonte: (BARBOSA, 2013)

Com os resultados obtidos, o autor identificou falhas que comprometem o correto funcionamento do sistema, assim como alguns quesitos que podem ser melhorados com relação à interface e conteúdo. Também foram detectados problemas que não comprometem as funcionalidades do sistema, mas que podem de certa forma afetar as regras de negócio.

3.3 APOGEN: automatic page object generator for web testing

O artigo desenvolvido por Stocco et al. (2017) apresenta a ferramenta Apogen, que realiza automaticamente uma engenharia reversa em páginas de aplicações web, gerando objetos de página Java para Selenium WebDriver.

A ferramenta basicamente combina clustering e análise estática para identificar abstrações significativas nas páginas de aplicações web que são automaticamente transformadas em objetos de página Java.

Segundo Stocco et al. (2017) o desenvolvimento manual de objetos de página é caro e espera-se que ele aumente com o tamanho do sistema desenvolvido. Além disso, para estes autores, a criação manual de objetos de página inclui muitas tarefas repetitivas e cansativas.

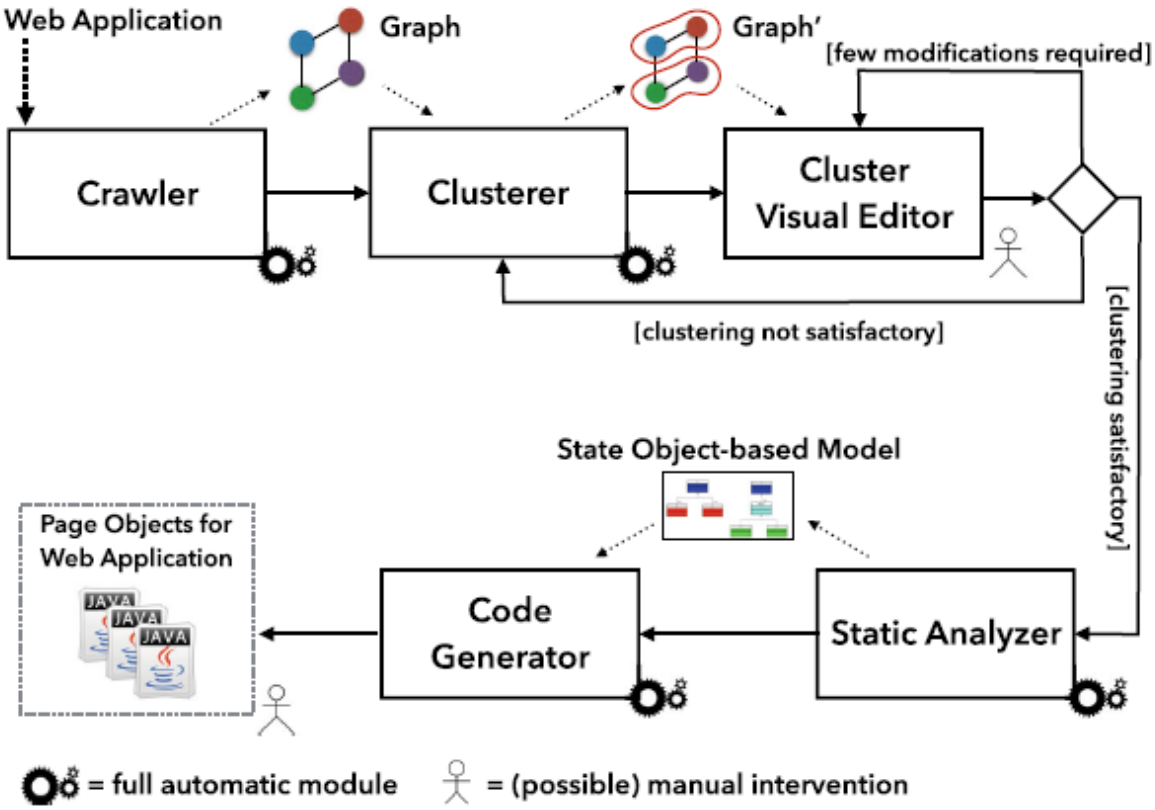
Com isso, a ferramenta APOGEN visa ajudar o testador a poupar tempo na criação de conjuntos de teste para aplicações web, automatizando a criação de uma quantidade considerável de código que, manualmente, seria escrito de outra forma.

A abordagem desenvolvida por Stocco et al. (2017) para a geração automática de objetos de página consiste em quatro etapas. Primeiro, é realizado uma engenharia reversa por meio de um rastreador baseado em eventos. Em seguida, páginas da Web semelhantes são agrupadas em grupos sintáticos e semanticamente significativos. O modelo baseado em eventos (Gráfico) e as informações adicionais (por exemplo, DOMs e clusters) são estaticamente

analisados para gerar um modelo baseado em objeto de estado. Por fim, esse modelo é transformado em significativos objetos de página Java, via transformação de modelo em texto. A seguir, é detalhado cada passo do sistema utilizado como estudo de caso.

A ferramenta APOGEN implementa a abordagem descrita anteriormente. Conforme é apresentado na Figura 2, a ferramenta consiste de cinco módulos principais: um Crawler, um Clusterer, um Editor Visual de Cluster, um Analisador Estático e um Gerador de Código.

Figura 2 – Abordagem do APOGEN para criação de objetos de páginas da Web



Fonte: (STOCCO et al., 2017)

A entrada de APOGEN é qualquer aplicativo da web, juntamente com os dados de entrada necessários para o login e a navegação de formulários. A saída é um conjunto de objetos de página Java, organizados usando o padrão de design Page Factory11, como suportado pela estrutura Selenium Web-Driver.

No artigo é realizada uma avaliação do nível de automatização do Apogen em relação a objetos de página gerados manualmente. Para realizar tal comparação, foi definido manualmente um “padrão ouro”, usado como referência para a geração manual de objetos de página.

Segundo os autores, a avaliação da ferramenta APOGEN em um aplicativo da Web de código aberto demonstrou que a abordagem utilizada pela ferramenta se mostra bastante promissora, onde os objeto de página gerados automaticamente cobriram a maioria das funcionalidades do aplicativo, resultando em um código mais legível.

4 Materiais e Métodos

Este trabalho consiste, segundo a definição de Vergara (2000), em uma pesquisa exploratória quanto aos fins, realizada em bases científicas e revistas técnicas e por meio da exploração da ferramenta utilizada. Quanto aos meios se trata de uma pesquisa experimental.

Como etapa inicial do trabalho, foi realizada a leitura da monografia “Ferramenta de Suporte a Automatização de Teste Funcional: Levantamento Bibliográfico” desenvolvido por Soares (2017). Tal monografia serviu como base para um maior aprofundamento sobre o assunto.

Logo após, realizando a leitura de diferentes artigos, cujos autores apresentavam diferentes abordagens acerca do teste funcional, definiu-se como base para o trabalho os conceitos apresentados por Bastos et al. (2012). Segundo esses autores, os tipos de testes que atendem as técnicas de teste funcional são: “testes de requisitos”, “testes de regressão”, “testes de tratamento de erros”, “testes de suporte manual”, “testes de interconexão com outros softwares”, “testes de controle” e “testes paralelos”, cujas definições se encontram na seção 2.2.

Passou-se então para a escolha da ferramenta a ser explorada, dentre as diversas citadas por Soares (2017) em seu trabalho. Para tal escolha, foram utilizados critérios de seleção que são apresentados na seção 5 deste trabalho, referente à “Análise e Discussão dos Resultados”.

Sendo escolhida a ferramenta dentre as citadas por Soares, passou-se para a etapa de pesquisa de artigos e monografias que também utilizavam a ferramenta escolhida para ser explorada no presente trabalho, mas que se diferenciavam em determinados aspectos. A pesquisa de tais artigos e monografias foi realizada em bases científicas e revistas técnicas. Os critérios de busca, assim como os resultados obtidos, são apresentados na seção 3 do presente trabalho.

Com a escolha da ferramenta para ser utilizada no presente trabalho, realizou-se a exploração da mesma, com o intuito de verificar quais tipos de teste referente à técnica de teste funcional a ferramenta conseguia abordar. Em um primeiro momento, foi realizado um estudo teórico da ferramenta, por meio da leitura de tutoriais, artigos, assim como da documentação presente na página oficial da ferramenta, para um maior entendimento do funcionamento da mesma.

Logo após a instalação, foram desenvolvidos diferentes casos de teste, para determinar na prática quais tipos de teste funcional a ferramenta é capaz de cobrir.

Realizou-se então a escolha dos sistemas a serem utilizados para a execução de cada um dos cenários de testes. Optou-se pela escolha de sistemas que possuíssem funcionalidades capazes de abordar todos os cenários de teste necessários para a exploração dos recursos da ferramenta escolhida.

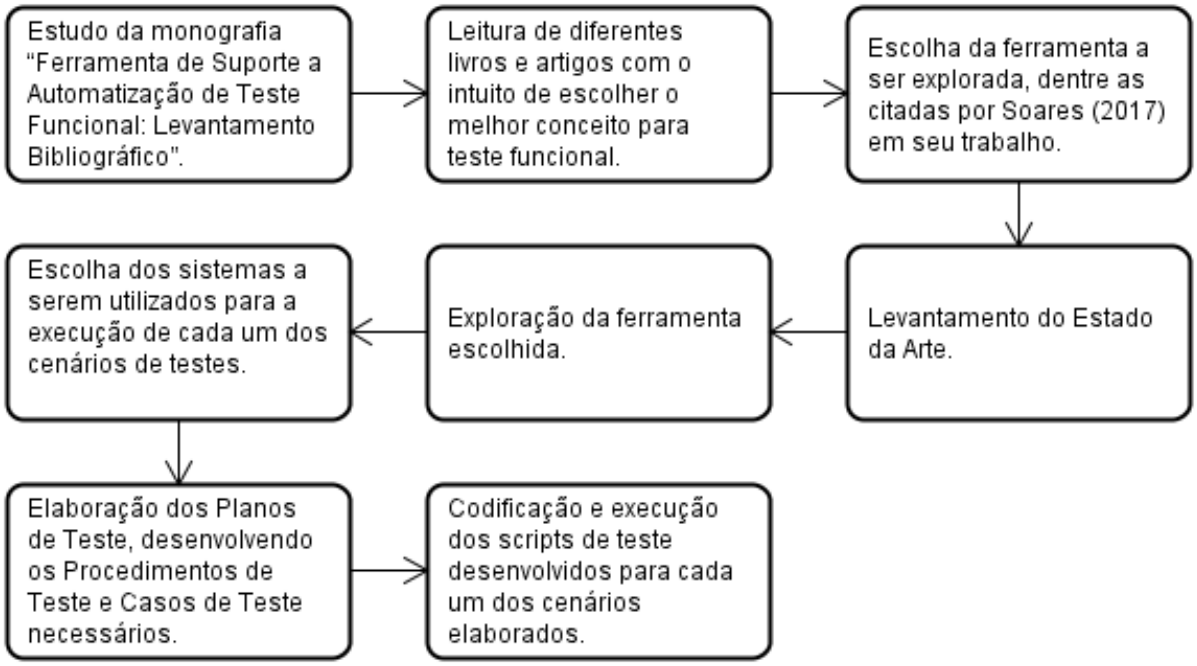
Com a definição dos sistemas a serem testados, pôde-se elaborar todos os planos de teste, desenvolvendo os procedimentos de teste necessários, assim como seus respectivos casos de teste. Nessa etapa, definiram-se, para cada um dos sistemas escolhidos, quais seriam as entradas de dados, assim como as saídas esperadas.

Após serem elaborados todos os procedimentos de teste e todos os casos de teste, iniciou-se a etapa de codificação e execução dos scripts de teste desenvolvidos para cada um dos cenários elaborados.

Todos os procedimentos de teste com seus respectivos casos de testes elaborados para cada um dos cenários de teste planejados são apresentados na seção 5 deste trabalho. Já os scripts de teste desenvolvidos são apresentados do Apêndice A ao Apêndice F.

A Figura 3 apresenta um diagrama de atividades que contem a sequência de todos os passos descritos anteriormente, que foram tomados para o desenvolvimento do presente trabalho.

Figura 3 – Métodos



Fonte: Elaborado pelo autor

Para desenvolvimento do presente trabalho foram utilizados os seguintes materiais:

- Eclipse: Como ferramenta para desenvolvimento e execução dos scripts de teste, integrado ao Selenium WebDriver, foi utilizado a versão Neon.3 do Eclipse IDE.
- Mozilla GeckoDriver: Driver externo utilizado para a execução do Selenium WebDriver no navegador Mozilla Firefox. Para o desenvolvimento do presente trabalho foi utilizado a versão 0.18.0 do Mozilla GeckoDriver.

- Selenium IDE: Como apoio para o desenvolvimento de alguns cenários de teste, foi utilizado a versão 2.9.1.1 do plug-in Selenium IDE.
- Selenium WebDriver: Para desenvolvimento e execução dos scripts de teste, referente a cada um dos cenários de teste, foi utilizado a versão 3.5.2 do Selenium WebDriver

5 Análise e Discussão dos Resultados

Esta seção apresenta os resultados obtidos após a exploração da ferramenta Selenium WebDriver em diferentes cenários que abordam a automatização de Teste Funcional, assim como vantagens e limitações encontradas com a utilização da ferramenta fora dos cenários apresentados.

Tendo a monografia desenvolvida por Soares (2017) como base, passou-se para a escolha da ferramenta a ser utilizada no trabalho. Tal monografia separa os resultados obtidos em duas seções, sendo uma para relatar sobre as técnicas e ferramentas de geração de casos de teste enquanto a outra relata o uso de ferramentas para a execução de teste funcional.

Conforme é apresentado no Quadro 5, definiu-se alguns critérios para nortear a escolha da ferramenta a ser analisada pelo presente trabalho, dentre as diversas ferramentas abordadas por Soares (2017).

Quadro 5 – Critérios de seleção

Ordem	**Critério de Seleção**
1	Escolha das ferramentas para execução de teste funcional (descartando as técnicas e ferramentas para geração de casos de teste).
2	Ferramenta disponível para download
3	Ferramenta possui documentação disponível
4	Ferramenta possui versão completa gratuita
5	Ferramenta possibilita realizar testes automáticos
6	Ferramenta é de fácil instalação

Fonte: Elaborado pelo autor

O Quadro 6 apresenta as ferramentas para a execução de teste funcional abordadas por Soares (2017) em seu trabalho, assim como os critérios atendidos por cada uma delas referente ao que foi abordado no Quadro 5.

Quadro 6 – Critérios atendidos

Nome	**Download disponível**	**Documentação**	**Versão completa gratuita**	**Realiza testes automáticos**
Abbot Framework	X	X	X	X
Marathon	X	X		X
MobileTest				
QF-Test	X	X		X
Selenium IDE	X	X	X	X
SoapUI	X	X		X
TestLink	X	X	X	

Fonte: Elaborado pelo autor

Após passar por tal critério de seleção e analisar os critérios atendidos por cada uma das ferramentas, havia a possibilidade de optar pelo Abbot Framework e Selenium IDE para utilização no trabalho a ser desenvolvido. Dentre as duas ferramentas, escolheu-se o Selenium IDE por se tratar de um plug-in do firefox de fácil instalação e utilização.

Após iniciar os estudos sobre o Selenium IDE, descobriu-se que o Selenium WebDriver (Também conhecido como Selenium 2) se tratava de uma versão mais atualizada, que cobre alguns aspectos que não podem ser abordados pelo Selenium IDE. Entre outros aspectos, o Selenium WebDriver permite que os testes sejam executados em diferentes tipos de navegadores e permite a captura do movimento do cursor, enquanto o Selenium IDE suporta apenas o Firefox. A partir disso, decidiu-se pela utilização do Selenium WebDriver para o desenvolvimento do presente trabalho.

Com a escolha da ferramenta, foram realizados testes em diferentes sistemas com o intuito de verificar se o Selenium WebDriver permite abordar os diversos tipos de teste referentes à técnica de teste funcional definidos por Bastos et al. (2012). Tais tipos de teste são abordados na seção 2.2.

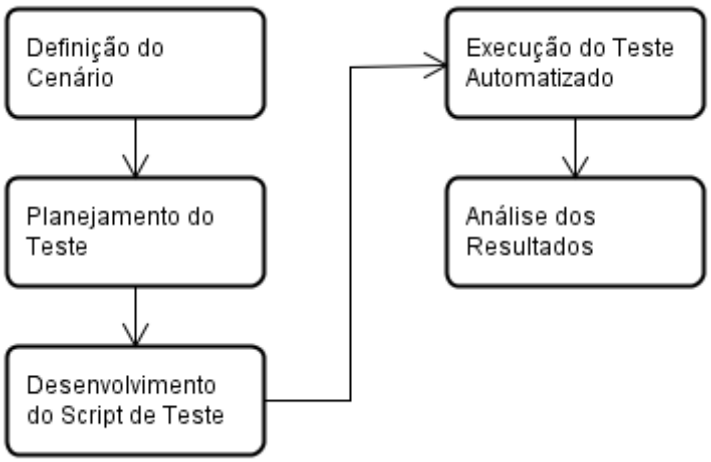
Para cada um dos tipos de testes que atendem a técnica de teste funcional foram elaborados um ou mais cenários de teste, por meio da identificação do que deveria ser testado e de como tal teste deveria ser realizado. Para cada cenário, foram elaborados procedimentos de teste e casos de teste como forma de registro do planejamento realizado.

Não foram criados cenários de teste para os testes de suporte manual e teste de controle, por se entender que tais tipos de testes não podem ser automatizados utilizando um script de teste executado pelo Selenium Webdriver.

Após o planejamento de cada cenário de teste, foram desenvolvidos os devidos scripts de teste. Com a execução de tais scripts, pode-se realizar a análise dos resultados obtidos. Vale ressaltar que para desenvolver o script de teste é necessário inspencionar a página a ser testada, a fim de verificar qual atributo do elemento será utilizado para realizar a busca de tal elemento.

A Figura 4 apresenta um diagrama de atividades com a sequência dos procedimentos adotados para a realização dos testes. Tais procedimentos foram realizados para cada um dos cenários de teste planejados.

Figura 4 – Procedimentos Adotados



Fonte: Elaborado pelo autor

Ao executar um cenário de teste, o teste realizado pode ser executado com sucesso ou resultar em erro ou falha. Segundo Maldonado et al. (2004), erro (*error*) é a diferença entre o valor obtido e o valor esperado, onde qualquer resultado inesperado na execução do teste constitui em erro. Já a falha (*failure*) é a produção de uma saída incorreta em relação ao que foi especificado.

Nas seções 5.1 a 5.7 são apresentados os diferentes testes realizados, assim como a análise dos resultados obtidos.

5.1 Teste de Requisitos

Os testes realizados para contemplar o teste de requisitos visam abordar situações onde a ferramenta é capaz de obter resultados que avaliem se o sistema testado se comporta ou não conforme especificado em seus requisitos. Para tal, foram projetados dois cenários de teste, sendo o primeiro realizado na plataforma do Sympla e o segundo no website do Submarino. Tais testes são detalhados nas seções 5.1.1 e 5.1.2.

5.1.1 Cenário 1 - Teste Automatizado de Execução e Confirmação de um Cadastro

Para a realização desse teste foi escolhido o Sympla (2018), que, de acordo com o próprio site, é uma plataforma online para venda de ingressos e inscrições e gestão de eventos.

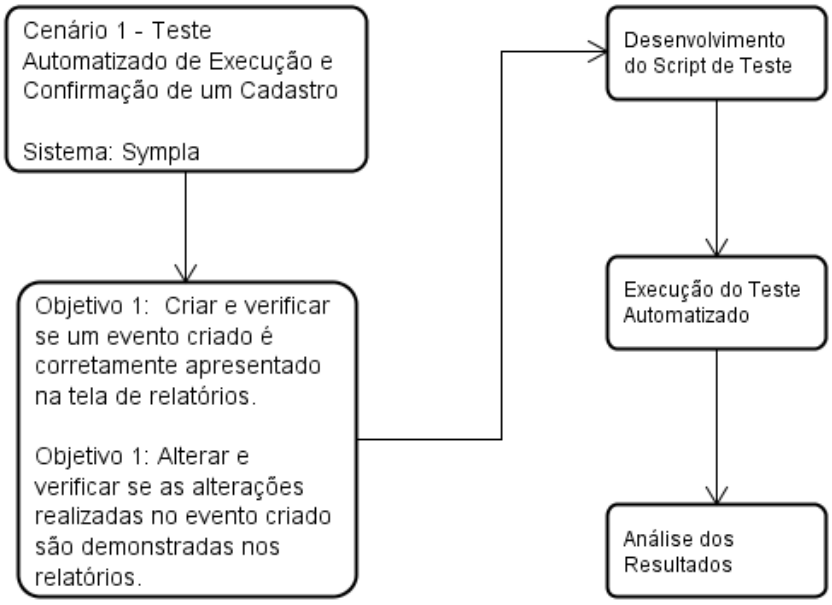
Na plataforma do Sympla o usuário tem a possibilidade de criar um novo evento, assim como editar todos os seus eventos criados. O usuário também tem a possibilidade de pesquisar por eventos criados por outros, podendo inscrever-se nos mesmos.

Nesse cenário, com Selenium WebDriver foi possível testar se os dados inseridos ao criar um novo evento são exibidos corretamente em um relatório na página “Meus eventos”, onde é possível listar todos os eventos criados por determinado usuário.

Antes do desenvolvimento e execução do script de teste utilizado para testar o cenário relatado, planejou-se a execução de tal, sendo criados dois procedimentos de teste, assim como seus respectivos casos de teste. Tais procedimentos e casos de teste tem como intuito definir o fluxo do teste, o valor de cada uma das entradas utilizadas e os valores de saída esperados.

A Figura 5 apresenta a sequência dos procedimentos tomados para o desenvolvimento e execução do Cenário 1, assim como os objetivos que se espera atingir ao se desenvolver tal cenário.

Figura 5 – Procedimentos do Cenário 1



Fonte: Elaborado pelo autor

Com os procedimentos de teste definidos, criou-se um novo evento com dados fictícios e, logo após este ser salvo, foi realizada a verificação se o mesmo aparecia no relatório da página “Meus eventos”. Logo após foi realizado a edição deste evento, verificando se as alterações foram salvas corretamente.

O Quadro 7 apresenta o fluxo adotado para realizar o primeiro procedimento de teste, onde se criou um novo evento e logo após foi verificado a existência do mesmo.

Quadro 7 – Procedimento de Teste 01

objetivo	Criar e verificar se um evento criado é corretamente apresentado na tela de relatórios.
Fluxo	1. Abrir a página do sistema; 2. Realizar Login; 3. Abrir página “Criar Evento”; 4. Adicionar os dados do evento; 5. Salvar o evento; 6. Abrir página “Meus Eventos”; 7. Verificar a existência do evento criado.

Fonte: Elaborado pelo autor

O Quadro 8 apresenta o caso de teste referente ao procedimento de teste 01, onde, para cada passo deste procedimento, o caso de teste apresenta os valores das entradas utilizadas, assim como os valores de saída esperados.

Quadro 8 – Caso de Teste 01

Identificação	Caso de Teste 01	
Itens a Testar	Processamento correto ao criar um evento	
Procedimento de Teste	Referente ao Quadro 7	
Entradas	Campo	Valor
	Login	vitor.comput@gmail.com
	Senha	XXXXX
	Nome do Evento	Simpósio em Engenharia de Computação
	Inicio do evento	25/06/2018
	Horas	10:30
	Término do evento	29/06/2018
	Horas	17:30
	Onde	Evento online
	Organizador (anfitrião)	Vitor
	Tipo de ingresso	Ingresso Único
	Preço	10,00
	Quantidade	100
	Início das vendas	02/06/2018
	Horas	05:30
	Término das vendas	25/06/2018
	Horas	19:00
Saídas Esperadas	Nome do Evento	Simpósio em Engenharia de Computação
Dependências	Possuir cadastro no Sympla.	

Fonte: Elaborado pelo autor

Com o segundo procedimento de teste, realizou-se as devidas alterações no evento criado anteriormente, e logo após, foi verificado se tais alterações eram apresentadas corretamente no relatório em “Meus Eventos”. O fluxo adotado para tal procedimento é detalhado no Quadro 9.

Quadro 9 – Procedimento de Teste 02

objetivo	Alterar e verificar se as alterações realizadas no evento criado são demonstradas nos relatórios.
Fluxo	1. Abrir página “Meus Eventos”; 2. Localizar o evento criado através do Procedimento de Teste 01; 3. Abrir o evento; 4. Editar o evento; 5. Alterar dados do evento; 6. Salvar as alterações do evento; 7. Abrir página “Meus Eventos”; 8. Verificar as alterações realizadas no evento.

Fonte: Elaborado pelo autor

O Quadro 10 apresenta o caso de teste com os dados de entrada referentes ao procedimento de teste 02, assim como a saída esperada.

Quadro 10 – Caso de Teste 02

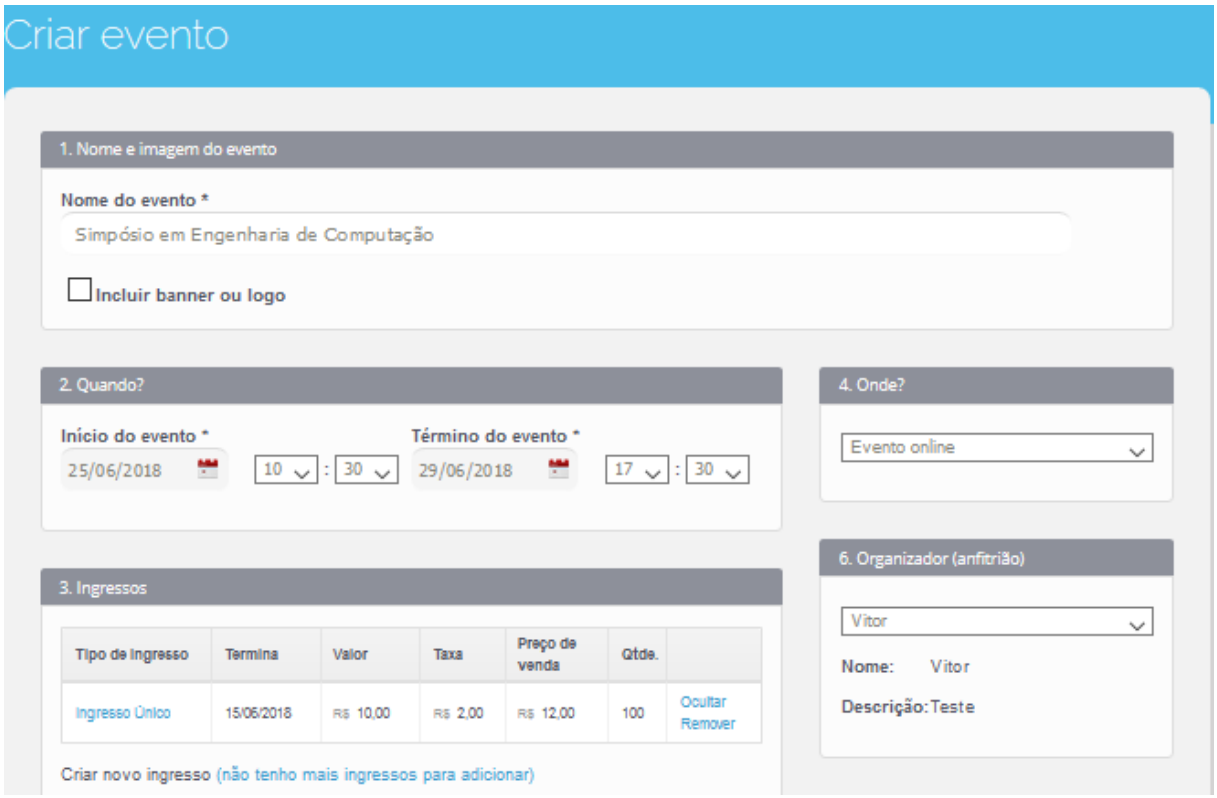
Identificação	Caso de Teste 02	
Itens a Testar	Processamento correto ao criar um evento	
Procedimento de Teste	Referente ao Quadro 9	
Entradas	Campo	Valor
	Nome do evento	Congresso Nacional de Engenharia de Computação
Saídas Esperadas	Nome do Evento	Congresso Nacional de Engenharia de Computação
Dependências	Possuir cadastro no Sympla. Ter concluído o Caso de Teste apresentado no Quadro 8.	

Fonte: Elaborado pelo autor

Após o planejamento e elaboração dos procedimentos de teste e casos de teste descritos acima, passou-se para a implementação de um script no Selenium WebDriver que atendesse ao que foi planejado.

Na primeira etapa do teste, a ferramenta Selenium WebDriver se comportou conforme o esperado, sendo possível criar automaticamente um novo evento conforme é apresentado na Figura 6.

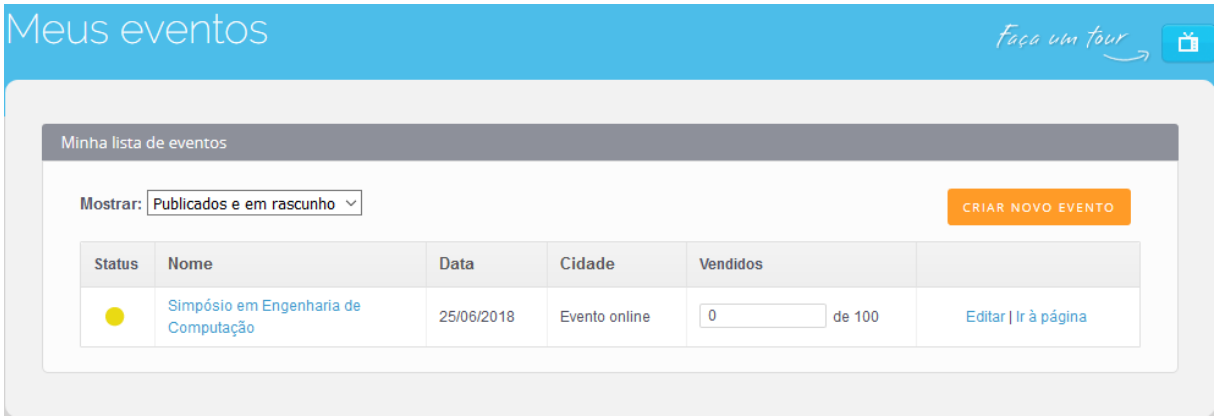
Figura 6 – Criando novo evento



Fonte: Elaborado pelo autor

Com o evento criado e salvo, pôde-se verificar a existência do mesmo na página “Meus eventos” conforme é apresentado na Figura 7.

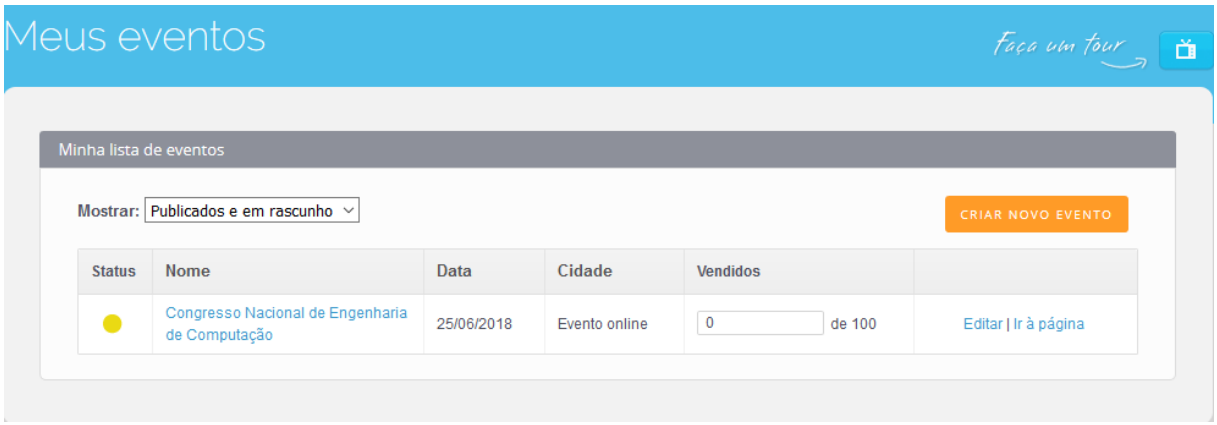
Figura 7 – Verificando a existência do evento criado



Fonte: Elaborado pelo autor

A Figura 8 mostra que, após o evento ser encontrado na página “Meus eventos” o mesmo foi editado e as alterações realizadas foram apresentadas corretamente.

Figura 8 – Verificando as alterações no evento



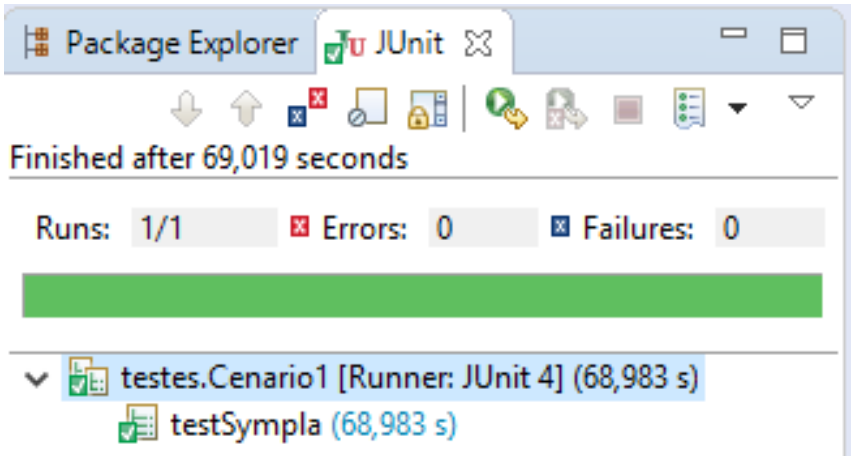
Fonte: Elaborado pelo autor

Todas as etapas de teste descritas nos procedimentos de teste 01 e 02 foram realizadas utilizando-se apenas um único script de teste. O script desenvolvido para tal e apresentado no Apêndice A.

Esse tipo de teste possibilitou testar o correto funcionamento de parte do sistema, verificando que os dados inseridos no cadastro do evento eram exibidos corretamente e que as alterações realizadas no mesmo eram atualizadas no relatório contido em “Meus eventos”.

Não foram encontradas dificuldades para desenvolver e executar o script de teste em questão. O Selenium WebDriver conseguiu realizar todos os procedimentos propostos para a realização das duas partes do teste, executando todo o script com sucesso conforme Figura 9.

Figura 9 – Cenário 1 executado com sucesso



Fonte: Elaborado pelo autor

Vale ressaltar a importância de verificar e alterar os dados de entrada dos campos referente a data sempre que for necessário reexecutar o script de teste desenvolvido. Caso as devidas alterações não sejam realizadas, a execução do teste pode ocasionar em falha. Tal

problema será melhor explicado na seção 5.2.1.

5.1.2 Cenário 2 - Teste Automatizado em um Carrinho de Compras

A ferramenta Selenium WebDriver também permite a realização de testes para verificar as regras de negócio dos sistemas, como por exemplo a compra online de produtos que são previamente adicionados a um carrinho de compras.

Neste teste foi possível avaliar o correto funcionamento de parte do sistema, ao comparar se a soma do valor de cada produto inserido no carrinho condizia com o valor total gerado pelo site no carrinho.

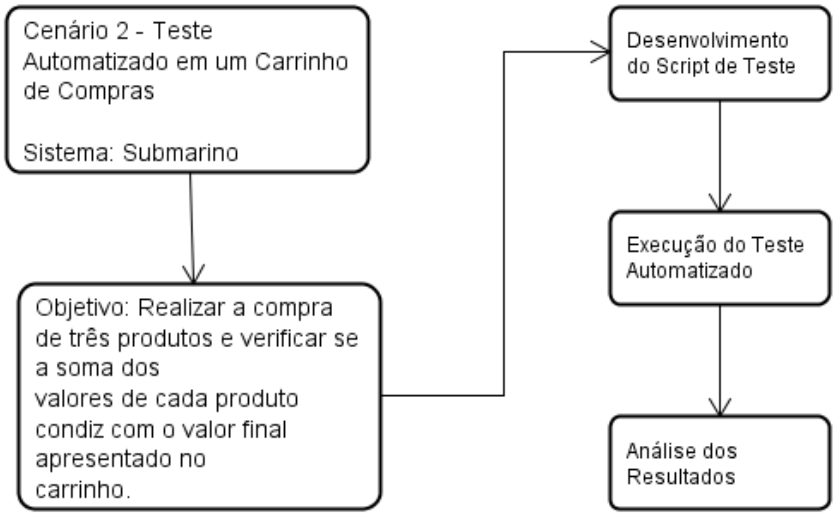
Para realizar o teste em questão, decidiu-se por utilizar o website do Submarino (2018), que é uma empresa brasileira que atua no segmento de comércio eletrônico, entre outros.

O teste consistiu em adicionar automaticamente três produtos ao carrinho, de forma que a ferramenta possibilitou capturar o valor de cada produto escolhido para a compra e atribuir o mesmo a uma variável.

Após todos os produtos serem escolhidos, adicionados ao carrinho e seus respectivos valores serem armazenados, foi realizado a soma dos valores dos mesmos, sendo comparado o resultado obtido com o valor gerado pelo site no campo “Valor Total” do carrinho.

Antes do desenvolvimento e execução do teste, planejou-se a execução do mesmo, onde foi desenvolvido um procedimento de teste com seu respectivo caso de teste. Nessa etapa também se definiu os objetivos a serem alcançados pelo teste. A Figura 10 mostra a sequência dos procedimentos realizados para execução dos testes no Cenário 2.

Figura 10 – Procedimentos do Cenário 2



Fonte: Elaborado pelo autor

Com todas as etapas bem definidas, foi criado um procedimento de teste contendo o fluxo a ser adotado na execução do teste. Tal procedimento é detalhado no Quadro 11.

Quadro 11 – Procedimento de Teste 03

objetivo	Realizar a compra de três produtos e verificar se a soma dos valores de cada produto condiz com o valor final apresentado no carrinho.
Fluxo	1. Abrir a página do sistema; 2. Realizar Login; 3. Para cada um dos produtos escolhidos faça: • Obter o produto a comprar; • Guardar o valor do produto; • Inserir o produto no carrinho de compras. 4. Realizar a soma dos valores dos três produtos; 5. Comparar o valor obtido pela soma com o valor final gerado pelo carrinho de compras.

Fonte: Elaborado pelo autor

O Quadro 12 apresenta o caso de teste referente ao procedimento de teste 03. Tal caso de teste tem como objetivo definir as entradas referentes a cada um dos passos detalhados no procedimento de teste, assim como as saídas esperadas.

Quadro 12 – Caso de Teste 03

Identificação	Caso de Teste 03
Itens a Testar	Processamento correto ao inserir produtos em um carrinho de compras.
Procedimento de Teste	Referente ao Quadro 11 – Itens de 1 a 3
Entradas	“iPhone X Cinza Espacial 256GB”
	“Notebook Gamer Acer Predator”
	“Smart TV LED 75 Samsung”
Saídas Esperadas	Ter os produtos “iPhone X Cinza Espacial 256GB”, “Notebook Gamer Acer Predator” e “Smart TV LED 75 Samsung” inseridos no carrinho.
Dependências	Possuir cadastro no Submarino.

Fonte: Elaborado pelo autor

Após desenvolver o procedimento de teste e o caso de teste referente ao planejamento realizado, foi desenvolvido um script no Selenium WebDriver para a realização do teste.

A Figura 11 apresenta parte do script de teste que verifica se o valor da soma dos produtos escolhidos é igual ao valor gerado pelo site no campo “Valor Total” do carrinho. Todo o script desenvolvido para realizar tal teste se encontra no Apêndice B.

Figura 11 – Parte do script de teste para comparação de valores

```
//Obtendo o valor total

String valorTotal = driver.findElement(By.id("total-amount")).getText();
String auxValorTotal = valorTotal.replaceAll("R$ ", "");
int valorT = Integer.parseInt(auxValorTotal);

//Realizando a soma dos valores dos produtos
int soma = valor1 + valor2 + valor3;

//Comparando o valor da soma com o valor gerado pelo carrinho
if(valorT != soma){
    driver.quit();
}
```

Fonte: Elaborado pelo autor

O teste em questão foi realizado com sucesso, porém, dependendo da forma como é realizada a busca pelos elementos, o script de teste desenvolvido pode resultar em falha caso seja reexecutado posteriormente. Este problema ocorre por se tratar de uma página onde a exibição dos produtos a serem vendidos não é feita de forma estática. Nesse cenário, a alta rotação dos produtos pode comprometer a execução do script. Tal problema é melhor detalhado na seção 5.7.3.

O teste se comportou conforme o esperado, obtendo valores idênticos para o campo "Valor Total" gerado pelo site e para a soma dos valores dos produtos escolhidos para a compra, demonstrando assim um correto funcionamento quanto a esta funcionalidade do sistema.

A ferramenta Selenium WebDriver conseguiu executar todos os procedimentos propostos para a realização do teste em questão, executando todo o script de teste com sucesso.

5.2 Teste de Tratamento de Erros

Outro tipo de teste abordado pelo Teste Funcional é o Teste de Tratamento de Erro. Esse tipo de teste consiste em verificar se as condições de erro previamente conhecidas são reconhecidas e tratadas pelo sistema a ser testado. Para tal, foram realizados os testes descritos nas Seções 5.2.1 e 5.2.2.

5.2.1 Cenário 3 - Teste Automatizado para Verificar o Tratamento de Erro

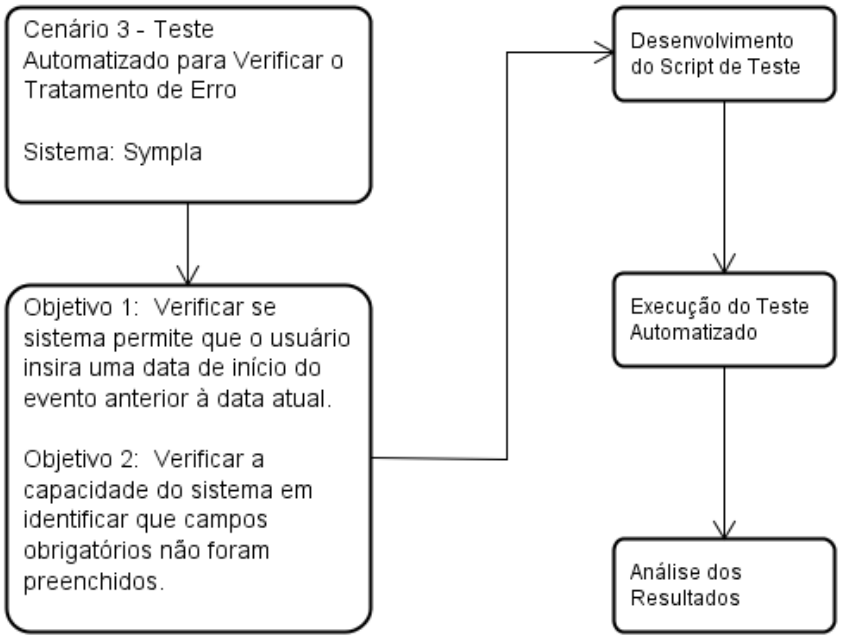
Para a realização do teste em questão, optou-se pela utilização do Sympla (2018) com o intuito de verificar, por meio dos testes automatizados, se o sistema tem capacidade de tratar determinados erros.

Para verificar o tratamento de erro do sistema, foram elaborados dois procedimentos de testes com o intuito de analisar como o sistema se comporta ao ser submetido a diferentes

tipos de erro.

A Figura 12 mostra a sequência dos procedimentos realizados para planejamento e execução do teste referente ao Cenário 3.

Figura 12 – Procedimentos do Cenário 3



Fonte: Elaborado pelo autor

Em um primeiro momento, foi criado um novo evento, onde se tentou introduzir uma data anterior a atual no campo de data de início do evento. Tal procedimento é detalhado no Quadro 13.

Quadro 13 – Procedimento de Teste 04

Objetivo	Verificar a capacidade do sistema em não permitir que o usuário insira uma data de início do evento anterior à data atual.
Fluxo	1. Abrir a página do sistema; 2. Realizar login; 3. Abrir página “Criar Evento”; 4. Adicionar os dados do evento; 5. Tentar adicionar, para início do evento, uma data anterior a atual; 6. Tentar salvar o evento;

Fonte: Elaborado pelo autor

Para a realização do procedimento de teste 04, foram atribuídos os valores de entrada relatados no caso de teste conforme o Quadro 14. O quadro também apresenta as saídas esperadas mediante aos valores de entrada.

Quadro 14 – Caso de Teste 04

Identificação	Caso de Teste 04	
Itens a Testar	Capacidade do sistema em não permitir a criação de evento com data de inicio anterior a atual.	
Procedimento de Teste	Referente ao Quadro 13	
Entradas	Campo	Valor
	Login	vitor.comput@gmail.com
	Senha	XXXXX
	Nome do evento	Simpósio em Engenharia de Computação
	Inicio do evento	“Data anterior à data atual”
Saídas Esperadas	Falha no teste	
Dependências	Possuir cadastro no Sympla.	

Fonte: Elaborado pelo autor

Em um segundo momento, foi elaborado outro procedimento de teste onde, ao criar um novo evento, não foi preenchido os campos obrigatórios, com o intuito de verificar o comportamento do sistema diante de tal situação. Tal procedimento de teste é apresentado no Quadro 15.

Quadro 15 – Procedimento de Teste 05

Objetivo	Verificar a capacidade do sistema em identificar que campos obrigatórios não foram preenchidos; Verificar se o sistema emite mensagem de erro para o usuário.
Fluxo	1. Abrir a página do sistema; 2. Realizar login; 3. Abrir página “Criar Evento”; 4. Tentar salvar o evento sem preencher nenhum campo obrigatório; 5. Verificar se o sistema permite salvar o evento; 6. Verificar se o sistema emite mensagens de erro.

Fonte: Elaborado pelo autor

Para a realização do procedimento de teste 05, foi elaborado o caso de teste referente ao Quadro 16, definindo todos os dados de entrada e as saídas esperadas.

Quadro 16 – Caso de Teste 05

Identificação	Caso de Teste 05	
Itens a Testar	Capacidade do sistema de tratar erros.	
Procedimento de Teste	Referente ao Quadro 15	
Entradas	Campo	Valor
	Login	vitor.comput@gmail.com
	Senha	XXXXX
	Nome do evento	
	Onde	
	Organizador (anfitrião)	
	Tipo de ingresso	
Saídas Esperadas	Mensagem de erro	O campo “Nome” é obrigatório.
	Mensagem de erro	Pelo menos 1 tipo de ingresso deve ser informado.
	Mensagem de erro	O campo “Local” do endereço é obrigatório.
	Mensagem de erro	O campo “Av./Rua” do endereço é obrigatório.
	Mensagem de erro	O campo “Cidade” do endereço é obrigatório.
	Mensagem de erro	O Campo “Estado” do endereço é obrigatório.
Dependências	Possuir cadastro no Sympla.	

Fonte: Elaborado pelo autor

Após a elaboração dos procedimentos de teste e seus respectivos casos de teste, passou-se para a etapa de desenvolvimento e execução do script de teste por meio da ferramenta. Ambos os procedimentos foram realizados utilizando-se apenas um único script de teste, sendo este separado em duas classes. Cada classe é responsável pela execução de um dos procedimentos de teste relatados anteriormente.

A primeira classe, denominada de testSympla1 contem os testes do primeiro procedimento de teste, referente ao Quadro 13. Para este procedimento de teste, o teste realizado pelo Selenium WebDriver resultou em erro no momento em que se tentou acessar uma data anterior à data atual para o início do evento.

Tal erro ocorreu, pois a plataforma do Sympla desabilita os campos referentes às datas anteriores a data atual. Com isso, ao realizar o teste, o Selenium WebDriver tentou simular o clique em um campo desabilitado. Como não foi possível realizar tal ação, o teste resultou em falha.

Esta é uma situação que deve ser analisada sempre que for necessário reexecutar o script de teste, adaptando a data de início do evento para uma data posterior a atual. Caso tal verificação não seja feita, um script de teste que antes era executado com sucesso pode passar a apresentar falha.

A segunda classe, denominada de testSympla2 contem os testes do segundo proce-

dimento de teste, referente ao Quadro 15. Para este procedimento de teste pôde-se realizar a verificação da existência das mensagens de erro por meio do Selenium WebDriver conforme parte do script apresentado na Figura 13. Todo o script referente ao teste realizado é apresentado no Apêndice C.

Figura 13 – Verificação das mensagens de erro

```
// Verificando a existência das mensagens de erro emitidas pelo sistema
driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[1]"));
driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[2]"));
driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[3]"));
driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[4]"));
```

Fonte: Elaborado pelo autor

Para a realização da segunda parte do teste, referente à classe testSympla2, é necessário que se conheça previamente os possível atributos referentes aos elementos que exibem as mensagens de erro na tela, visto que a verificação é realizada buscando-se tais elementos.

Caso também seja necessário verificar se a mensagem de erro está de acordo com o esperado, deve-se saber previamente qual é a mensagem de erro a ser exibida. Para realizar tal verificação, além de se conhecer previamente as mensagens de erros a serem exibidas e os elementos de tela referentes a tais mensagens, é necessário que cada mensagem seja atribuída corretamente a seu elemento correspondente. Caso contrário, o teste resultará em falha. Esta outra possibilidade de verificação se encontra em forma de comentário no script apresentado no Apêndice C.

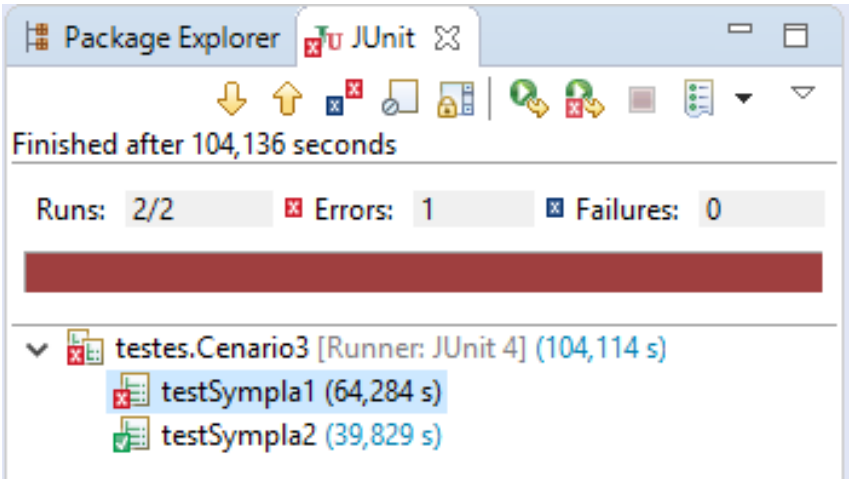
Conforme é apresentado na Figura 14, o sistema se comportou como se esperava durante a execução da segunda parte do script, não permitindo que o evento fosse salvo e exibindo para o usuário as devidas mensagens de erro, referentes aos campos obrigatórios.

Figura 14 – Exibição das mensagens de erro do Sympla

Fonte: Elaborado pelo autor

A Figura 15 apresenta os resultados exibidos pelo Selenium WebDriver após a execução de todo o script de teste, sendo a primeira classe executada com erro e a segunda executada com sucesso.

Figura 15 – Execução do Cenário 3



Fonte: Elaborado pelo autor

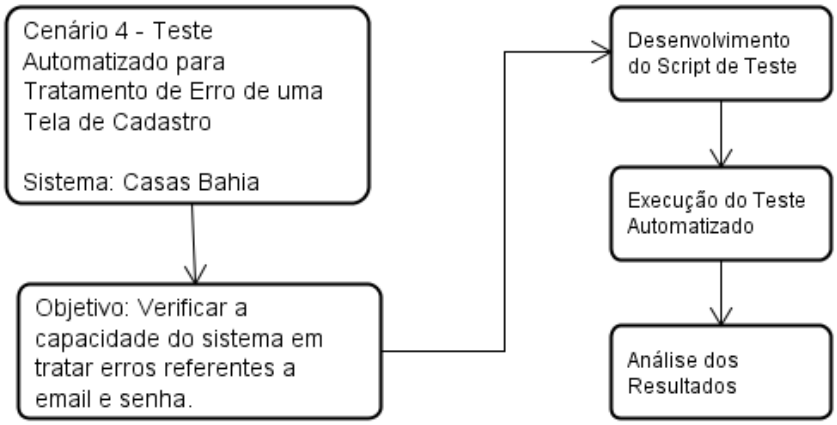
5.2.2 Cenário 4 - Teste Automatizado para Tratamento de Erro de uma Tela de Cadastro

Com a utilização da ferramenta Selenium WebDriver, criou-se um cenário de teste automatizado de uma tela de cadastro de usuário, onde foi possível verificar o comportamento do sistema ao ser submetido a erros.

Para o desenvolvimento desse tipo de teste em específico, foi utilizado o formulário de cadastro de usuários das Casas Bahia, onde foi simulado o cadastro de um novo usuário no sistema. “Casas Bahia é uma rede que comercializa eletrodomésticos, eletroeletrônicos, móveis e utilidades domésticas.” (BAHIA, 2018).

Antes da execução do teste, realizou-se o planejamento do mesmo, onde foi desenvolvido um procedimento de teste e seu respectivo caso de teste, com o intuito de definir quais os passos seriam tomados. A Figura 16 mostra a sequência dos procedimentos realizados para planejamento e execução do teste referente ao Cenário 4.

Figura 16 – Procedimentos do Cenário 4



Fonte: Elaborado pelo autor

Nesse tipo de teste, a ferramenta permite que seja feito a inserção automática de valores pré-definidos aos campos de preenchimento da tela de cadastro de usuário do sistema.

Com os valores atribuídos a seus devidos campos, a ferramenta deu à possibilidade de testar algumas das exigências da página para que fosse concluído o cadastro, conforme abaixo:

- Comparação dos dados de entrada dos campos “e-mail” e “confirmação de e-mail” com o intuito de verificar se os dados inseridos nos dois campos eram iguais;
- Comparação dos dados de entrada dos campos “senha” e “confirmação de senha” com o intuito de verificar se os dados inseridos nos dois campos eram iguais;

O teste também possibilitou avaliar o comportamento do sistema testado quanto ao tratamento de erro, verificando se o mesmo exibe mensagens de erro ao usuário quando os valores introduzidos nos campos “e-mail” e “confirmação de e-mail” são diferentes. O mesmo foi feito para os campos “senha” e “confirmação de senha”, conforme descrito no Quadro 17.

Quadro 17 – Procedimento de Teste 06

objetivo	Verificar a capacidade do sistema em tratar erros referentes a email e senha.
Fluxo	1. Abrir a página do sistema; 2. Abrir a página de cadastro de novo usuário; 3. Inserir dados do cadastro; 4. Inserir dados diferentes nos campos “Email” e “Confirmar email”; 5. Inserir dados diferentes nos campos “Senha” e “Confirmar senha”; 6. Verificar se o sistema emite mensagens de erro.

Fonte: Elaborado pelo autor

Para a realização deste teste, utilizou-se o Selenium WebDriver para capturar os dados do campo “Email” e salvar em uma variável. O mesmo foi feito para o campo “Confirmação de

Email" onde o dado obtido foi salvo em outra variável. Após ter os dois dados salvos em diferentes variáveis, foi possível realizar a comparação dos mesmos. Como os dados eram diferentes, o Selenium WebDriver verificou se a página exibia para o usuário a mensagem de erro "Os endereços informados não são iguais".

O mesmo teste foi feito para os campos de senha. Primeiramente obtiveram-se os dados referentes aos campos "Senha" e "Confirmação de Senha", sendo estes salvos em diferentes variáveis. Logo após, foi possível realizar a comparação entre os dados obtidos e, sendo estes diferentes, verificou se a página exibia para os usuários a mensagem de erro "As senhas informadas não são iguais".

O Quadro 18 apresenta as entradas referentes a cada um dos passos a ser tomado para realização do procedimento de teste apresentado anteriormente, assim como as saídas esperadas.

Quadro 18 – Caso de Teste 06

Identificação	Caso de Teste 06	
Itens a Testar	Capacidade do sistema de tratar erros em uma tela de cadastro no que se diz respeito aos campos email e senha.	
Procedimento de Teste	Referente ao Quadro 17	
Entradas	Campo	Valor
	Email	vitor.comput@gmail.com
	Nome Completo	Vitor Campos e Silva
	CPF	090.411.396-56
	Celular	31 981002154
	Telefone Fixo	31 38231050
	Data de Nascimento	14/06/1992
	Sexo	M
	Confirmação de Email	vitor@gmail.com
	Senha	Senha123
	Confirmação de Senha	Senha
Saídas Esperadas	Mensagem de erro	O campo "Nome" é obrigatório.
	Mensagem de erro	Os endereços informados não são iguais.
	Mensagem de erro	As senhas informadas não são iguais.
Dependências	Possuir cadastro no Sympla.	

Fonte: Elaborado pelo autor

Após o planejamento e elaboração do procedimento de teste e seu respectivo caso de teste, foi elaborado o script com o intuito de realizar o teste descrito. Neste cenário, o teste se comportou conforme esperado. O Selenium WebDriver conseguiu obter os dados de todos os campos necessários, assim como realizar as comparações desejadas.

A Figura 17 apresenta a parte do script onde foi realizada a comparação entre os dados presentes nos campos "Email" e "Confirmação de Email" e a verificação da mensagem de erro caso estes sejam diferentes. O mesmo acontece para os campos "Senha" e "Confirmação de

Senha".

Email1 e Email2 são as variáveis que armazenam os dados obtidos pelos campos "Email" e "Confirmação de Email" respectivamente, assim como Senha1 e Senha2 são as variáveis que armazenam os dados obtidos pelos campos "Senha" e "Confirmação de Senha" respectivamente. Todo o script desenvolvido para a execução do teste se encontra no Apêndice D.

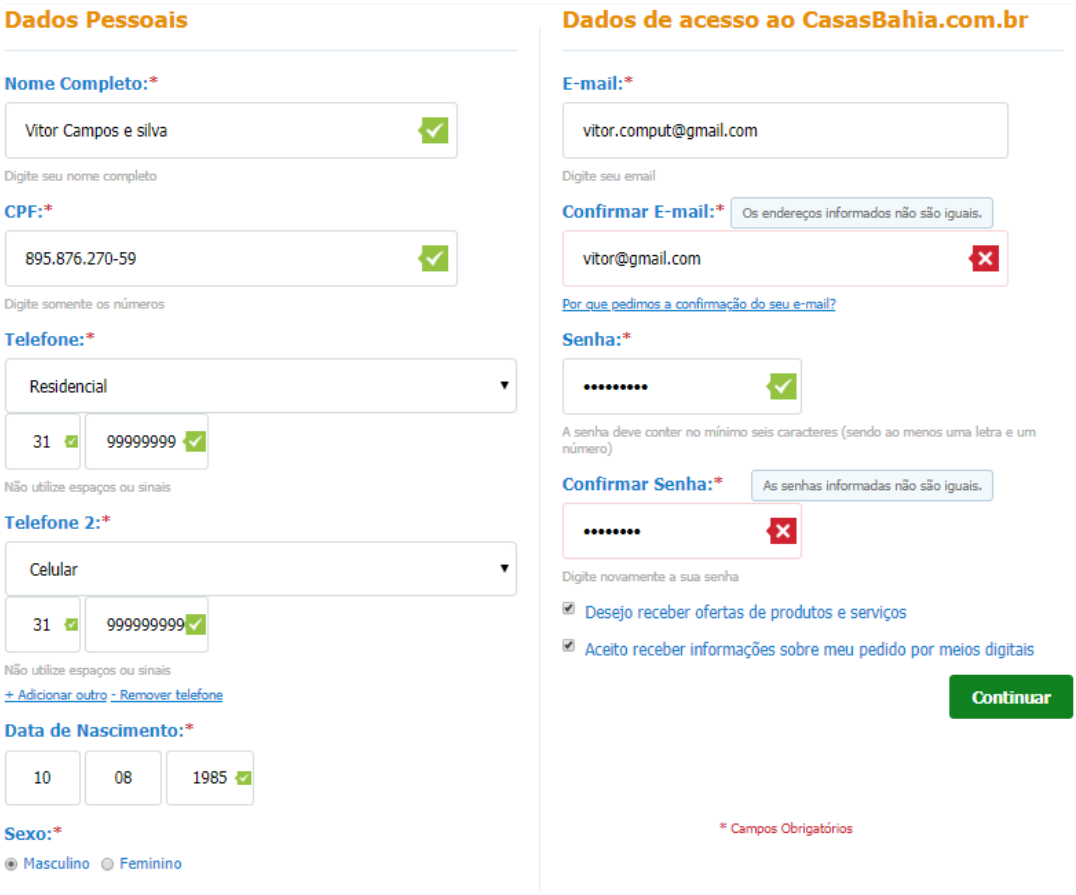
Figura 17 – Comparação dos dados

```
// Verificando se os dados dos campos Email e Confirmar Email são iguais
if (!Email1.equals(Email2)) {

    // Verificando se o sistema exibe a mensagem de erro
    assertEquals("Os endereços informados não são iguais.", driver
            .findElement(By.xpath("//*[@id='cliente_cadastro']/form/fieldset[4]/p[2]/span/span")).getText());

}

// Verificando se os dados dos campos Senha e Confirmar Senha são iguais
if (!Senha1.equals(Senha2)) {

    // Verificando se o sistema exibe a mensagem de erro
    assertEquals("As senhas informadas não são iguais.", driver
            .findElement(By.xpath("//*[@id='cliente_cadastro']/form/fieldset[4]/p[4]/span[1]/span")).getText());

}
```

Fonte: Elaborado pelo autor

Como os dados presentes nas variáveis Email1 e Email2 eram diferentes, verificou-se que o site exibia as devidas mensagens de erro ao usuário. O mesmo aconteceu para os campos Senha1 e Senha2. Tais mensagens são apresentadas na Figura 18.

Figura 18 – Exibição das mensagens de erro das Casas Bahia



Fonte: Elaborado pelo autor

Para a realização deste tipo de teste, foi necessário saber previamente quais eram as mensagens de erro que o sistema exibe para que as mesmas pudessem ser procuradas caso os campos “Email” e “Confirmação de Email” possuíssem valores diferentes. O mesmo aconteceu para os campos “Senha” e “Confirmação de Senha”. Neste cenário, o Selenium WebDriver verificou se o sistema exibe a mensagem de erro corretamente, analisando se o texto de tal mensagem é exibido na tela caso o dado dos campos comparados sejam diferentes.

Há também a possibilidade de realizar tal teste quando não se sabe exatamente qual a mensagem de erro que aparece na tela para o usuário. Para este caso, é necessário verificar se o elemento responsável por exibir a mensagem apareceu na tela, independente da mensagem exibida pelo mesmo.

Todo o script de teste foi executado com sucesso, não sendo apresentados erros ou falhas pela ferramenta Selenium WebDriver para tal execução. Para este cenário, não foi possível identificar uma situação em que o Selenium WebDriver não fosse capaz de realizar os testes ou que o resultado obtido fosse diferente do que se era esperado.

5.3 Teste de Regressão

O Teste de Regressão visa testar, após ser realizada alguma mudança no sistema, segmentos que já foram testados anteriormente. Tais testes foram realizados com o intuito de verificar se funções previamente testadas continuam funcionando corretamente após as alterações realizadas no sistema. Caso as alterações realizadas afetem as funcionalidades já testadas, devem-se realizar as devidas modificações no script de teste, visando atender a nova versão do sistema.

Devem-se classificar também como Teste de Regressão as situações onde as novas alterações do sistema não afetem diretamente as funcionalidades que já foram testadas. Nessas situações, deve-se reexecutar os scripts de teste que já haviam sido elaborados anteriormente.

5.3.1 Cenário 5 – Teste Automatizado no Sistema Clubes Online

Para a realização do teste em questão, decidiu-se pela utilização do sistema Clubes Online. Clubes Online é um sistema web desenvolvido pelo próprio autor e que tem como objetivo possibilitar que clubes disponibilizem a venda online de ingressos a seus usuários.

Tal sistema foi escolhido visto que para a realização desse tipo de teste, tem a necessidade de alteração do código fonte. Por tal motivo, é mais adequado realizar os testes em um sistema no qual se tenha acesso ao código.

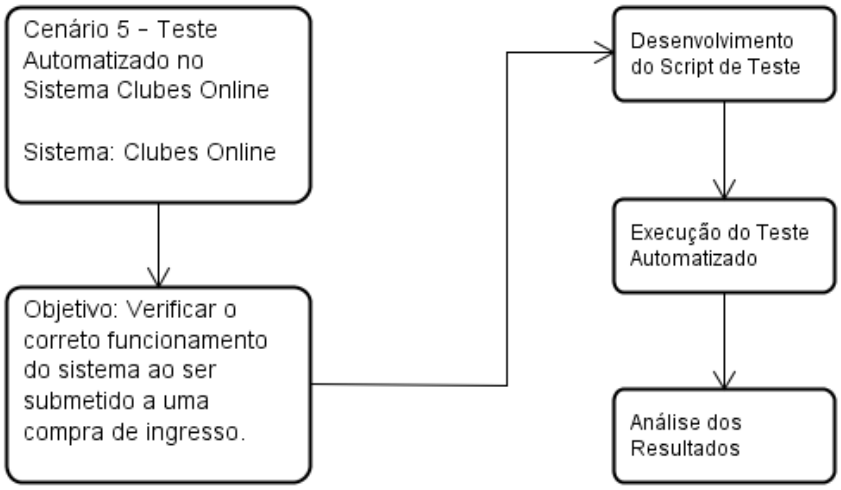
O sistema Clubes Online tem como finalidade modificar o esquema de vendas de ingressos de determinado clube, deixando de vender tais ingressos localmente para realizar a venda online. O usuário poderá retirar por meio do sistema online as cortesias que o mesmo tem disponível, assim como comprar convites quando não possuir mais cortesias.

O teste em questão foi realizado em duas etapas. Na primeira etapa, utilizou-se a ferramenta Selenium WebDriver para realizar automaticamente a compra de um convite.

Na segunda etapa, logo após o teste referente à primeira etapa ser executado com sucesso, realizou-se uma alteração no sistema, introduzindo a nova funcionalidade “Histórico de Convites”, onde são listados todos os convites comprados. Após tal alteração, o mesmo script de teste utilizado na primeira etapa foi executado novamente.

Antes de realizar o teste desenvolvido para o cenário relatado, planejou-se a execução do mesmo, sendo criado um procedimento de teste, assim como seu respectivo caso de teste. A Figura 19 mostra a sequência dos procedimentos realizados para planejamento e execução do teste referente ao Cenário 5.

Figura 19 – Procedimentos do Cenário 5



Fonte: Elaborado pelo autor

Com a definição do cenário, foi elaborado o procedimento de teste com o intuito de definir o objetivo do teste e o fluxo para a realização do mesmo. Tal procedimento é apresentado no Quadro 19.

Quadro 19 – Procedimento de Teste 07

objetivo	Verificar o correto funcionamento do sistema ao ser submetido a uma compra de ingresso.
Fluxo	1. Abrir a página do sistema; 2. Realizar login; 3. Abrir a aba “Comprar Convite”; 4. Realizar a compra de um novo convite.

Fonte: Elaborado pelo autor

O Quadro 20 apresenta o caso de teste com as respectivas entradas referentes a cada um dos passos a ser tomado no procedimento de teste 07. Neste quadro, também são apresentadas as saídas esperadas com a execução do script de teste.

Quadro 20 – Caso de Teste 07

Identificação	Caso de Teste 07	
Itens a Testar	Teste da funcionalidade Comprar Convite com o intuito de verificar se o sistema realiza a compra com sucesso.	
Procedimento de Teste	Referente ao Quadro 19	
Entradas	Campo	Valor
	Email	isabela@gmail.com
	Senha	XXXXX
	Nome	Izabella
	Tipo Documento	CPF
	Documento	XXXXXXXXXXX
	Data da Utilização	30/06/2018
Saídas Esperadas	Nome	Izabella
	Documento	XXXXXXXXXXX
	Data	30/06/2018
	Tipo	Comprado
	Status	Pendente
	Valor	R$ 30,00
Dependências	Possuir cadastro no sistema Clubes Online.	

Fonte: Elaborado pelo autor

Após o planejamento e elaboração do procedimento de teste 07 e seu respectivo caso de teste, foi elaborado o script de teste com o intuito de realizar o teste descrito. Todo o script referente ao teste realizado se encontra no Apêndice E.

A execução do teste ocorreu conforme planejado. A ferramenta Selenium WebDriver conseguiu realizar automaticamente a compra de um ingresso conforme esperado.

Sendo a primeira etapa realizada com sucesso, passou-se para a segunda etapa do teste, onde se implementou uma nova funcionalidade no sistema. “Histórico de Convites” foi desenvolvido com o objetivo de apresentar aos usuários do sistema um relatório apresentando todos os convites retirados e comprados.

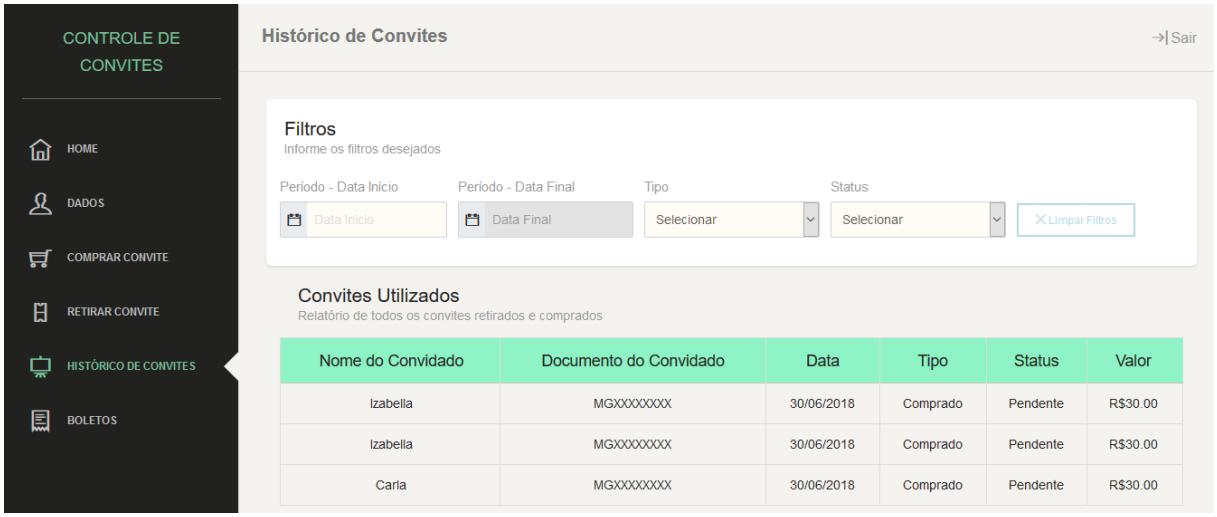
Após a nova funcionalidade ser implementada, realizou-se novamente o teste da primeira etapa referente à compra de um ingresso. Nesse cenário, utilizou-se o mesmo script da primeira etapa, pelo fato de que a nova funcionalidade do sistema não interfere diretamente no teste realizado anteriormente.

Vale ressaltar a importância de verificar se algum dos atributos dos elementos buscados no teste foram alterados juntamente com a nova implementação do sistema. Caso algum atributo utilizado seja alterado, é de extrema importância que se faça as devidas modificações no script de teste, para que o teste não resulte em falha, sendo que o sistema continua a funcionar conforme o esperado. Tal problema é melhor detalhado na seção 5.7.4.

Mesmo com a alteração do sistema, a reexecução do teste foi realizada com sucesso, conforme era esperado. Em cenários onde a alteração do sistema se dá em funcionalidades que não afetem diretamente os testes já realizados, é de se esperar que a reexecução de tais testes, após a alteração, ocorra com sucesso e sem apresentar problemas.

Nesse cenário, o Selenium WebDriver conseguiu, com o mesmo script de teste, efetuar automaticamente, antes e depois da implementação da nova funcionalidade, a compra de um novo convite, conforme é apresentado na Figura 20.

Figura 20 – Teste executado após implementação de nova funcionalidade



Fonte: Elaborado pelo autor

Por meio da Figura 20, pode-se observar que o relatório dos convites utilizados apresenta, nas duas últimas compras realizadas, todos os dados iguais. Isso ocorreu, pois para simular tais compras, foi utilizado o mesmo script de teste, com os mesmos dados de entrada.

Todo o script de teste foi executado corretamente, sem apresentar falha. Com isso, foi possível realizar o que o teste de regressão se propõe a testar.

5.4 Teste de Interconexão com outros Softwares

Com esse tipo de teste, pretende-se garantir a correta interconexão entre diferentes tipos de software. Espera-se poder verificar, por meio do Selenium WebDriver, se os parâmetros e dados são transferidos corretamente entre os sistemas testados.

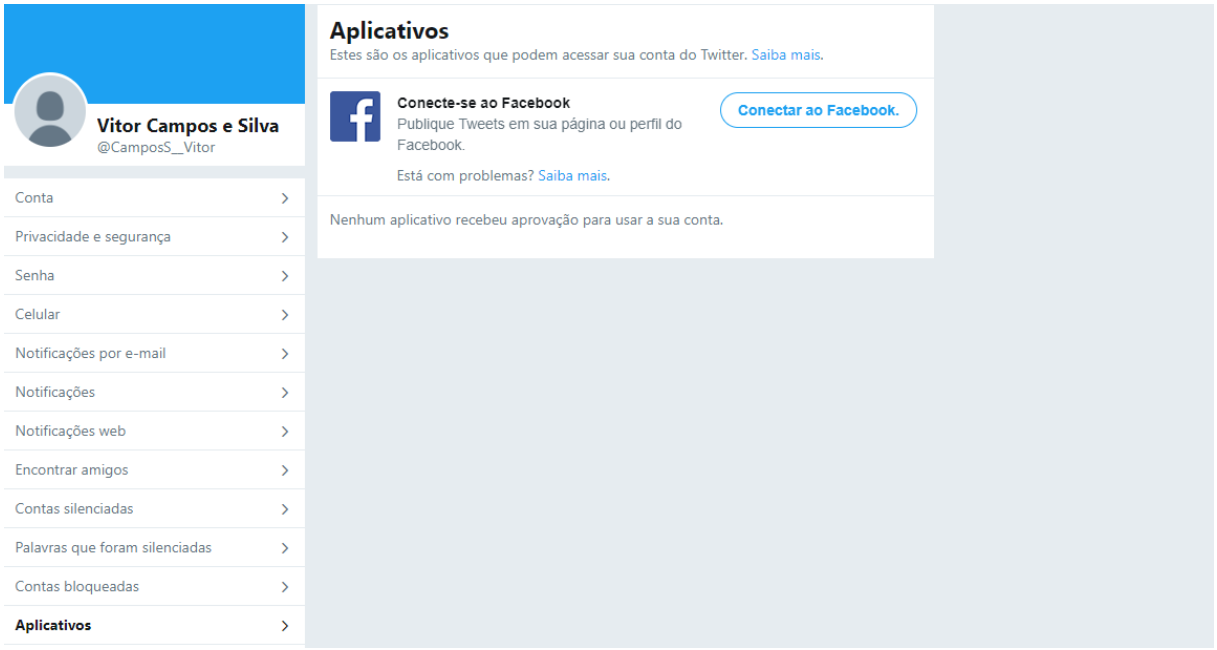
5.4.1 Cenário 6 - Teste de Conexão entre Twitter e Facebook

Para a realização do teste em questão, utilizaram-se as plataformas do Twitter e Facebook. Twitter (2018) é uma rede social que permite aos usuários enviar e receber atualizações pessoais de outros contatos, enquanto Facebook (2018) é uma mídia social e rede social virtual.

Vale ressaltar que para a correta execução deste teste, anteriormente deve-se realizar manualmente o procedimento para conectar o Twitter com o Facebook, onde se pressupõe que todo o conteúdo postado no Twitter aparecerá na linha do tempo do Facebook, validando o que o teste propõe comprovar.

Para realizar tal conexão, ao logar no Twitter, deve-se abrir a aba Aplicativos do menu Configurações e clicar no botão “Conectar ao Facebook”, conforme é apresentado na Figura 21.

Figura 21 – Teste executado após implementação de nova funcionalidade



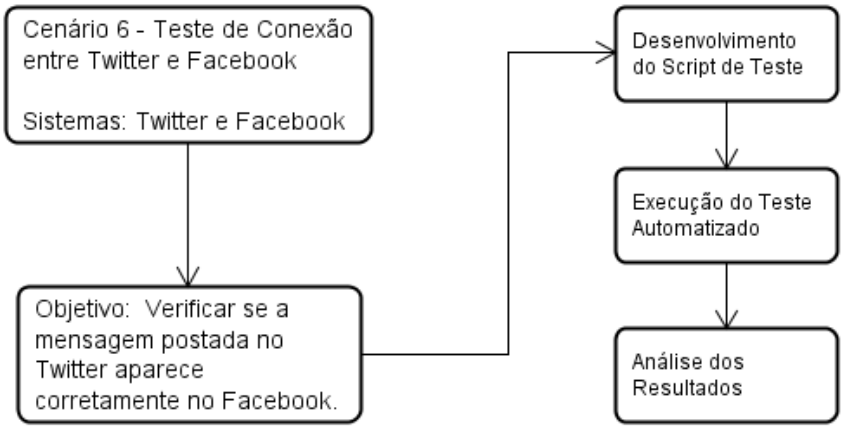
Fonte: Elaborado pelo autor

Utilizou-se o Selenium WebDriver com o intuito de verificar a correta conexão entre os dois sistemas. Para tal, desenvolveu-se um procedimento de teste com seu respectivo caso de teste.

Espera-se com a execução deste teste, poder realizar login no Twitter e postar a mensagem “Teste de Interconexão de Softwares - TCC”. Após a mensagem ser postada no Twitter, deve-se realizar o login no Facebook com o intuito de verificar se a mesma mensagem se encontra na linha do tempo, comprovando assim a correta conexão entre os dois sistemas.

Antes de realizar o teste desenvolvido para o cenário relatado, planejou-se a execução do mesmo, sendo criado um procedimento de teste e o seu respectivo caso de teste. A Figura 22 mostra a sequência dos procedimentos realizados para planejamento e execução do teste referente ao Cenário 6.

Figura 22 – Procedimentos do Cenário 6



Fonte: Elaborado pelo autor

O procedimento de teste apresentado no Quadro 21 relata todos os passos tomados para a realização do teste.

Quadro 21 – Procedimento de Teste 08

objetivo	Verificar se a mensagem postada no Twitter aparece corretamente no Facebook.
Fluxo	1. Logar no Twitter; 2. Postar a mensagem “Teste de Interconexão de Softwares - TCC”; 3. Logar no Facebook; 4. Verificar a existência da mensagem “Teste de Interconexão de Softwares - TCC”.

Fonte: Elaborado pelo autor

Para a execução do que foi apresentado no procedimento de teste 08, desenvolveu-se um caso de teste referente ao mesmo. O Quadro 22 apresenta tal caso de teste, com todas as entradas necessárias e as saídas esperadas.

Quadro 22 – Caso de Teste 08

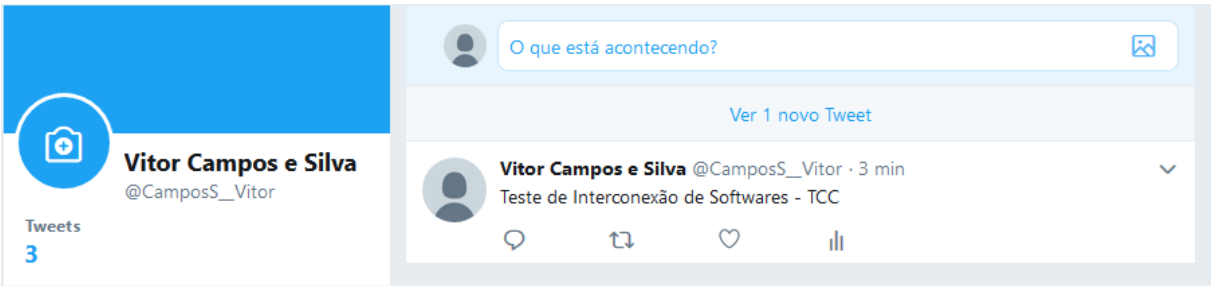
Identificação	Caso de Teste 08	
Itens a Testar	Verificar a correta conexão entre Twitter e Facebook.	
Procedimento de Teste	Referente ao Quadro 21	
Entradas	Campo	Valor
	Login_Twitter	vitor.comput@gmail.com
	Senha	XXXX
	Feed_Twitter	Teste de Interconexão de Softwares - TCC
	Login_Facebook	vitorcampossilva@hotmail.com
	Senha	XXXX
Saídas Esperadas	Feed_Facebook	Teste de Interconexão de Softwares - TCC
Dependências	Possuir cadastro no Twitter; Possuir cadastro no Facebook; Ter realizado manualmente a conexão entre os dois sistemas.	

Fonte: Elaborado pelo autor

Após o planejamento e desenvolvimento do procedimento de teste e seu respectivo caso de teste, passou-se para a etapa de desenvolvimento e execução do script de teste. Tal script se encontra disponível no Apêndice F.

O teste no geral se comportou conforme o esperado. O Selenium WebDriver conseguiu automaticamente abrir o Twitter e postar a mensagem, conforme é apresentado na Figura 23.

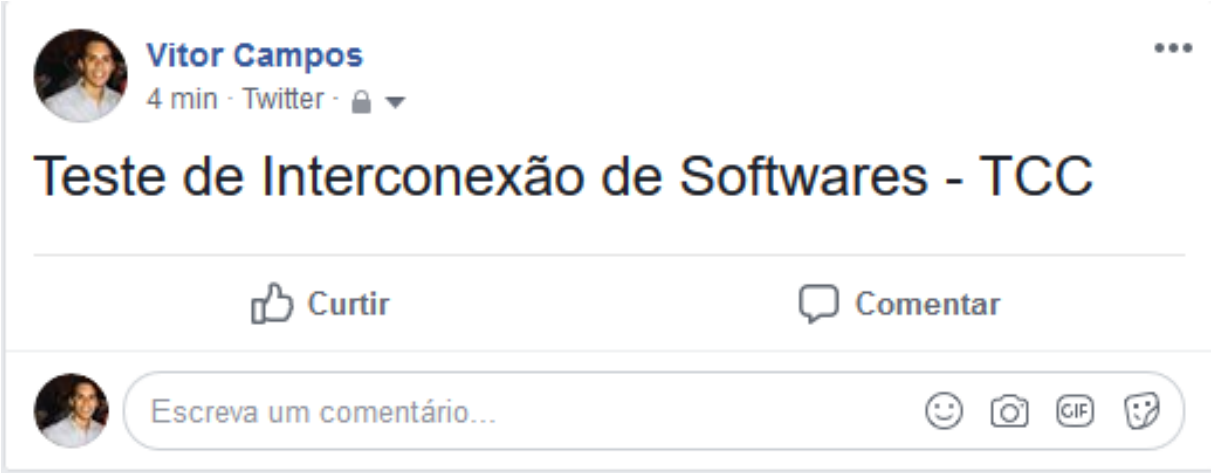
Figura 23 – Comentário no Twitter realizado com sucesso



Fonte: Elaborado pelo autor

Conforme a Figura 24, ao realizar o login no Facebook, a ferramenta conseguiu verificar a existência da mesma mensagem, comprovando assim que os dados foram transferidos de um sistema para o outro de forma correta.

Figura 24 – Comentário no Facebook verificado com sucesso

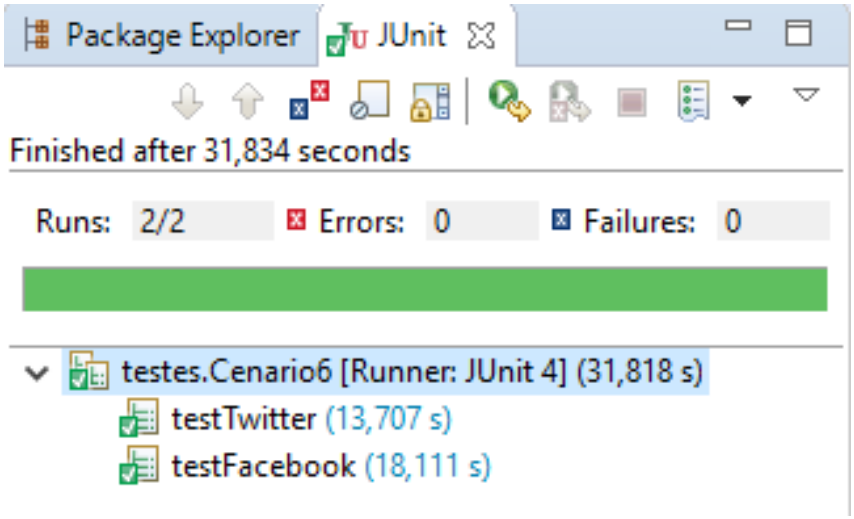


Fonte: Elaborado pelo autor

Para a execução deste teste é necessário que ao postar a mensagem no Twitter, a mesma seja armazenada em uma variável. Tal variável será utilizada ao realizar a busca da mensagem no Facebook, onde será comparado o texto encontrado na página com o texto presente na variável.

Conforme é apresentado na Figura 25, todo o teste foi realizado utilizando apenas um script de teste, sendo este dividido em duas classes. A primeira classe denominada de "testTwitter" é referente aos procedimentos que foram realizados na página do Twitter. O mesmo acontece com a segunda classe denominada "testFacebook" que é referente aos procedimentos realizados na página do Facebook. Ambas as instâncias foram executadas com sucesso, não apresentando erros ou falhas.

Figura 25 – Cenário 6 executado com sucesso



Fonte: Elaborado pelo autor

É necessário que tais classes sejam executadas na sequência correta, sendo execu-

tado primeiro a classe “testTwitter” e logo após a classe “testFacebook”. Caso a sequência de execução seja alterada, o teste resultará em falha, pois o mesmo tentará verificar no Facebook a mensagem “Teste de Interconexão de Softwares - TCC”, sendo que a mesma ainda não terá sido postada no Twitter.

5.5 Teste Paralelo

Os Testes Paralelos tem como intuito verificar se os resultados obtidos em uma nova versão de determinado software são compatíveis com os resultados obtidos a partir do processamento do sistema antigo ou da antiga versão do sistema.

Para realização de tal tipo de teste é necessário a utilização de duas versões distintas de uma mesma aplicação ou dois sistemas distintos, mas que se propõem a realizar as mesmas tarefas. Um exemplo da segunda situação seria a migração de um sistema desenvolvido em determinada linguagem para outra linguagem.

No cenário em questão, o teste realizado na seção 5.3.1 (cenário 5) além de se tratar de um Teste de Regressão também pode ser considerado como Teste Paralelo se levar em conta que a alteração feita no sistema Clubes Online consiste em uma nova versão do mesmo. Com isso o mesmo teste foi realizado com êxito em ambas as versões do sistema, resultando no que o teste paralelo se propõe a testar.

Como não foi utilizado para a realização do teste paralelo outro sistema online que possua duas versões distintas ou dois sistemas distintos que tenham objetivos semelhantes, fez-se a simulação deste tipo de teste utilizando dois softwares distintos e que não se assemelham quanto a seus requisitos.

Entende-se que o teste utilizando dois softwares distintos é válido, pois o que é realmente necessário verificar com a execução do teste paralelo é se a ferramenta Selenium WebDriver possibilita por meio de um único script de teste acionar simultaneamente os dois sistemas a serem testados. Com isso, se o teste for capaz de, com um único script, executar os procedimentos em dois sistemas diferentes, também será possível executar os procedimentos em dois sistemas equivalentes ou em duas versões diferentes de um mesmo software.

A possibilidade de acionamento de dois sistemas distintos por meio de um único script já foi realizado na seção 5.4.1, onde o Selenium WebDriver possibilita realizar testes simultâneos nas páginas do Twitter e Facebook.

Conclui-se então que se o teste for executado corretamente para dois sistemas distintos, ele também executará ao se utilizar sistemas equivalentes ou versões distintas de um mesmo sistema. Mesmo que os resultados obtidos na execução de cada um dos sistemas seja diferentes, o teste ainda é válido.

Para que os testes paralelos sejam executados corretamente nas duas versões do sistema ou nos dois sistemas distintos que se propõe a fazer as mesmas tarefas, é de extrema importância que os atributos dos elementos buscados em ambos os sistemas sejam idênticos. Caso haja diferença nos atributos de um sistema para o outro, não será possível executar com

sucesso o mesmo script de teste em ambos os sistemas.

Para esse tipo de teste, não se conseguiu encontrar um cenário adequado onde a ferramenta Selenium WebDriver não conseguisse realizar os testes automatizados.

5.6 Vantagens encontradas

Nessa seção são relatados recursos encontrados com a utilização do Selenium WebDriver, a partir dos testes realizados anteriormente. Tais recursos contribuem na tomada de decisão a respeito da utilização do Selenium WebDriver como ferramenta para a automatização de testes de software.

Uma das maiores vantagens da utilização do Selenium WebDriver para a automatização dos testes, é que o usuário tem a possibilidade de reutilizar os testes desenvolvidos sempre que necessário. Dentre todos os tipos de testes que atendem a técnica de teste funcional, este recurso é uma grande vantagem para o teste de regressão explicado na seção 5.3, onde, para cada alteração realizada no sistema, é necessário executar novamente toda a bateria de testes.

Não sendo utilizada a automatização para a realização dos testes, a cada alteração do sistema seria necessário que a pessoa responsável por realizar tais testes repetisse manualmente toda a bateria testes já feita anteriormente. Esta situação sendo repetidas diversas vezes em um cenário com muitas entradas de dados resultaria em um trabalho custoso e cansativo.

O Selenium WebDriver permite também, com a utilização de um mesmo script de teste, abrir duas ou mais páginas web e realizar diferentes tipos de testes em cada uma delas. Com isso, podem-se comparar dados ou até mesmo objetos de página de diferentes sites. Tal recurso foi utilizado na seção 5.4.1, onde o Selenium WebDriver possibilitou, no mesmo script, abrir as páginas do Twitter e Facebook para realizar o Teste de Interconexão de Software.

Em alguns cenários, como o do teste realizado na seção 5.4.1, acredita-se que o teste manual seria executado mais rapidamente, pelo fato de não ter o custo do desenvolvimento do script e se tratar de um cenário que envolve apenas dois sistemas e poucas entradas de dados. Já em um cenário mais complexo, que envolvesse a comunicação entre vários tipos de software, tendo várias entradas de dados em cada um dos sistemas, e sendo realizando a comparação entre dados gerados por tais sistemas, acredita-se que a realização do teste manual resultaria em um trabalho muito custoso e mais suscetível a erros.

Dependendo de como se pretende usar o Selenium Webdriver, pode-se ou não precisar do Selenium Server. Se o navegador e os testes forem todos executados na mesma máquina e os testes usarem apenas a API do WebDriver, não será necessário executar o Selenium Server. Nesse caso o Selenium WebDriver executará o navegador diretamente.

Segundo Selenium (2017), existem algumas razões para usar o Selenium Server com o Selenium WebDriver:

- Se o usuário utilizar o Selenium Grid para distribuir seus testes em várias máquinas ou máquinas virtuais (VMs);
- Se o usuário deseja se conectar a uma máquina remota que tenha uma versão de navegador específica que não esteja na sua máquina atual;
- Se o usuário não está usando as ligações Java (ou seja, Python, C ou Ruby) e gostaria de usar o HtmlUnit Driver.

Caso o usuário necessite de realizar algum dos procedimentos descritos anteriormente é necessário a utilização do Selenium Server, visto que tais procedimentos não podem ser realizados utilizando apenas o Selenium WebDriver.

De forma geral, entende-se que ferramenta Selenium WebDriver deve ser utilizada para testes independentes, em um único computador. Caso seja necessário realizar os testes de forma remota ou em máquinas virtuais, deve-se utilizar o Selenium Server juntamente com o Selenium Webdriver.

O presente trabalho não realiza uma comparação, em termos de velocidade, entre um cenário de teste executado automaticamente e manualmente. Mas entende-se pelos testes realizados, que com a utilização do Selenium Webdriver como ferramenta para automatização dos testes, podem-se realizar tais testes com maior rapidez, podendo ser executado uma grande bateria de testes em um menor intervalo de tempo.

Há certo custo para o desenvolvimento dos scripts de testes a serem utilizados por ferramentas de automatização, demandando tempo para a elaboração dos mesmos. Porem, após tais scripts estarem prontos, pode-se utiliza-los sempre que necessário. Tendo o script desenvolvido, também fica mais fácil realizar alterações no mesmo visando atender a alterações no sistema.

5.7 Limitações encontradas

A utilização do Selenium WebDriver possibilitou cobrir diferentes tipos de testes relacionados ao Teste Funcional, conforme relatado nas seções 5.1 a 5.5. Porém em alguns casos não foi possível desenvolver scripts de testes que sejam capazes de resolver determinados problemas, conforme são apresentados nas seções 5.7.1 a 5.7.6.

5.7.1 Automatização de páginas que possuem Captcha

Uma das limitações apresentadas na utilização do Selenium WebDriver para a automatização de testes é quanto à tentativa de automatizar páginas da web que possuam CAPTCHA.

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) é um teste de desafio cognitivo, utilizado como ferramenta anti-spam.

Como o CAPTCHA se trata de uma imagem, onde os computadores são incapazes de resolvê-lo, não possui um comando do Selenium WebDriver que consiga interpretar e resolver tal tecnologia, impossibilitando assim, a automatização por completo de funcionalidades que precisam utilizar desse recurso para serem concluídas.

5.7.2 Automatização em determinados sites responsivos

Uma limitação encontrada quanto à utilização da ferramenta Selenium Webdriver se diz respeito à tentativa de automatização de determinados sites responsivos.

Ao se modificar o tamanho da janela da página, alguns elementos que existiam anteriormente na tela maximizada, deixam de existir. Essa situação pode causar erro na execução do teste, visto que, um script projetado para testar uma determinada funcionalidade, considerando uma tela maximizada, pode tentar acessar um objeto que deixou de existir após o redimensionamento.

Vale ressaltar que nessa situação a limitação não é total do Selenium Webdriver, pois, uma vez que o elemento deixa de existir na tela, também não seria possível a identificação do mesmo por meio do teste manual. A limitação do Selenium neste caso é apenas em não conseguir automaticamente identificar tal problema, reajustando o script de teste para que o mesmo continue a executar.

5.7.3 Automatização em páginas que não possuem elementos estáticos

Em determinados sites, como por exemplo, alguns sites de compras, as páginas têm seus produtos modificados constantemente, visto que há uma alta rotação dos produtos disponíveis para a venda.

Tal situação inviabiliza a utilização do Selenium WebDriver para a automatização dos testes onde a busca por determinado produto seja feita pela descrição do mesmo, uma vez que o mesmo teste que executa corretamente em um dado momento, pode deixar de funcionar em outro momento, pelo fato de que o produto buscado pelo script de teste pode não estar mais presente naquela página.

Caso a busca seja feita pelo xpath, passando o "caminho"do elemento dentro da página html, o teste pode não resultar em falha, mas o produto obtido com a busca pode ser diferente do encontrado ao rodar o mesmo script de teste em outro momento.

Tal limitação se dá pelo fato do teste estar sendo aplicado em uma página com uma alta rotação de seus produtos, onde a utilização de um mesmo script de teste do Selenium WebDriver pode obter resultados diferentes em momentos distintos.

Vale ressaltar que não é tão provável que este problema ocorra ao se realizar um teste de requisitos, visto que tal teste é realizado na fase de desenvolvimento, onde acredita-se que a página não terá tantos produtos cadastrados e, com isso, não sofrerá grandes alterações dos mesmos.

É mais provável a ocorrência de tal problema ao se realizar um teste de regressão, após ser implementado uma nova versão do sistema, onde o mesmo já esteja sendo utilizado e possua uma considerável alteração dos produtos a venda.

5.7.4 Reexecução do script de teste

Um problema da utilização do Selenium WebDriver que afeta todos os tipos de teste referentes a técnica de teste funcional descritos anteriormente, está relacionado com a reexecução do script de teste desenvolvido para realizar testes automatizados em determinado sistema web.

Conforme foi relatado anteriormente no Quadro 3, o Selenium WebDriver utiliza o comando findElement() para buscar os elementos da página por:

- className;
- cssColector
- id
- linkText
- name
- partialLinkText
- tagName
- xpath

Caso determinado elemento tenha algum desses atributos alterados, a ferramenta Selenium Webdriver não conseguirá localizar o mesmo ao utilizar o script de teste já desenvolvido. Nesse cenário, um teste que era executado com sucesso passará a resultar em falha.

Mesmo após alterações dos atributos de determinado elemento da página, o teste sendo realizado de forma manual é executado com sucesso, pelo fato de que, ao realizar o teste desta forma, não se tem a necessidade de conhecer os atributos do elemento e nem realizar a busca por tais atributos.

Conclui-se então que, na situação relatada, o teste automatizado pelo Selenium WebDriver resultará em falha em um cenário onde o sistema testado está funcionando perfeitamente.

Tal problema está sujeito a ocorrer com a automatização de qualquer tipo de teste, mas é mais provável que o mesmo ocorra ao realizar o teste de regressão, onde a cada alteração do sistema é necessário realizar a reexecução de toda a bateria de testes já realizada anteriormente.

A reexecução dos scripts de teste também se mostra um problema nos cenários onde o sistema a ser testado trabalha com datas e horários. Nesses cenários, o Selenium WebDriver não consegue garantir a consistência dos resultados obtidos todas as vezes que o script de teste for executado, sendo necessário realizar manualmente os devidos ajustes no mesmo.

Um exemplo do problema citado anteriormente ocorre no Cenário 3 presente na seção 5.2.1. Nesse cenário o Selenium WebDriver tenta acessar uma data anterior a data atual e a execução resulta em falha. O mesmo problema pode ocorrer ao tentar reexecutar um script de teste onde a data de inicio de evento que antes era posterior a data atual passe a ser anterior a data atual.

5.7.5 Automatização em páginas que possuem o código html complexo

O serviço de webmail Gmail foi um dos sistemas pensados para se utilizar no desenvolvimento do presente trabalho. Por se tratar de um sistema robusto, o Gmail possui diversas funcionalidades que poderiam ser exploradas para a criação de outros cenários de teste, além dos relatados anteriormente. Ao tentar utilizar o Gmail para realizar tais testes, foram encontradas várias dificuldades.

Em um primeiro momento, houve dificuldade ao tentar realizar o login no sistema. O Selenium WebDriver realizou corretamente os procedimentos iniciais, abrindo a página do sistema e inserindo os dados no campo email. Porém a ferramenta não conseguia inserir os dados referentes ao campo senha, fazendo com que o teste resultasse em falha. Tal falha ocorreu pois a ferramenta não conseguiu encontrar o elemento referente ao campo senha, buscando o mesmo pelo "id"(mesma forma utilizada para buscar o campo email). Foi necessário então, utilizar os comandos apresentados na Figura 26 para realizar o login.

Figura 26 – Login no Gmail

```
@Test
public void gmail() throws Exception {

    // Abrindo o site
    driver.get(baseUrl + "/");

    // Realizando o login
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    WebElement email_phone = driver.findElement(By.xpath("//input[@id='identifierId']"));
    email_phone.sendKeys("seu_email");
    driver.findElement(By.id("identifierNext")).click();
    WebElement password = driver.findElement(By.xpath("//input[@name='password']"));
    WebDriverWait wait = new WebDriverWait(driver, 20);
    wait.until(ExpectedConditions.elementToBeClickable(password));
    password.sendKeys("sua_senha");
    driver.findElement(By.id("passwordNext")).click();
```

Fonte: Elaborado pelo autor

Após logar no sistema tentou-se desenvolver scripts de teste para automatizar cenários pensados, porém houve dificuldades na execução de tal tarefa. O Gmail não apresenta um

código html claro, onde muitas vezes não é possível identificar atributos necessários para realizar a busca de determinados elementos da página.

Acredita-se que o embaralhamento do código da página html do Gmail se dá justamente para dificultar que bots acessem seu conteúdo. Há quem diga ainda que automatizaria o envio de emails no Gmail não por meio da interface da Web, mas por meio de um protocolo de e-mail. Por estes motivos, não se conseguiu utilizar o Selenium WebDriver para realizar a automatização de testes no Gmail.

5.7.6 Localizar elemento pelo layout

Como já foi dito anteriormente na seção 2.6, a ferramenta Selenium Webdriver possibilita realizar a busca dos elementos presentes em determinada página web pelas seguintes formas:

- Id;
- Class Name;
- Tag name;
- Name;
- Link Text;
- Partial Link Text;
- CSS;
- Xpath.

Porém, caso seja necessário, o Selenium WebDriver não apresenta a alternativa de realizar a busca de tais elementos por meio da posição dos mesmos na tela ou, ainda, pela ordem de exibição.

Realizar a busca pela posição do elemento na tela seria útil em um cenário onde a página automatizada apresente um código HTML muito "embaralhado", não sendo possível identificar facilmente os atributos referentes ao elemento buscado.

5.8 Vantagens x Limitações

Nessa seção será apresentado em forma de quadro uma comparação entre as vantagens e limitações na utilização do Selenium WebDriver como ferramenta para a automatização de teste funcional.

A comparação realizada em forma de quadro tem como intenção apresentar de maneira mais clara e intuitiva quais são os prós e os contras do Selenium WebDriver citados no presente trabalho a partir da exploração da ferramenta.

Quadro 23 – Selenium WebDriver - Vantagens x Limitações

Vantagens	**Limitações**
• Os scripts podem ser escritos em várias linguagens; • É suportado por vários navegadores; • Possibilita reutilizar os scripts de teste a qualquer momento; • Possibilita a execução em paralelo de sistemas distintos, utilizando um único script; • Possibilita a integração com diferentes APIs e plug-ins; • Com o auxílio do Selenium Server possibilita automatizar aplicações remotas ou realizar testes em várias máquinas virtuais.	• Não possibilita automatizar páginas que possuem Captcha; • Não possibilita automatizar determinados sites responsivos que tem seus elementos modificados ao se alterar o tamanho da janela; • Apresenta dificuldade em automatizar páginas que não possuem elementos estáticos; • Falha na reexecução do script de teste quando são alterados os atributos dos elementos buscados; • Apresenta dificuldade em automatizar testes em sistemas que não sejam bem estruturados e que não possuam uma padronização do código html; • Pode ocorre falha na reexecução do script de teste em cenários que trabalhem com data e horário; • Apresenta dificuldade em automatizar sistemas que possuem o código html embaralhado, onde não se é possível identificar os atributos do elemento buscado; • Não possibilita localizar o elemento pela posição no layout da página.

Fonte: Elaborado pelo autor

A comparação realizada também visa ser uma maneira de auxiliar os leitores a definirem sobre a utilização do Selenium WebDriver ao se depararem em uma situação de escolha de uma ferramenta para a realização de testes automatizados.

6 Considerações Finais

Com a realização dos testes espera-se trazer grandes benefícios para o desenvolvimento de softwares, gerando sistemas mais confiáveis onde se espera que a maior quantidade possível de defeitos tenha sido identificados e corrigidos antes do produto final ser entregue ao usuário.

A automatização dos testes de software da à possibilidade de ter os cenários de teste reproduzíveis a qualquer momento, caso seja necessário a ré execução dos mesmos. Em algumas situações serão necessárias apenas algumas modificações no script de teste, caso haja alterações na parte do sistema já testada. Tais alterações visam ajustar o script de teste para executar na nova versão do software.

Realizando os testes automaticamente, evita-se um excessivo trabalho manual e repetitivo, reduzindo o tempo dedicado para essa tarefa, além de ser um grande facilitador para a realização do teste de regressão, obtendo assim um *feedback* mais rápido e confiável.

O estudo em questão visa ser um apoio a mais aos estudantes da área de testes de software, uma vez que apresenta diversos scripts para cenários distintos, facilitando assim o aprendizado e uso da ferramenta.

Por meio dos testes realizados e dos resultados obtidos, pode-se perceber que o Selenium WebDriver apresentou alguns problemas para a realização dos testes automatizados, mas, no geral, a ferramenta atendeu bem aos requisitos exigidos, executando conforme o esperado em quase todos os cenários de teste no qual foi submetida.

A ferramenta obteve um resultado satisfatório quanto à realização dos testes de requisitos, demonstrando que as funcionalidades testadas realizavam o que era esperado. Quanto ao teste realizado com um carrinho de compras deve-se ter uma atenção quanto às alterações necessárias no script de teste para que o mesmo execute com sucesso, visto que se trata de uma página onde os elementos não são apresentados de forma estática.

Dentre os cenários abordados para o teste de tratamento de erro, a ferramenta, além de identificar os casos que resultavam em erro, também conseguiu verificar que os sistemas exibiam as devidas mensagens de erro para os usuários, conforme se era esperado. Não se conseguiu pensar em algum cenário onde a ferramenta Selenium WebDriver não conseguisse realizar os testes submetidos.

É promissora a utilização da ferramenta para realizar o teste de regressão, onde se tem a necessidade de realizar toda a bateria de testes a cada nova alteração no sistema. Neste cenário é muito útil poder repetir sempre que necessário os scripts de testes já desenvolvidos, ganhando em tempo e qualidade, porém é de extrema importância realizar as devidas alterações no script de teste, visando atender às adaptações realizadas no sistema e não permitindo que a execução do teste acuse falha em um sistema que apresenta a execução de suas funcionalidades conforme o esperado.

O teste de interconexão com outros softwares foi realizado conforme o esperado, onde a ferramenta Selenium WebDriver conseguiu com a utilização de um único script de teste acessar simultaneamente as páginas do Twitter e Facebook, realizando as verificações e comparações necessárias em ambos sistemas.

Mesmo não sendo criado um cenário específico para a realização do teste paralelo, pôde-se chegar a conclusão, por meio dos testes realizados nos outros tipos de teste, de que o Selenium WebDriver também tem a capacidade de realizar tal tipo de teste sem apresentar falhas.

Por mais que os testes de suporte manual e teste de controle sejam dois tipos de testes que atendem a técnica de teste funcional, não foram criados cenários de teste para os mesmos no presente trabalho, por se entender que tais tipos de testes não podem ser automatizados utilizando um script de teste executado pelo Selenium Webdriver.

Os testes realizados, em sua maioria, não foram testes complexos, por não serem resultados de cenários complexos. Porém, dentro do que se foi proposto a realizar, a ferramenta Selenium WebDriver atendeu de forma satisfatória em quase todos os cenários projetados, executando sem apresentar falhas quase todos os scripts de teste desenvolvidos.

Para que os testes sejam realizados com maior facilidade e sem gerar grandes problemas, é necessário que se tenha um sistema bem estruturado, com uma padronização do código do html relativo a interface que está sendo testada. Espera-se que cada um dos elementos possuam atributos únicos e bem definidos e que as mensagens de erro possuam um posicionamento pré-definido. Uma boa estruturação do código html da página do sistema, com um "código limpo", auxilia na utilização da ferramenta Selenium WebDriver.

É de extrema importância que se faça as devidas modificações no script de teste caso seja realizado alterações nos atributos de determinado elemento. Deve-se observar se o script de teste realiza a busca de tal elemento por meio do atributo alterado. Caso essa alteração não seja realizada, o script de teste resultará em falha, em um cenário onde o sistema se comporta corretamente. Este é um problema grave e que deve ser analisado para todos os tipos de teste referente a técnica de teste funcional.

Vale ressaltar a importância de realizar uma análise prévia para definir se os testes deverão ser ou não automatizados, avaliando quais os ganhos a automatização trará para o desenvolvimento do sistema. Em projetos pequenos, com funcionalidades simples e que tenham pouco tempo de desenvolvimento, a utilização do Selenium WebDriver como ferramenta de automatização talvez não se faz necessária. Nesses casos, o custo existente para o desenvolvimento dos scripts de teste pode comprometer com o prazo para a entrega do sistema.

O custo da criação dos scripts de teste para a execução automatizada geralmente é maior do que o custo dos testes realizados de forma manual, justamente por causa do tempo gasto com a implementação de tais scripts. Por este motivo, é necessário que este custo adicional da implementação seja compensado com os benefícios gerados pela automatização.

De forma geral, é recomendado a utilização da ferramenta Selenium WebDriver para a automatização de tarefas repetitivas, projetos longos e que são atualizados com frequência.

Nessas situações, a realização dos testes de forma manual se torna uma tarefa cansativa, onde para cada alteração do sistema se faz necessário a execução de todos os testes já realizados. Com a automatização de tais testes, é necessário apenas fazer pequenos ajustes para que estes sejam executados sempre que necessário.

Talvez não seja a melhor escolha utilizar a ferramenta Selenium WebDriver para realizar testes em projetos pequenos, que possuam poucas entradas de dados e que não sejam atualizados frequentemente. Também não é indicado a automatização dos testes em projetos que apresentem um prazo de entrega reduzido, visto que o desenvolvimento dos scripts de teste tem um custo alto que deve ser considerado. Nesses cenários, talvez seja mais prático e rápido a realização dos testes de forma manual.

Como sugestão para trabalhos futuros, recomenda-se a exploração de outras ferramentas de automatização de teste funcional, avaliando-as e realizando uma comparação em relação aos recursos e limitações de uma em relação às demais.

Referências

BAHIA, C. *Nossa História*. São Paulo: [s.n.], 2018. Disponível em: <https://institucional.casasbahia.com.br/empresa/nossa-historia>. Acesso em: 20/02/2018. Citado na página 47.

BARBOSA, I. d. O. *Avaliaçao de Qualidade no Sistema de Locações da Faculdade de Balsas através de Testes Funcionais Automatizados*. 2013, 2013. Citado nas páginas 26 e 27.

BASTOS, A. et al. *Base de Conhecimento em Teste de Software*. 3. ed. São Paulo: [s.n.], 2012. Citado nas páginas 15, 16, 17, 18, 19, 29 e 33.

BERNARDO, P. C.; KON, F. A importância dos testes automatizados. *Engenharia de Software Magazine*, v. 3, p. 54–57, 2008. Citado na página 12.

CAETANO, C. Melhores práticas na automação de testes. *Engenharia de Software Magazine*, v. 5, p. 42–47, 2008. Citado na página 12.

CAMPOS, R. *Selenium IDE, ferramenta de teste automatizado*. 2011. Disponível em: <https://www.profissionaisti.com.br/2011/03/testes-automatizados-utilizando-selenium-ide/>. Acesso em: 06/02/2018. Citado na página 21.

DAILBERT, M. S.; TOLEDO, J. V.; ARAUJO, M. A. P. Testes automatizados de software em web service. *Engenharia de Software Magazine*, v. 8, p. 52–59, 2008. Citado na página 12.

FACEBOOK. *Sobre*. Califórnia: [s.n.], 2018. Disponível em: <https://www.facebook.com/pg/facebook/about/>. Acesso em: 10/03/2018. Citado na página 55.

FILHO, W. d. P. P. *Engenharia de Software: fundamentos, métodos e padrões*. 2. ed. Travessa do Ouvidor, 11 - Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora S.A., 2001. Citado nas páginas 20 e 21.

FILHO, W. d. P. P. *Engenharia de Software: fundamentos, métodos e padrões*. 3. ed. Travessa do Ouvidor, 11 - Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora S.A., 2012. Citado na página 20.

HEINEN, E. G. Automatização de testes de software com selenium ide e webdriver. *Engenharia de Software Magazine*, v. 73, p. 26–32, 2014. Citado na página 22.

MALDONADO, J. C. et al. Introdução ao teste de software. *Notas Didaticas do Icmc*, World Scientific, São Carlos, 2004. Disponível em: <http://pbjug-grupo-de-usuarios-java-da-paraiba.1393240.n2.nabble.com/attachment/2545944/0/IntroducaoAoTesteDeSoftware-Maldonado,Jose.pdf>. Acesso em: 03/07/2017. Citado nas páginas 15 e 34.

MATIELI, L. V.; Oliveira de Araujo, F. Estimativa de teste de software: Levantamento bibliográfico em veículos brasileiros e internacionais. *Relatórios de Pesquisa em Engenharia de Produção*, v. 15, p. 1–13, 2015. Disponível em: <http://www.rpep.uff.br/index.php/RPEP/article/view/11/8>. Acesso em: 03/03/2017. Citado na página 13.

NETO, A. C. D. Introdução a teste de software. *Engenharia de Software Magazine*, v. 1, p. 54–59, 2008. Citado na página 15.

NETO, A. C. D. Planejamento de Testes a partir de Casos de Uso. *Engenharia de Software Magazine*, v. 6, p. 36–41, 2008. Citado na página 12.

NETO, A. C. D. Automatizando testes funcionais em aplicações. *Engenharia de Software Magazine*, v. 24, p. 49–56, 2010. Citado nas páginas 15, 16 e 21.

SELENIUM. *Selenium Documentation*. Chicago: [s.n.], 2017. Disponível em: <http://www.seleniumhq.org/docs>. Acesso em: 05/07/2017. Citado nas páginas 21, 22, 23 e 61.

SOARES, J. V. *Ferramentas de Suporte a Automatização de Teste Funcional: Levantamento Bibliográfico*. 2017 — Centro Federal de Educação Tecnológica de Minas Gerais, 2017. Disponível em: <http://sistemas.timoteo.cefetmg.br/nos/bd:tcc:ec:2016:soares>. Acesso em: 27/04/2017. Citado nas páginas 12, 13, 24, 25, 29 e 32.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson, 2011. ISBN 8579361087. Citado na página 16.

STOCCO, A. et al. *APOGEN: automatic page object generator for web testing*. Springer US, 2017. v. 25. 1007–1039 p. ISSN 15731367. ISBN 1121901693319. Disponível em: <https://link-springer-com.ez107.periodicos.capes.gov.br/content/pdf/10.1007%2Fs11219-016-9331-9.pdf>. Acesso em: 05/12/2017. Citado nas páginas 27 e 28.

SUBMARINO. 2018. Disponível em: <https://www.submarino.com.br/>. Acesso em: 19/02/2018. Citado na página 40.

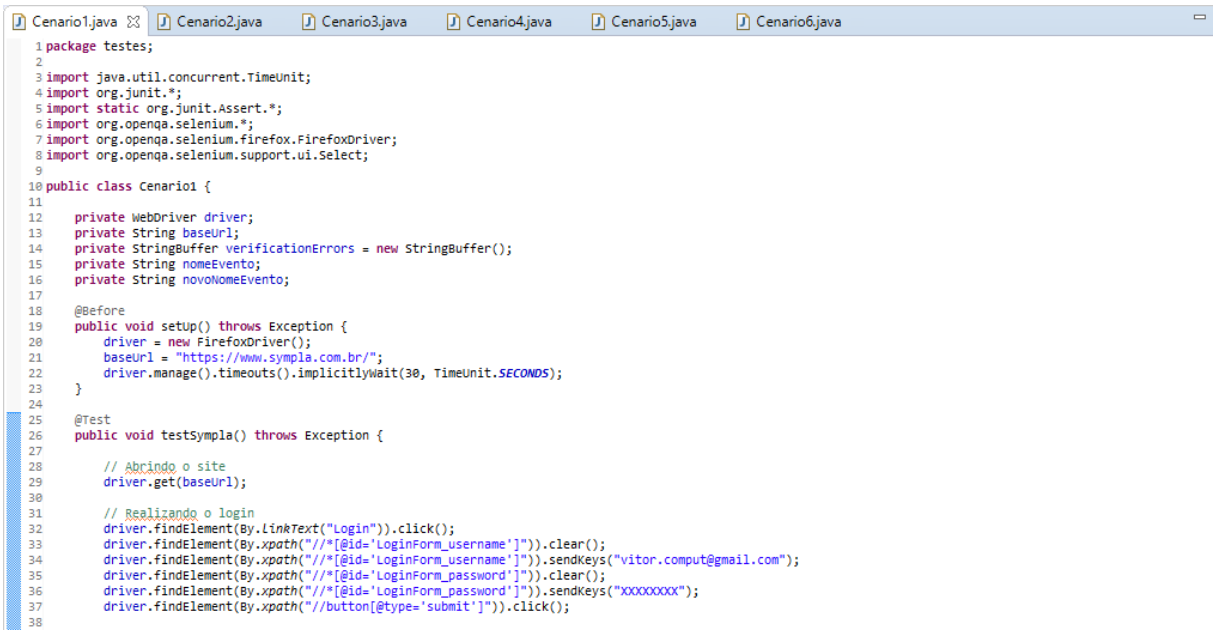
SYMPLA. *Quem Somos*. 2018. Disponível em: <https://www.sympla.com.br/sobre-sympla>. Acesso em: 17/02/2018. Citado nas páginas 34 e 42.

TWITTER. Califórnia: [s.n.], 2018. Disponível em: <https://about.twitter.com/pt.html>. Acesso em: 10/03/2018. Citado na página 55.

VERGARA, S. C. *Projetos e relatórios de pesquisa em administração*. 15. ed. São Paulo: Atlas, 2000. Citado na página 29.

A SCRIPT DE TESTE – CENÁRIO 1

Figura 27 – Cenário 1 - Primeira parte do script



```
Cenario1.java | Cenario2.java | Cenario3.java | Cenario4.java | Cenario5.java | Cenario6.java

1  package testes;
2
3  import java.util.concurrent.TimeUnit;
4  import org.junit.*;
5  import static org.junit.Assert.*;
6  import org.openqa.selenium.*;
7  import org.openqa.selenium.firefox.FirefoxDriver;
8  import org.openqa.selenium.support.ui.Select;
9
10 public class Cenario1 {
11
12     private WebDriver driver;
13     private String baseUrl;
14     private StringBuffer verificationErrors = new StringBuffer();
15     private String nomeEvento;
16     private String novoNomeEvento;
17
18     @Before
19     public void setUp() throws Exception {
20         driver = new FirefoxDriver();
21         baseUrl = "https://www.sympla.com.br/";
22         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
23     }
24
25     @Test
26     public void testSympla() throws Exception {
27
28         // Abrindo o site
29         driver.get(baseUrl);
30
31         // Realizando o login
32         driver.findElement(By.linkText("Login")).click();
33         driver.findElement(By.xpath("//*[@id='LoginForm_username']")).clear();
34         driver.findElement(By.xpath("//*[@id='LoginForm_username']")).sendKeys("vitor.comput@gmail.com");
35         driver.findElement(By.xpath("//*[@id='LoginForm_password']")).clear();
36         driver.findElement(By.xpath("//*[@id='LoginForm_password']")).sendKeys("XXXXXXXX");
37         driver.findElement(By.xpath("//button[@type='submit']")).click();
38
```

Fonte: Elaborado pelo autor

Figura 28 – Cenário 1 - Segunda parte do script

```
39         // Criando novo evento
40         driver.findElement(By.linkText("Criar evento")).click();
41         driver.findElement(By.xpath("//*[@id='Event_NAME']")).clear();
42         driver.findElement(By.xpath("//*[@id='Event_NAME']")).sendKeys("Simpósio em Engenharia de Computação");
43         nomeEvento = driver.findElement(By.xpath("//*[@id='Event_NAME']")).getText();
44         driver.findElement(By.cssSelector("img.ui-datepicker-trigger")).click();
45         driver.findElement(By.xpath("//*[@id='ui-datepicker-div']/table/tbody/tr[5]/td[2]/a")).click();
46         new Select(driver.findElement(By.xpath("//*[@id='Event_startTimeHour']"))).selectByVisibleText("10");
47         new Select(driver.findElement(By.xpath("//*[@id='Event_startTimeMinute']"))).selectByVisibleText("30");
48         driver.findElement(By.xpath("(//img[@alt='...'])[2]")).click();
49         driver.findElement(By.xpath("//*[@id='ui-datepicker-div']/table/tbody/tr[5]/td[6]/a")).click();
50         new Select(driver.findElement(By.xpath("//*[@id='Event_endTimeHour']"))).selectByVisibleText("17");
51         new Select(driver.findElement(By.xpath("//*[@id='Event_endTimeMinute']"))).selectByVisibleText("30");
52         new Select(driver.findElement(By.xpath("//*[@id='Event_EVENT_ADDRESS_ID']")))
53                 .selectByVisibleText("Evento online");
54         new Select(driver.findElement(By.xpath("//*[@id='Event_EVENT_HOST_ID']"))).selectByVisibleText("Vitor");
55         driver.findElement(By.xpath("//*[@id='txtTicketName']")).clear();
56         driver.findElement(By.xpath("//*[@id='txtTicketName']")).sendKeys("Ingresso Único");
57         driver.findElement(By.xpath("//*[@id='txtTicketPrice']")).clear();
58         driver.findElement(By.xpath("//*[@id='txtTicketPrice']")).sendKeys("10,00");
59         driver.findElement(By.xpath("//*[@id='txtTicketQty']")).clear();
60         driver.findElement(By.xpath("//*[@id='txtTicketQty']")).sendKeys("100");
61         driver.findElement(By.cssSelector("#ticketStartData > img.ui-datepicker-trigger")).click();
62         driver.findElement(By.xpath("//*[@id='ui-datepicker-div']/table/tbody/tr[3]/td[4]/a")).click();
63         new Select(driver.findElement(By.xpath("//*[@id='TicketStartTimeHour']"))).selectByVisibleText("05");
64         new Select(driver.findElement(By.xpath("//*[@id='TicketStartTimeMin']"))).selectByVisibleText("30");
65         driver.findElement(By.cssSelector("#ticketEnd > img.ui-datepicker-trigger")).click();
66         driver.findElement(By.xpath("//*[@id='ui-datepicker-div']/table/tbody/tr[3]/td[6]/a")).click();
67         new Select(driver.findElement(By.xpath("//*[@id='TicketEndTimeHour']"))).selectByVisibleText("19");
68         new Select(driver.findElement(By.xpath("//*[@id='TicketEndTimeMin']"))).selectByVisibleText("00");
69         driver.findElement(By.cssSelector("#btnAddTicket > span")).click();
70         new Select(driver.findElement(By.id("Event_CATEGORY_PRIM_ID"))).selectByVisibleText("Congresso, seminário");
71         new Select(driver.findElement(By.id("Event_CATEGORY_SEC_ID"))).selectByVisibleText("Curso, workshop");
72         new Select(driver.findElement(By.id("Event_PRIVATE_EVENT"))).selectByVisibleText("Privado");
73         new Select(driver.findElement(By.id("Event_PRIVATE_EVENT"))).selectByVisibleText("Público");
74         new Select(driver.findElement(By.id("Event_TYPE_SERVICE"))).selectByVisibleText("Inscrição");
75         new Select(driver.findElement(By.id("Event_TYPE_SERVICE"))).selectByVisibleText("Ingresso");
76         driver.findElement(By.cssSelector("#draft > span")).click();
77
```

Fonte: Elaborado pelo autor

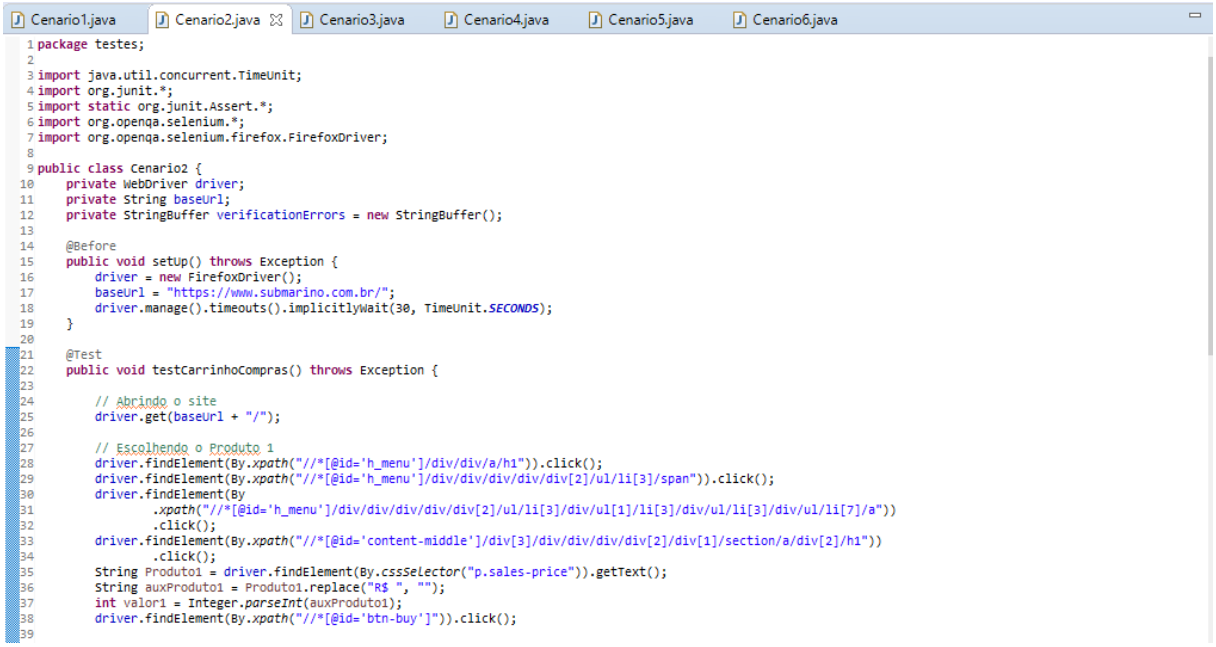
Figura 29 – Cenário 1 - Terceira parte do script

```
78          // Verificando a existencia do evento criado na página "Meus eventos"
79          driver.findElement(By.linkText("Meus eventos")).click();
80          assertEquals(nomeEvento,
81                  driver.findElement(By.xpath("//*[@id='event-grid']/table/tbody/tr/td[2]/a")).getText());
82
83          // Editando o evento
84          driver.findElement(By.xpath("//*[@id='event-grid']/table/tbody/tr/td[2]/a")).click();
85          driver.findElement(By.xpath("//*[@id='yt0']/span")).click();
86          driver.findElement(By.xpath("//*[@id='Event_NAME']")).clear();
87          driver.findElement(By.xpath("//*[@id='Event_NAME']"))
88                  .sendKeys("Congresso Nacional de Engenharia de Computação");
89          novoNomeEvento = driver.findElement(By.xpath("//*[@id='Event_NAME']")).getText();
90          driver.findElement(By.xpath("//*[@id='draft']/span")).click();
91
92          // Verificando as alterações realizadas no evento na página "Meus
93          // eventos"
94          driver.findElement(By.linkText("Meus eventos")).click();
95          assertEquals(novoNomeEvento,
96                  driver.findElement(By.xpath("//*[@id='event-grid']/table/tbody/tr/td[2]/a")).getText());
97      }
98
99      @After
100     public void tearDown() throws Exception {
101         // driver.quit();
102         String verificationErrorString = verificationErrors.toString();
103         if (!"".equals(verificationErrorString)) {
104             fail(verificationErrorString);
105         }
106     }
107 }
108
```

Fonte: Elaborado pelo autor

B SCRIPT DE TESTE – CENÁRIO 2

Figura 30 – Cenário 2 - Primeira parte do script



Cenario1.java | Cenario2.java | Cenario3.java | Cenario4.java | Cenario5.java | Cenario6.java

```
1  package testes;
2
3  import java.util.concurrent.TimeUnit;
4  import org.junit.*;
5  import static org.junit.Assert.*;
6  import org.openqa.selenium.*;
7  import org.openqa.selenium.firefox.FirefoxDriver;
8
9  public class Cenario2 {
10     private WebDriver driver;
11     private String baseUrl;
12     private StringBuffer verificationErrors = new StringBuffer();
13
14     @Before
15     public void setUp() throws Exception {
16         driver = new FirefoxDriver();
17         baseUrl = "https://www.submarino.com.br/";
18         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
19     }
20
21     @Test
22     public void testCarrinhoCompras() throws Exception {
23
24         // Abrindo o site
25         driver.get(baseUrl + "/");
26
27         // Escolhendo o Produto 1
28         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/a/h1")).click();
29         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/div/div/div[2]/ul/li[3]/span")).click();
30         driver.findElement(By
31                 .xpath("//*[@id='h_menu']/div/div/div/div/div[2]/ul/li[3]/div/ul[1]/li[3]/div/ul/li[3]/div/ul/li[7]/a"))
32                 .click();
33         driver.findElement(By.xpath("//*[@id='content-middle']/div[3]/div/div/div/div[2]/div[1]/section/a/div[2]/h1"))
34                 .click();
35         String Produto1 = driver.findElement(By.cssSelector("p.sales-price")).getText();
36         String auxProduto1 = Produto1.replace("R$ ", "");
37         int valor1 = Integer.parseInt(auxProduto1);
38         driver.findElement(By.xpath("//*[@id='btn-buy']")).click();
39
```

Fonte: Elaborado pelo autor

Figura 31 – Cenário 2 - Segunda parte do script

```
40         // Escolhendo o produto 2
41
42         driver.findElement(By.id("brd-link")).click();
43         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/a/h1")).click();
44         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/div/div/div[2]/ul/li[5]/span")).click();
45         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/div/div/div[2]/ul/li[5]/div/ul[1]/li[3]/div/ul/li[3]/a"))
46                 .click();
47         driver.findElement(By.xpath("//*[@id='content-middle']/div[3]/div/div/div/div[2]/div[14]/section/a/div[2]/h1"))
48                 .click();
49         String Produto2 = driver.findElement(By.cssSelector("p.sales-price")).getText();
50         String auxProduto2 = Produto2.replace("R$ ", "");
51         int valor2 = Integer.parseInt(auxProduto2);
52         driver.findElement(By.xpath("//*[@id='btn-buy']")).click();
53         driver.findElement(By.xpath("//*[@id='btn-continue']")).click();
54
55         // Escolhendo o produto 3
56
57         driver.findElement(By.id("brd-link")).click();
58         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/a/h1")).click();
59         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/div/div/div[2]/ul/li[6]/span")).click();
60         driver.findElement(By.xpath("//*[@id='h_menu']/div/div/div/div/div[2]/ul/li[6]/div/ul[1]/li[3]/div/ul/li[3]/a"))
61                 .click();
62         driver.findElement(By.xpath("//*[@id='content-middle']/div[3]/div/div/div/div[2]/div[28]/section/a/div[2]/h1"))
63                 .click();
64         String Produto3 = driver.findElement(By.cssSelector("p.sales-price")).getText();
65         String auxProduto3 = Produto3.replace("R$ ", "");
66         int valor3 = Integer.parseInt(auxProduto3);
67         driver.findElement(By.xpath("//*[@id='btn-buy']")).click();
68         driver.findElement(By.xpath("//*[@id='btn-continue']")).click();
69
```

Fonte: Elaborado pelo autor

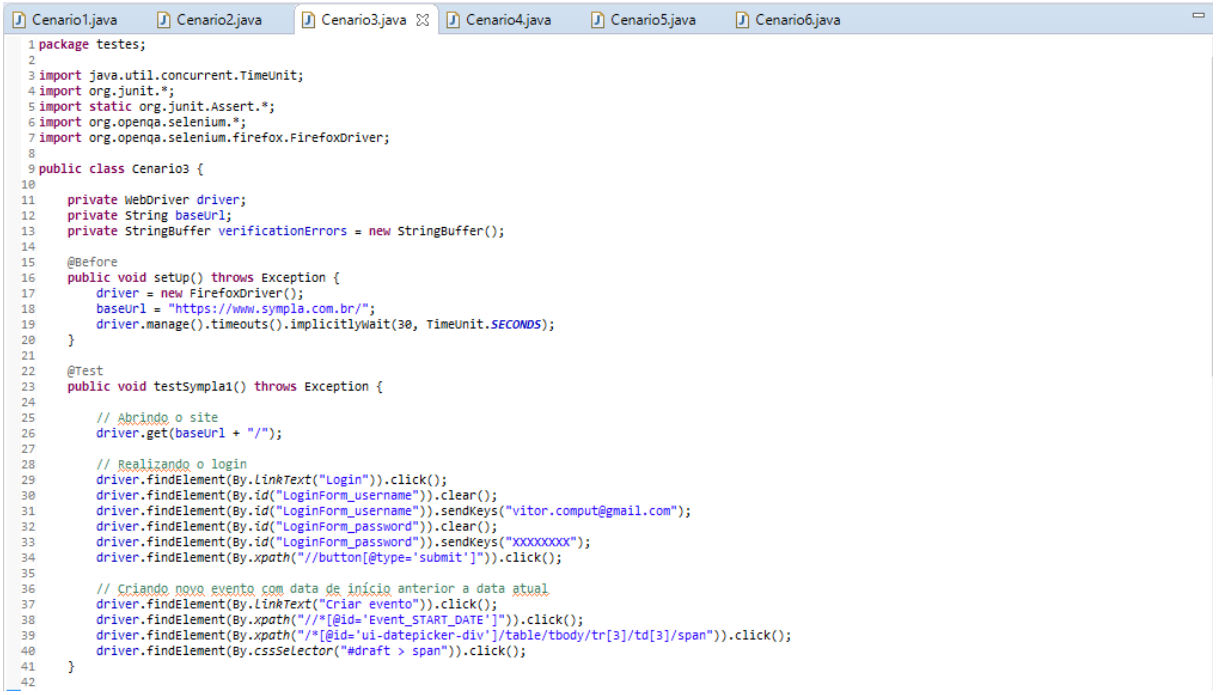
Figura 32 – Cenário 2 - Terceira parte do script

```
70          // Obtendo o valor total
71
72          String valorTotal = driver.findElement(By.id("total-amount")).getText();
73          String auxValorTotal = valorTotal.replaceAll("R$ ", "");
74          int valorT = Integer.parseInt(auxValorTotal);
75
76          // Realizando a soma dos valores dos produtos obtidos
77          int soma = valor1 + valor2 + valor3;
78
79          // Comparando o valor da soma com o valor gerado pelo carrinho
80          if (valorT != soma) {
81              driver.quit();
82          }
83
84      }
85
86      @After
87      public void tearDown() throws Exception {
88          // driver.quit();
89          String verificationErrorString = verificationErrors.toString();
90          if (!"".equals(verificationErrorString)) {
91              fail(verificationErrorString);
92          }
93      }
94  }
95
```

Fonte: Elaborado pelo autor

C SCRIPT DE TESTE – CENÁRIO 3

Figura 33 – Cenário 3 - Primeira parte do script

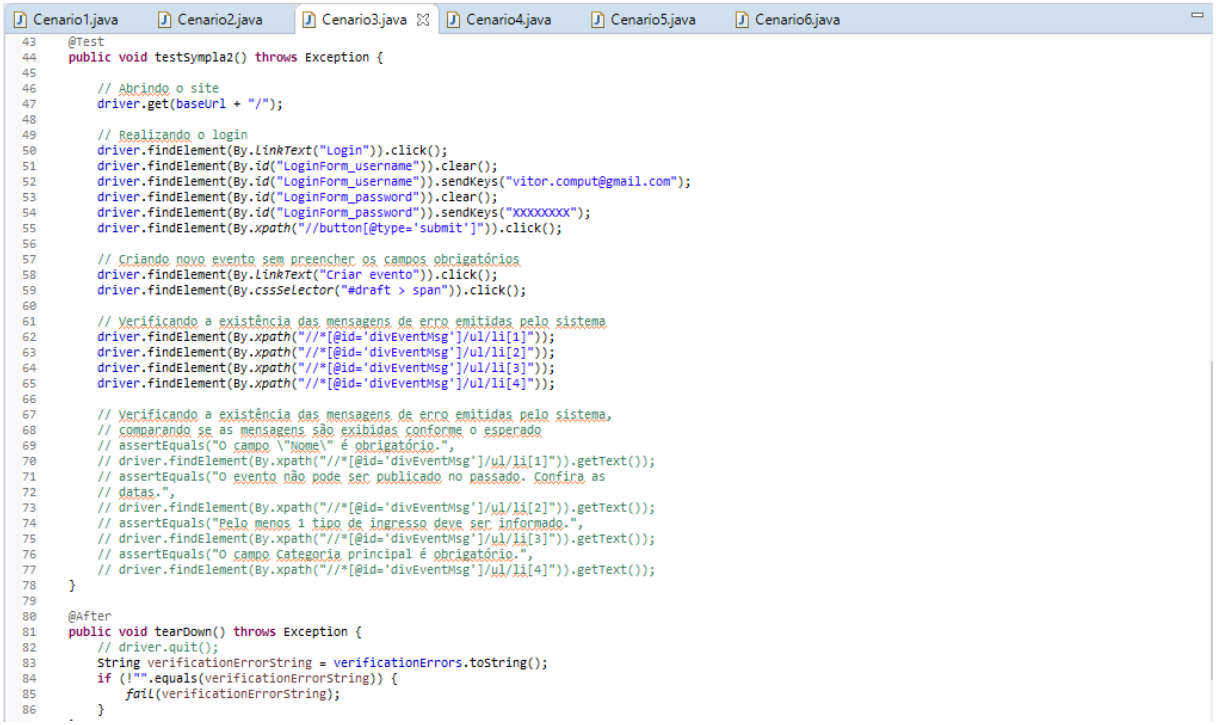


Cenario1.java | Cenario2.java | Cenario3.java | Cenario4.java | Cenario5.java | Cenario6.java

```
1  package testes;
2
3  import java.util.concurrent.TimeUnit;
4  import org.junit.*;
5  import static org.junit.Assert.*;
6  import org.openqa.selenium.*;
7  import org.openqa.selenium.firefox.FirefoxDriver;
8
9  public class Cenario3 {
10
11     private WebDriver driver;
12     private String baseUrl;
13     private StringBuffer verificationErrors = new StringBuffer();
14
15     @Before
16     public void setUp() throws Exception {
17         driver = new FirefoxDriver();
18         baseUrl = "https://www.sympla.com.br/";
19         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
20     }
21
22     @Test
23     public void testSympla1() throws Exception {
24
25         // Abrindo o site
26         driver.get(baseUrl + "/");
27
28         // Realizando o login
29         driver.findElement(By.linkText("Login")).click();
30         driver.findElement(By.id("LoginForm_username")).clear();
31         driver.findElement(By.id("LoginForm_username")).sendKeys("vitor.comput@gmail.com");
32         driver.findElement(By.id("LoginForm_password")).clear();
33         driver.findElement(By.id("LoginForm_password")).sendKeys("XXXXXXXX");
34         driver.findElement(By.xpath("//button[@type='submit']")).click();
35
36         // Criando novo evento com data de início anterior a data atual
37         driver.findElement(By.linkText("Criar evento")).click();
38         driver.findElement(By.xpath("//*[@id='Event_START_DATE']")).click();
39         driver.findElement(By.xpath("/*[@id='ui-datepicker-div']/table/tbody/tr[3]/td[3]/span")).click();
40         driver.findElement(By.cssSelector("#draft > span")).click();
41     }
42
```

Fonte: Elaborado pelo autor

Figura 34 – Cenário 3 - Segunda parte do script

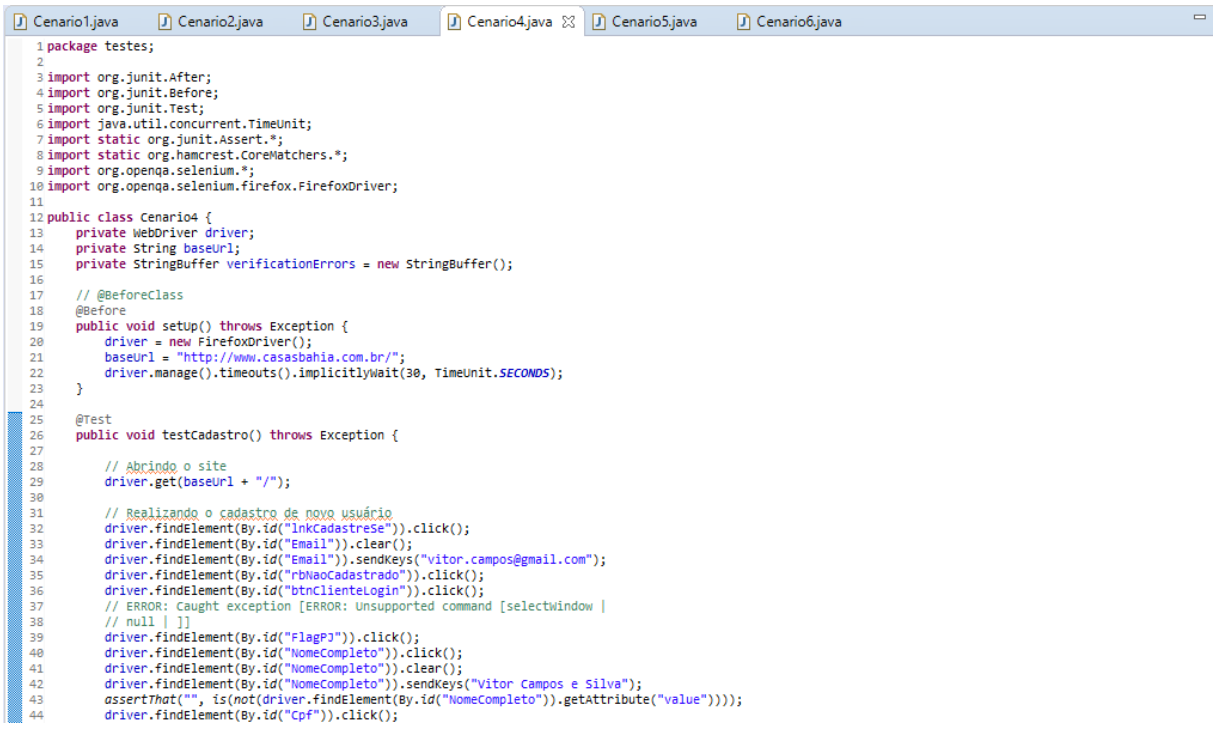



```
43  @Test
44  public void testSympla2() throws Exception {
45
46      // Abrindo o site
47      driver.get(baseUrl + "/");
48
49      // Realizando o login
50      driver.findElement(By.linkText("Login")).click();
51      driver.findElement(By.id("LoginForm_username")).clear();
52      driver.findElement(By.id("LoginForm_username")).sendKeys("vitor.comput@gmail.com");
53      driver.findElement(By.id("LoginForm_password")).clear();
54      driver.findElement(By.id("LoginForm_password")).sendKeys("XXXXXXXX");
55      driver.findElement(By.xpath("//button[@type='submit']")).click();
56
57      // Criando novo evento sem preencher os campos obrigatórios
58      driver.findElement(By.linkText("Criar evento")).click();
59      driver.findElement(By.cssSelector("#draft > span")).click();
60
61      // Verificando a existência das mensagens de erro emitidas pelo sistema
62      driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[1]"));
63      driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[2]"));
64      driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[3]"));
65      driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[4]"));
66
67      // Verificando a existência das mensagens de erro emitidas pelo sistema,
68      // comparando se as mensagens são exibidas conforme o esperado
69      // assertEquals("O campo \"Nome\" é obrigatório.",
70      // driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[1]")).getText());
71      // assertEquals("O evento não pode ser publicado no passado. Confira as
72      // datas.",
73      // driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[2]")).getText());
74      // assertEquals("Pelo menos 1 tipo de ingresso deve ser informado.",
75      // driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[3]")).getText());
76      // assertEquals("O campo Categoria principal é obrigatório.",
77      // driver.findElement(By.xpath("//*[@id='divEventMsg']/ul/li[4]")).getText());
78  }
79
80  @After
81  public void tearDown() throws Exception {
82      // driver.quit();
83      String verificationErrorString = verificationErrors.toString();
84      if (!"".equals(verificationErrorString)) {
85          fail(verificationErrorString);
86      }
```

Fonte: Elaborado pelo autor

D SCRIPT DE TESTE – CENÁRIO 4

Figura 35 – Cenário 4 - Primeira parte do script



Cenario1.java | Cenario2.java | Cenario3.java | Cenario4.java | Cenario5.java | Cenario6.java

```
1  package testes;
2
3  import org.junit.After;
4  import org.junit.Before;
5  import org.junit.Test;
6  import java.util.concurrent.TimeUnit;
7  import static org.junit.Assert.*;
8  import static org.hamcrest.CoreMatchers.*;
9  import org.openqa.selenium.*;
10 import org.openqa.selenium.firefox.FirefoxDriver;
11
12 public class Cenario4 {
13     private WebDriver driver;
14     private String baseUrl;
15     private StringBuffer verificationErrors = new StringBuffer();
16
17     // @BeforeClass
18     @Before
19     public void setUp() throws Exception {
20         driver = new FirefoxDriver();
21         baseUrl = "http://www.casasbahia.com.br/";
22         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
23     }
24
25     @Test
26     public void testCadastro() throws Exception {
27
28         // Abrindo o site
29         driver.get(baseUrl + "/");
30
31         // Realizando o cadastro de novo usuário
32         driver.findElement(By.id("lnkCadastreSe")).click();
33         driver.findElement(By.id("Email")).clear();
34         driver.findElement(By.id("Email")).sendKeys("vitor.campos@gmail.com");
35         driver.findElement(By.id("rbNaoCadastrado")).click();
36         driver.findElement(By.id("btnClienteLogin")).click();
37         // ERROR: Caught exception [ERROR: Unsupported command [selectWindow |
38         // null | ]]
39         driver.findElement(By.id("FlagPJ")).click();
40         driver.findElement(By.id("NomeCompleto")).click();
41         driver.findElement(By.id("NomeCompleto")).clear();
42         driver.findElement(By.id("NomeCompleto")).sendKeys("Vitor Campos e Silva");
43         assertThat("", is(not(driver.findElement(By.id("NomeCompleto")).getAttribute("value"))));
44         driver.findElement(By.id("Cpf")).click();
```

Fonte: Elaborado pelo autor

Figura 36 – Cenário 4 - Segunda parte do script

```
45          driver.findElement(By.id("Cpf")).clear();
46          driver.findElement(By.id("Cpf")).sendKeys("999.999.999-99");
47          assertThat("", is(not(driver.findElement(By.id("Cpf")).getAttribute("value"))));
48          driver.findElement(By.id("Telefone1DDDPF")).click();
49          driver.findElement(By.id("Telefone1DDDPF")).clear();
50          driver.findElement(By.id("Telefone1DDDPF")).sendKeys("31");
51          assertThat("", is(not(driver.findElement(By.id("Telefone1DDDPF")).getAttribute("value"))));
52          driver.findElement(By.id("Telefone1PF")).click();
53          driver.findElement(By.id("Telefone1PF")).clear();
54          driver.findElement(By.id("Telefone1PF")).sendKeys("999999999");
55          assertThat("", is(not(driver.findElement(By.id("Telefone1PF")).getAttribute("value"))));
56          driver.findElement(By.id("Telefone2DDDPF")).click();
57          driver.findElement(By.id("Telefone2DDDPF")).clear();
58          driver.findElement(By.id("Telefone2DDDPF")).sendKeys("31");
59          assertThat("", is(not(driver.findElement(By.id("Telefone2DDDPF")).getAttribute("value"))));
60          driver.findElement(By.id("Telefone2PF")).click();
61          driver.findElement(By.id("Telefone2PF")).clear();
62          driver.findElement(By.id("Telefone2PF")).sendKeys("99999999");
63          assertThat("", is(not(driver.findElement(By.id("Telefone2PF")).getAttribute("value"))));
64          driver.findElement(By.id("DataNascimentoDia")).clear();
65          driver.findElement(By.id("DataNascimentoDia")).sendKeys("10");
66          driver.findElement(By.id("DataNascimentoMes")).clear();
67          driver.findElement(By.id("DataNascimentoMes")).sendKeys("08");
68          driver.findElement(By.id("DataNascimentoAno")).clear();
69          driver.findElement(By.id("DataNascimentoAno")).sendKeys("1985");
70          driver.findElement(By.id("Sexo")).click();
71          assertThat("", is(not(driver.findElement(By.cssSelector("p.grp.fEmail > #Email")).getAttribute("value"))));
72
73          // Obtendo o valor do campo Email
74          String Email1 = driver.findElement(By.cssSelector("p.grp.fEmail > #Email")).getAttribute("value");
75
76          driver.findElement(By.id("ConfirmarEmail")).click();
77          driver.findElement(By.id("ConfirmarEmail")).clear();
78          driver.findElement(By.id("ConfirmarEmail")).sendKeys("vitor@gmail.com");
79          assertThat("", is(not(driver.findElement(By.id("ConfirmarEmail")).getAttribute("value"))));
80
81          // Obtendo o valor do campo Confirmar Email
82          String Email2 = driver.findElement(By.id("ConfirmarEmail")).getAttribute("value");
83
84          driver.findElement(By.cssSelector("p.grp.fSenha > #Senha")).clear();
85          driver.findElement(By.cssSelector("p.grp.fSenha > #Senha")).sendKeys("Teste123");
86
```

Fonte: Elaborado pelo autor

Figura 37 – Cenário 4 - Terceita parte do script

```
87          // Obtendo o valor do campo Senha
88          String Senha1 = driver.findElement(By.cssSelector("p.grp.fSenha > #Senha")).getAttribute("value");
89
90          driver.findElement(By.id("ConfirmarSenha")).clear();
91          driver.findElement(By.id("ConfirmarSenha")).sendKeys("Teste12");
92
93          // Obtendo o valor do campo Confirmar Senha
94          String Senha2 = driver.findElement(By.id("ConfirmarSenha")).getAttribute("value");
95
96          driver.findElement(By.id("ReceberAvisoEmail")).click();
97          driver.findElement(By.id("ReceberAvisoSms")).click();
98          driver.findElement(By.id("btnClienteSalvar")).click();
99
100         // Verificando se os dados dos campos Email e Confirmar Email são iguais
101         if (!Email1.equals(Email2)) {
102
103             // Verificando se o sistema exibe a mensagem de erro
104             assertEquals("Os endereços informados não são iguais.", driver
105                     .findElement(By.xpath("//*[@id='cliente_cadastro']/form/fieldset[4]/p[2]/span/span")).getText());
106
107         }
108
109         // Verificando se os dados dos campos Senha e Confirmar Senha são iguais
110         if (!Senha1.equals(Senha2)) {
111
112             // Verificando se o sistema exibe a mensagem de erro
113             assertEquals("As senhas informadas não são iguais.", driver
114                     .findElement(By.xpath("//*[@id='cliente_cadastro']/form/fieldset[4]/p[4]/span[1]/span")).getText());
115
116         }
117     }
118
119     // @AfterClass
120     @After
121     public void tearDown() throws Exception {
122         // driver.quit();
123         String verificationErrorString = verificationErrors.toString();
124         if (!"".equals(verificationErrorString)) {
125             fail(verificationErrorString);
126         }
127     }
128
129 }
130
```

Fonte: Elaborado pelo autor

E SCRIPT DE TESTE – CENÁRIO 5

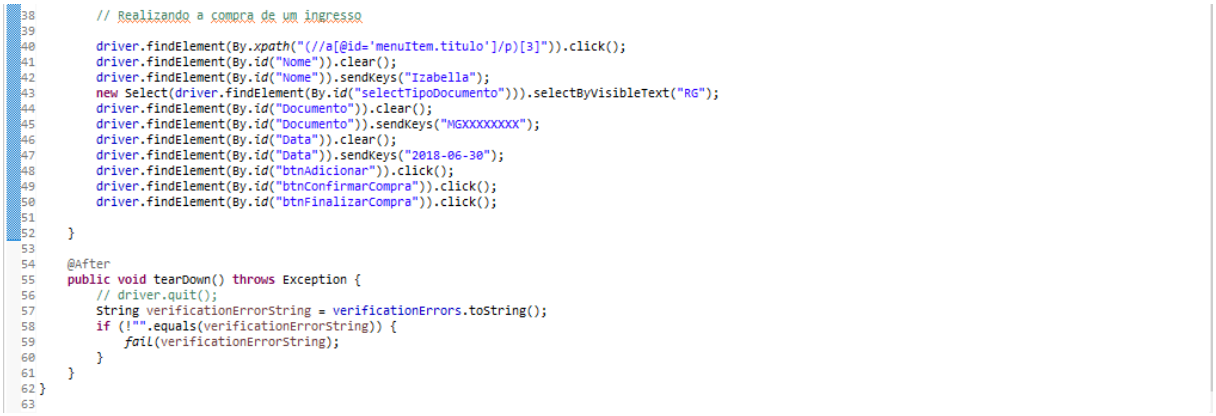
Figura 38 – Cenário 5 - Primeira parte do script

Cenario1.java | Cenario2.java | Cenario3.java | Cenario4.java | Cenario5.java | Cenario6.java

```
1  package testes;
2
3  import java.util.concurrent.TimeUnit;
4  import org.junit.*;
5  import static org.junit.Assert.*;
6  import org.openqa.selenium.*;
7  import org.openqa.selenium.firefox.FirefoxDriver;
8  import org.openqa.selenium.support.ui.Select;
9
10 public class Cenario5 {
11     private WebDriver driver;
12     private String baseUrl;
13     private StringBuffer verificationErrors = new StringBuffer();
14
15     @Before
16     public void setUp() throws Exception {
17         driver = new FirefoxDriver();
18         baseUrl = "http://localhost:4200/";
19         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
20     }
21
22     @Test
23     public void testClubesOnline() throws Exception {
24
25         // Abrindo o site
26         driver.get(baseUrl + "/");
27
28         // Realizando login no sistema
29         driver.findElement(By.id("inputEmail")).click();
30         driver.findElement(By.id("inputEmail")).clear();
31         driver.findElement(By.id("inputEmail")).sendKeys("isabela@gmail.com");
32         driver.findElement(By.id("inputPassword")).click();
33         driver.findElement(By.id("inputPassword")).clear();
34         driver.findElement(By.id("inputPassword")).sendKeys("123mudar");
35         driver.findElement(By.id("logar")).click();
36         // ERROR: Caught exception [unknown command [clickA]]
37
```

Fonte: Elaborado pelo autor

Figura 39 – Cenário 5 - Segunda parte do script

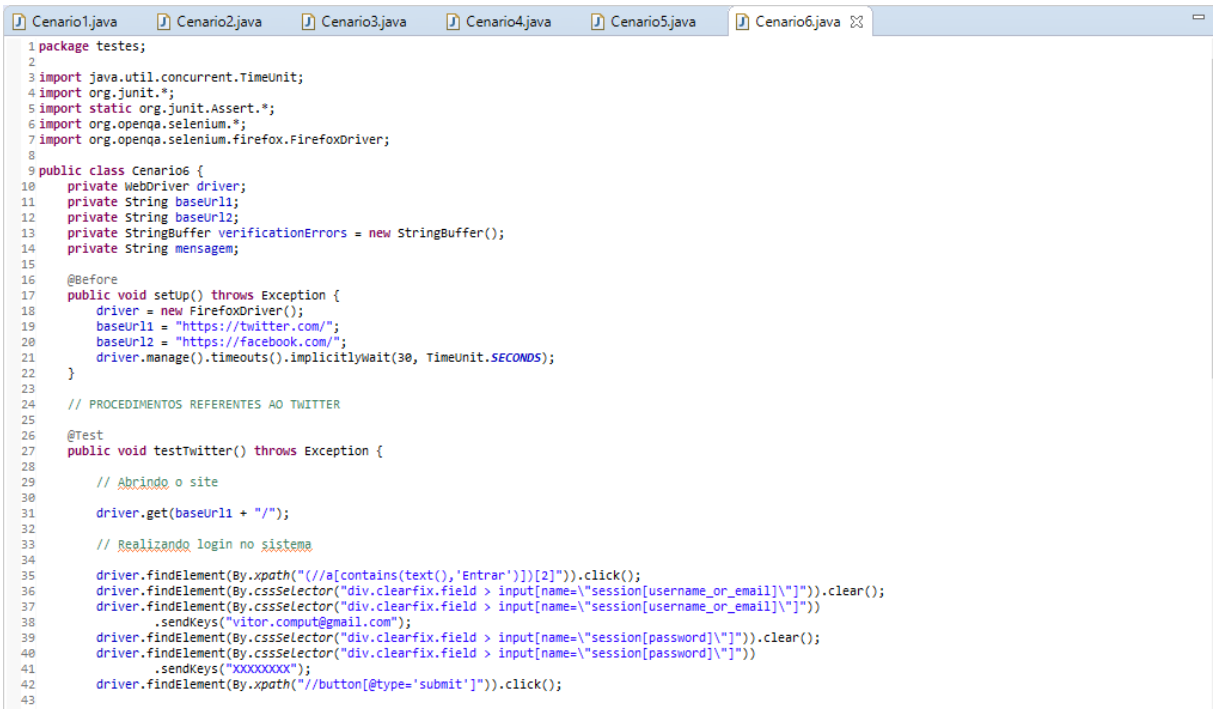


```
38         // Realizando a compra de um ingresso
39
40         driver.findElement(By.xpath("(//a[@id='menuItem.titulo']/p)[3]")).click();
41         driver.findElement(By.id("Nome")).clear();
42         driver.findElement(By.id("Nome")).sendKeys("Izabella");
43         new Select(driver.findElement(By.id("selectTipoDocumento"))).selectByVisibleText("RG");
44         driver.findElement(By.id("Documento")).clear();
45         driver.findElement(By.id("Documento")).sendKeys("MGXXXXXXXX");
46         driver.findElement(By.id("Data")).clear();
47         driver.findElement(By.id("Data")).sendKeys("2018-06-30");
48         driver.findElement(By.id("btnAdicionar")).click();
49         driver.findElement(By.id("btnConfirmarCompra")).click();
50         driver.findElement(By.id("btnFinalizarCompra")).click();
51
52     }
53
54     @After
55     public void tearDown() throws Exception {
56         // driver.quit();
57         String verificationErrorString = verificationErrors.toString();
58         if (!"".equals(verificationErrorString)) {
59             fail(verificationErrorString);
60         }
61     }
62 }
63
```

Fonte: Elaborado pelo autor

F SCRIPT DE TESTE – CENÁRIO 6

Figura 40 – Cenário 6 - Primeira parte do script



Cenario1.java | Cenario2.java | Cenario3.java | Cenario4.java | Cenario5.java | Cenario6.java

```
package testes;

import java.util.concurrent.TimeUnit;
import org.junit.*;
import static org.junit.Assert.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Cenario6 {
    private WebDriver driver;
    private String baseUrl1;
    private String baseUrl2;
    private StringBuffer verificationErrors = new StringBuffer();
    private String mensagem;

    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl1 = "https://twitter.com/";
        baseUrl2 = "https://facebook.com/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    }

    // PROCEDIMENTOS REFERENTES AO TWITTER

    @Test
    public void testTwitter() throws Exception {

        // Abrindo o site

        driver.get(baseUrl1 + "/");

        // Realizando login no sistema

        driver.findElement(By.xpath("(//a[contains(text(),'Entrar')])[2]")).click();
        driver.findElement(By.cssSelector("div.clearfix.field > input[name=\"session[username_or_email]\"]")).clear();
        driver.findElement(By.cssSelector("div.clearfix.field > input[name=\"session[username_or_email]\"]"))
                .sendKeys("vitor.comput@gmail.com");
        driver.findElement(By.cssSelector("div.clearfix.field > input[name=\"session[password]\"]")).clear();
        driver.findElement(By.cssSelector("div.clearfix.field > input[name=\"session[password]\"]"))
                .sendKeys("XXXXXXXX");
        driver.findElement(By.xpath("//button[@type='submit']")).click();

```

Fonte: Elaborado pelo autor

Figura 41 – Cenário 6 - Segunda parte do script

```
44          // Postando a mensagem desejada
45
46          driver.findElement(By.xpath("//*[@id='tweet-box-home-timeline']")).click();
47          driver.findElement(By.xpath("//*[@id='tweet-box-home-timeline']"))
48                  .sendKeys("Teste de Interconexão de Softwares - TCC");
49
50          //Armazendoa a mensagem desejada na variável "mensagem"
51
52          mensagem = driver.findElement(By.xpath("//*[@id='tweet-box-home-timeline']")).getText();
53
54          driver.findElement(By.xpath("//*[@id='timeline']/div[2]/div/form/div[3]/div[2]/button")).click();
55      }
56
57      // PROCEDIMENTOS REFERENTES AO FACEBOOK
58
59      @Test
60      public void testFacebook() throws Exception {
61
62          // Abrindo o site
63
64          driver.get(baseUrl2 + "/");
65
66          // Realizando login no sistema
67
68          driver.findElement(By.xpath("//*[@id='email']")).click();
69          driver.findElement(By.xpath("//*[@id='email']")).sendKeys("vitorcampossilva@hotmail.com");
70          driver.findElement(By.xpath("//*[@id='pass']")).click();
71          driver.findElement(By.xpath("//*[@id='pass']")).sendKeys("XXXXXXXX");
72          driver.findElement(By.id("loginbutton")).click();
73
74          // Verificando a existência da mensagem
75
76          driver.findElement(By.xpath("//*[@id='navItem_100001678468105']/a/div")).click();
77          assertEquals(mensagem, driver.findElement(By.xpath("//*[@id='js_ny']/div/p")).getText());
78      }
79
80      @After
81      public void tearDown() throws Exception {
82          // driver.quit();
83          String verificationErrorString = verificationErrors.toString();
84          if (!"".equals(verificationErrorString)) {
85              fail(verificationErrorString);
86          }
87      }
```

Fonte: Elaborado pelo autor