INSTITUTO FEDERAL DE MINAS GERAIS
CAMPUS SÃO JOÃO EVANGELISTA

ARIANA CARLA RODRIGUES PEREIRA
DIVA KAYLA DE OLIVEIRA
GRACILANE ELINAIDE DE LIMA

ESTUDO DE CASO DA FERRAMENTA DE TESTE SELENIUM APLICADO AO MÓDULO DE PEDIDOS ONLINE DO SISTEMA FALCONFV

São João Evangelista
2013

ARIANA CARLA RODRIGUES PEREIRA

DIVA KAYLA DE OLIVEIRA

GRACILANE ELINAIDE DE LIMA

ESTUDO DE CASO DA FERRAMENTA DE TESTE SELENIUM APLICADO AO MÓDULO DE PEDIDOS ONLINE DO SISTEMA FALCONFV

Monografia apresentada ao Curso de Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus São João Evangelista, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Me. Rosinei Soares de Figueiredo
Coorientador: Esp. Bruno de Souza Toledo

São João Evangelista

2013

FICHA CATALOGRÁFICA
Elaborada pelo Serviço Técnico da Biblioteca do
Instituto Federal Minas Gerais – Campus São João Evangelista

P436e PEREIRA, Ariana Carla Rodrigues, 1989-

Estudo de caso da ferramenta de teste Selenium aplicado ao módulo de pedidos online do Sistema Falconfv./ Ariana Carla Rodrigues Pereira; Diva Kayla de Oliveira; Gracilane Elinaide de Lima. São João Evangelista, MG: IFMG - Campus São João Evangelista, 2013.

88 p.: il.

Trabalho de Conclusão de Curso - TCC (graduação) apresentado ao Instituto Federal Minas Gerais – Campus São João Evangelista – IFMG, Curso de Bacharelado em Sistemas de Informação, 2013.

Orientador: Prof. Me. Rosinei Soares de Figueiredo

Coorientador: Prof. Esp. Bruno de Souza Toledo

1. Sistema de informação. 2. Software livre . 3. Software. I. Instituto Federal Minas Gerais – Campus São João Evangelista. Curso de Bacharelado em Sistemas de Informação. II. Título.

CDD 006.6

ARIANA CARLA RODRIGUES PEREIRA

DIVA KAYLA DE OLIVEIRA

GRACILANE ELINAIDE DE LIMA

ESTUDO DE CASO DA FERRAMENTA DE TESTE SELENIUM APLICADO AO MÓDULO DE PEDIDOS ONLINE DO SISTEMA FALCONFV

Monografia apresentada ao Curso de Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus São João Evangelista, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Me. Rosinei Soares de Figueiredo (Orientador) – IFMG-SJE

Esp. Bruno de Souza Toledo (Coorientador) – IFMG-SJE

Me. Ronan Dutra Mendonça – IFMG-SJE

São João Evangelista, 12 de Novembro de 2013.

AGRADECIMENTOS

Agradecemos primeiramente ao nosso pai superior Deus, por todas as graças proporcionadas em nossas vidas.

Aos nossos pais que nos deram a vida, pelos incentivos e pela transmissão de valores vitais para a formação de nosso caráter e aos demais familiares que acompanharam nossa jornada.

Ao professor Rosinei Soares de Figueiredo, nosso orientador, por seu auxílio e dedicação, e por amenizar a nossa caminhada para a conquista desse objetivo através das correções descontraídas, o que tornou a formulação do trabalho mais prazerosa e menos cansativa.

Ao coordenador do curso Bruno de Souza Toledo, por nos prestar assistências em inúmeras questões cotidianas envolvendo o desenvolvimento da pesquisa.

A empresa Eagle Tecnologia da Informação pelas colaborações fundamentais para a construção do trabalho.

Aos amigos, pela compreensão e apoio nos momentos mais difíceis.

A nós pela dedicação prestada, por vencer o cansaço, pelo espírito de equipe, pela boa convivência e pelo enriquecimento da amizade.

Lista de Abreviaturas e Siglas

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
APP	Application
ASP	Active Server Pages
C#	C Sharp
DHTML	Dynamic HyperText Markup Language
DOM	Document Object Model
DSL	Domain Specific Language
ERP	Enterprise Resource Planning
FalconFV	Falcon Força de Vendas
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDE	Integrated Development Environment
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
ITESTE	Instituto de Teste de Software
J2EE	Java Platform, Enterprise Edition
J2SE	Java Platform, Standard Edition
JSON	JavaScript Object Notation
LTDA	Limitada
NSTI	Núcleo Setorial de Tecnologia de Informação
PHP	*Hypertext Preprocessor*
RC	Remote- Control
SWEBOK	Software Engineering Body of Knowledge
SWT	Standard Widget Toolkit
UML	Unified Modeling Language
URL	Uniform Resource Locator
V&V	Verificação e Validação
XML	eXtensible Markup Language
XPath	XML Path Language

RESUMO

O desenvolvimento de *software* é composto por um conjunto de etapas, cada uma com metodologias e objetivos diferentes, dentre estas etapas existe uma de grande relevância conhecida como Teste de Software. O Teste de *Software* refere-se à busca por possíveis falhas no *software* em construção ou já concluído, a fim de possibilitar a correção destas falhas. Consequentemente, a etapa de teste é fundamental para a busca da qualidade de um determinado produto de *software*. Pela relevância que tem para o processo de desenvolvimento, a fase de testes é realizada utilizando um conjunto de técnicas específicas, uma das técnicas mais importantes é denominada Teste Funcional, esta técnica tem por finalidade verificar as funções do sistema, ou seja, preocupa-se em testar os requisitos funcionais implementados. Ainda assim, mesmo valendo-se de técnicas bem definidas, há uma grande dificuldade em se aplicar os testes nos *softwares*, tarefa que exige grandes quantidades de tempo e recursos diversos. Então, para facilitar a execução desta atividade, foram criadas ferramentas das quais os testadores podem fazer uso para automatizar as tarefas de teste. O *Selenium* é um exemplo de ferramenta *open source* que proporciona esta automação, seu uso tem crescido bastante, colocando-o entre as principais ferramentas da área. Para complementar, o Selenium, possibilita a implementação de testes em várias linguagens de programação, como PHP, C# e Java. Este trabalho apresenta um estudo sobre a ferramenta *Selenium* e demonstra sua utilização em testes funcionais, o objetivo é apresentar o processo de aplicação deste tipo de teste de forma automática com a referida ferramenta. Tais testes serão aplicados no módulo de Pedidos *Online* do sistema *FalconFV*, desenvolvido pela empresa Eagle Tecnologia LTDA. A partir dos testes aplicados e seus resultados, serão levantadas e publicadas as boas práticas relacionadas ao uso da ferramenta.

Palavras-chave: Ferramenta de teste. Selenium. Teste web. Automação.

ABSTRACT

The software development consists of a set of steps, each with different objectives and methods, among these steps there is a great relevance known as Software Test. The Software Testing refers to the search for potential faults in the software under construction or already completed, to allow the correction of these faults. Consequently, the test is fundamental to the pursuit of quality of a particular software product. The relevance which is in the process of development, testing is performed using a specific set of techniques called Functional Test, this technique is intended to check the functions of the system, or worrying in testing the functional requirements implemented. Still, even resorting to techniques well defined, there is a great difficulty in applying the tests in software, a task that requires large amounts of time and many resources. Then, to facilitate the implementation of this activity were created tools which testers can use to automate tasks test. Selenium is an example of open source tool that provides this automation, its use has grown considerably, placing it among the main tools of the area. To complement, the Selenium enables the implementation of tests in various programming languages like PHP, C# and Java. This work presents a study on the Selenium tool and demonstrates its use in functional testing, the goal is to present the application process this type of test automatically with that tool. Such tests will be applied in the module Online Ordering system FalconFV, developed by Eagle Technology LTD. From tests applied and their results shall be lifted and published a good practice relating to the use of the tool.

Keywords: Testing tool. Selenium. Test web. Automation.

SUMÁRIO

1 INTRODUÇÃO

A Engenharia de *Software*[1] passou por fortes mudanças nas últimas décadas ocasionando seu progresso. Esta evolução deu-se, principalmente, pela necessidade de produtos de *software* mais estáveis e seguros, evidenciando-se no período que sucedeu a chamada Crise do *Software*[2].

Durante o período que se caracterizou como Crise do *Software*, houve um aumento rápido na demanda por *software*s, porém as soluções construídas não atendiam aos requisitos de qualidade necessários. A explicação para o fato estava no aumento da complexidade dos problemas a serem resolvidos e na inexistência de técnicas e métodos sistemáticos de desenvolvimento de *software*, fatos esses ocorridos no final dos anos 60 (SOMMERVILLE, 2011).

Essa evolução repercutiu também no Desenvolvimento *Web*, a área da Engenharia de *Software* que trata, sobretudo, da construção de *software*s que funcionem sobre a plataforma *web*. Muitas melhorias na arquitetura do *software*[3] e na qualidade das aplicações tornaram-se possíveis no decorrer do tempo, configurando grandes oportunidades de negócio para as empresas de *software* que queiram oferecer serviços aos usuários em escala mundial (GINIGE e MURUGESAN, 2001a).

O processo de desenvolvimento de *software* é composto por diversas fases, sendo as mais tradicionais: Levantamento de Requisitos, Análise de Requisitos, Projeto, Implementação, Testes, Implantação e Manutenção. Cada uma destas fases apresenta um grau de diferente importância visando à qualidade final do projeto a ser desenvolvido. Este trabalho aborda, sobretudo, a fase de testes das aplicações.

Segundo Crosby 1979 e Juran 1988 citados por Guerra e Colombo 2009, qualidade de *software* é a totalidade das características que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas esperadas de todo *software* desenvolvido profissionalmente. Uma das características é a ausência de *bugs*[4]

---

[1] É uma disciplina de engenharia relacionada com todos os aspectos da produção de *software*. (SOMMERVILLE 2011)

[2] O termo expressava as dificuldades do desenvolvimento de software frente ao rápido crescimento da demanda por *software*, da complexidade dos problemas a serem resolvidos e da inexistência de técnicas estabelecidas para o desenvolvimento de sistemas que funcionassem adequadamente ou pudessem ser validados. (NETO 2010)

[3] Estrutura dos componentes de um programa/sistema, seus inter-relacionamentos, princípios e diretrizes guiando o projeto e evolução ao longo do tempo. (GARLAN *apud* FILHO 2006)

[4] Termo é usado em informática designando a ocorrência de erros ou defeitos.

durante a utilização. O uso de técnicas, métodos e ferramentas minimizam as possíveis falhas, mas não impedem que ainda ocorram erros no produto. Desta forma, é necessária a utilização de testes que ajudem a identificar possíveis erros, para que então tais erros possam ser corrigidos. Além disso, o teste representa a revisão final das especificações, do projeto e dos códigos gerados, caracterizando-se, portanto, como um elemento crítico para a qualidade dos sistemas (PRESSMAN 2010; MYERS 2004).

Ressalta-se, porém, que o processo de desenvolvimento de aplicações *web*[5] possui particularidades que o distingue do processo dos sistemas *desktop*[6]. As aplicações *web* são desenvolvidas por pequenas equipes com conhecimento em diversas tecnologias, com prazos reduzidos e necessidade maior de flexibilidade a frequentes mudanças de requisitos. Tais características acabam por exigir que os testes das aplicações sejam feitos de forma rápida, sem prejuízo dos padrões de qualidade necessários (GINIGE e MURUGESAN, 2001a).

Ante a este contexto, surgiram diversas ferramentas de testes destinadas a inspecionar, de maneira automática ou semiautomática, os sistemas *web*. Tais ferramentas têm como intuito tornar a execução dos testes mais ágeis e eficientes, facilitando a atividade que feita manualmente pode se tornar um trabalho repetitivo e extremamente cansativo. A automação de testes facilitou esta atividade, com isso, testes manuais que levariam horas para serem concluídos, podem ser executados em minutos (FEWSTER *apud* OLIVEIRA *et al*. 2007).

Este trabalho aborda a ferramenta *Selenium* que se caracteriza por ser *open source* e apresenta crescimento no mercado de teste, onde atualmente é amplamente utilizada pelos profissionais da área. A ferramenta é disponibilizada em três versões: o *Selenium* IDE (*Integrated Development Environment*), o *Selenium* RC (*Remote Control*) e o *Selenium* Grid, cada uma com suas particularidades a serem apresentadas posteriormente, no decorrer do trabalho.

Diante do paradigma da automatização de testes para aplicações *web* e do cenário no qual o *Selenium* se apresenta como a principal ferramenta para este fim, define-se a pergunta problema deste trabalho da seguinte forma: como utilizar a

[5] *Softwares* projetados para utilização através de um navegador na *Internet* ou em redes privadas (*Intranet*)
[6] Sistemas desenvolvidos para execução e instalação de arquivos localmente podendo ter banco de dados em rede

ferramenta *Selenium* para aplicar testes predefinidos em sistemas *web*?

O presente trabalho tem por finalidade explorar a ferramenta *Selenium*, através de sua utilização para aplicar testes predefinidos ao módulo de Pedidos *Online* do sistema *FalconFV*[7] , para mostrar, através disso, o processo de automatização de testes com a referida ferramenta. Para alcançar o objetivo geral do presente trabalho necessita-se:

a) Estudar, de maneira geral, e apresentar as teorias, métodos e ferramentas relacionadas a teste de *software*, com foco direcionado em teste de sistemas *web*;
b) Estudar detalhadamente e apresentar a ferramenta de automatização de testes *Selenium*, abordando seus princípios e funcionalidades;
c) Executar o estudo de caso que será a aplicação dos recursos da ferramenta *Selenium* no teste do *software FalconFV*, especificadamente no módulo de Pedidos *Online*;
d) Enriquecer o acervo bibliográfico sobre a ferramenta *Selenium*.

Tendo em vista que o *Selenium* é uma novidade e as informações sobre o mesmo são escassas, o presente trabalho é uma fonte de referência da ferramenta, divulgando suas funcionalidades e demonstrando sua utilização para testes em sistemas *web*. Também reforça a importância de testes no processo de desenvolvimento de *softwares*, principalmente em aplicações *web*, sendo aplicado na prática através do módulo de Pedidos *Online* do sistema *FalconFV* desenvolvido pela Eagle Tecnologia da Informação LTDA, servindo assim, de incentivo para gestores empresariais da indústria de *softwares* utilizarem testes em seus programas.

O trabalho contempla o estudo sobre a ferramenta *Selenium* mostrando suas principais funcionalidades, e descreve as versões RC e *Grid* e detalha a versão IDE desta ferramenta com o intuito de mostrar as particularidades de cada uma. Apresenta também uma indicação de boas práticas de desenvolvimento, voltadas para teste em sistemas *web* e o histórico de desenvolvimento do módulo de Pedidos *Online* do sistema *FalconFV* contendo suas principais características e funções.

[7] Sistema completo para apoio a força de vendas, desenvolvido pela empresa Eagle Tecnologia da Informação LTDA.

O restante dos capítulos da monografia está definido conforme a estrutura descrita abaixo:

O **Capítulo 2** apresenta os principais conceitos, técnicas, métodos e ferramentas necessárias para o desenvolvimento deste trabalho. O **Capítulo 3** contém a ideia central de trabalhos que abordam a automação de testes em sistemas *web*. No **Capítulo 4** é descrita a metodologia empregada para o desenvolvimento da pesquisa. No **Capítulo 5** estão os procedimentos utilizados e os resultados obtidos na realização deste trabalho. O **Capítulo 6** expõe a análise dos resultados da pesquisa. No **Capítulo 7** está a conclusão do trabalho, bem como, a sugestão de trabalhos futuros relacionados ao contexto desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por finalidade a apresentação dos principais conceitos, técnicas, métodos e ferramentas necessárias para o desenvolvimento deste trabalho. A seção 2.1 refere-se à conceituação de s*oftware*. A seção 2.2 descreve a Engenharia de *Software* especificando seus principais aspectos, enquanto a seção 2.3 foca a engenharia voltada para Desenvolvimento *Web*. Na seção 2.4 encontra-se uma breve descrição das características gerais do sistema *FalconFV*. A seção 2.5 define teste de *software* bem como apresenta os estágios e técnicas fundamentais e a seção 2.6 traz a ideia de automação dos testes. Por fim seção 2.7 expõe a ferramenta de teste *Selenium*.

2.1 Software

Segundo Sommerville (2011) o *software* é o conjunto de vários artefatos e não apenas o código fonte como pode parecer. O termo *software* foi usado pela primeira vez no ano de 1958, em um artigo escrito pelo cientista americano John Wilder Tukey (HIRAMA, 2012).

Software é uma sequência de instruções a serem seguidas e/ou executadas, na manipulação, redirecionamento ou modificação de um dado/informação. *Software* também é o nome dado ao comportamento exibido por essa sequência de instruções quando executada em um computador ou máquina semelhante, além de um produto desenvolvido pela engenharia de *software*, e inclui não só o programa de computador propriamente dito, mas também manuais e especificações (NSTI, 2013).

2.2 Engenharia de Software

A Engenharia de *Software* é um ramo da engenharia cujo foco é o desenvolvimento dentro de custos adequados de sistemas de *software* de alta qualidade (SOMMERVILLE, 2011).

Engenharia de *Software* possui um extenso material e várias técnicas para sucesso de um *software*. Como o desenvolvimento de *software* é dividido em fases, o

IEEE[8] montou um comitê que criou um guia chamado S*WEB*OK[9] desmembrando a engenharia de *software* em dez áreas de conhecimento descritas no Quadro 1.

Quadro 1 – Áreas de conhecimento da Engenharia de *Software*

ÁREAS DE CONHECIMENTO	DESCRIÇÃO
Requisitos de *Software*	São objetivos ou restrições estabelecidas por clientes e usuários que definem as suas diversas propriedades do sistema.
Projeto (Design) de *Software*	Encarrega-se de transformar os resultados da Análise de Requisitos em um documento ou conjunto de documentos capazes de serem interpretados diretamente pelo programador.
Construção de *Software*	Enfoca mais a codificação e a depuração, é uma atividade muito complexa. Ela toma a maior parte do desenvolvimento de um projeto de *software*.
Teste de *Software*	É a investigação do *software* a fim de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar.
Manutenção de *software*	É o processo de melhoria e otimização de um *software* já desenvolvido (versão de produção), como também reparo de defeitos. A manutenção do *software* é uma das fases do processo de desenvolvimento de *software*.
Gerência de Configuração de *Software*	Fornece o apoio para o desenvolvimento de *software*. Suas principais atribuições são o controle de versão, o controle de mudança e a auditoria das configurações.
Gerência de Engenharia de *Software*	Pode ser definido como a aplicação de atividades de acompanhamento, planejamento e controle que garantem um desenvolvimento e uma manutenção sistemática dos projetos de *software*s.
Processos de Engenharia de *Software*	Formado por um conjunto de passos de processo parcialmente ordenados, relacionados com artefatos, pessoas, recursos, estruturas organizacionais e restrições, tendo como objetivo produzir e manter os produtos de *software* finais requeridos.
Ferramentas e Métodos de Engenharia de *Software*	Ferramentas de *software* automatizam o processo de engenharia de *software*, os métodos impõem estrutura sobre a atividade de desenvolvimento e manutenção de *software* com o objetivo de torna-la sistemática e mais propensa ao sucesso.
Qualidade de *Software*	É um conjunto de atividades relacionadas com garantia de qualidade de *software*, entre estas as atividades de verificação e validação.

Fonte: Adaptado de S*WEB*OK, 2004

2.3 Engenharia de Software para Web

A evolução e o rápido crescimento das aplicações *web* têm afetado todos os aspectos de vidas das pessoas (GINIGE E MURUGESAN, 2001). As aplicações *web* representam a evolução do *software* convencional, isto motivou pesquisas de como manter os princípios da Engenharia de *Software* convencional no Desenvolvimento

[8] Instituto de Engenheiros Eletricistas e Eletrônicos
[9] Software Engineering Body of Knowledge

Web (PRESSMAN, 2010). De forma similar à engenharia do *software* convencional, seu foco está em como desenvolver uma aplicação correta e completa, de acordo com os requisitos do usuário. O diferencial está no fato de que esta deve ser desenvolvida no contexto de um projeto que deve considerar a infraestrutura *Web* para sua execução e disponibilização (SOMMERVILLE, 2011).

2.4 Teste de Software

Entende-se por teste de *software*, o(s) procedimento(s) projetado(s) para avaliar se o *software* corresponde ao pretendido, não demonstrando comportamento inesperado e indesejável ao cliente (MYERS, 2004). Teste de *software* se caracteriza como uma atividade complexa que tem como um dos objetivos descobrir a ocorrência de erros e segundo Pressman (2010), representam até 40% do esforço no processo de desenvolvimento.

Entre diversos fatores que circundam qualidade, a satisfação do cliente é um fator de grande relevância. Com isso, confirmar que o *software* reage conforme os requisitos do cliente se compõem como uma finalidade dos testes, tornando a atividade de teste um elemento importante para garantia de qualidade de *software* (MALDONADO *apud* SOUZA et. al. 2003; PRESSMAN, 2010).

A atividade de teste está inserida em um contexto mais amplo identificado como Verificação e Validação (V&V). A verificação consiste em garantir que o *software* funciona corretamente e validação em confirmar que o mesmo corresponde às exigências do cliente (PRESSMAN, 2010). Dentre as técnicas de verificação e validação, a atividade de teste é uma das mais utilizadas, sendo assim um requisito para afirmar que um *software* é confiável.

2.4.1 Estágios de teste

Os testes se distinguem em vários estágios durante o desenvolvimento de um *software*. Estes estágios ou níveis vão desde testar partes de um sistema a testar sistemas completos. Tais níveis estão descritos de diferentes formas na literatura e a divisão mais comum desses estágios está discriminada de maneira simplificada a seguir no Quadro 2:

Quadro 2 – Estágio de testes

ESTÁGIO DE TESTE	DEFINIÇÃO
Teste de Unidade	Também chamado de teste de componente, testa os componentes do *software* individualmente, verificando se operam corretamente (SOMMERVILLE, 2011).
Teste de Integração	Teste realizado depois da integração dos componentes, com o propósito de testar se o relacionamento entre os componentes funciona perfeitamente após a união.
Teste de Sistemas	Testa o sistema como um todo, observando todas as funcionalidades e a sua precisão.
Teste de Aceitação	Um dos últimos testes feitos antes da implantação do sistema com o propósito de obter a aprovação do cliente em todos os aspectos.
Teste de Regressão	Caracteriza-se pela repetição dos testes realizados anteriormente em todos os módulos ou apenas nos módulos interligados, após alguma alteração no *software* a fim de descobrir a presença de algum erro posteriormente à modificação.

Fonte: Elaborado pelas autoras

A cada estágio de teste são aplicadas técnicas de acordo com o objetivo e aspecto a ser testado.

2.4.2 Técnicas de Teste

As técnicas de testes são utilizadas visando que o teste seja bem sucedido, sendo este conceituado como aquele que consegue determinar situações nas quais o *software* falhe. O teste funcional (caixa preta) e o teste estrutural (caixa branca) são as duas principais técnicas nas quais os testes são baseados.

2.4.2.1 Teste Funcional

Nessa técnica desconsidera-se o código e verifica-se as funções do sistema, ou seja, testam os requisitos funcionais. Este teste trata o *software* como uma caixa de conteúdo desconhecido, apenas vista exteriormente, por isso a denominação caixa preta (SILVA, 2011).

Os testes funcionais tendem a ser realizados nas últimas etapas do processo de teste, visando descobrir funções incorretas ou ausentes, erros de interface, erros nas estruturas de dados ou acesso ao banco de dados externos, erros de desempenho e erros de inicialização e término (PRESMAN, 2010).

Existem inúmeras ferramentas destinadas à técnica funcional. Algumas destas

são: *Badboy Web Tool*, *Canoo WebTest* e *Sikuli*. Abaixo há uma sucinta descrição de tais ferramentas.

Badboy Web Tool é uma poderosa ferramenta projetada para apoiar testes funcionais no desenvolvimento de aplicações dinâmicas complexas. Utiliza a técnica captura/reprodução na criação dos *scripts* de testes, pode ser executada juntamente com os navegadores *Internet Explorer* e *Firefox* e possibilita a emissão de relatórios detalhados. Está disponível em uma versão simplificada gratuita e outra mais completa paga (BADBOY..., 2012).

O *Canoo WebTest* é uma ferramenta *open source* para testes automatizados de aplicações *web*. O diferencial desta ferramenta é que os testes podem ser feitos com arquivos XML ou como códigos escritos na linguagem Groovy (CANOO..., 2007).

Sikuli é uma ferramenta visual para automatizar e testar interfaces gráficas (GUI - *Graphical User Interface*), utilizando imagens para identificar e controlar seus componentes. A solução *Sikuli* inclui: *Sikuli Script* e *Sikuli IDE* e é liberado sob a licença do Instituto de Tecnologia de Massachusetts. O *Sikuli* possibilita colocar diretamente a imagem da interface gráfica do usuário de seu computador ao invés de comandos, simplificando enormemente sua codificação (ITESTE, 2011).

Além destas ferramentas, o *Selenium* se enquadra nesta categoria. Porém, é explicado em uma seção específica por apresentar grande importância para o desenvolvimento deste estudo.

2.4.2.2 Teste Estrutural

A técnica de teste estrutural ou caixa branca, baseia-se no conhecimento da estrutura interna da implementação analisando o código fonte e testando os caminhos lógicos do *software*. Executar testes estruturais com rigor induz a pensar na obtenção de um *software* totalmente correto, contudo os testes baseados na implementação são exaustivos e por mais simples que seja o sistema a definição de todos os caminhos lógicos é impraticável (PRESSMAN, 2010).

Dentre as várias ferramentas para testes estruturais estão o *WebLoad* e *Apache JMeter*. Abaixo há uma breve descrição sobre tais ferramentas.

O *WebLoad* é uma ferramenta *open source* específica para Testes de Carga e Performance, podendo ser facilmente usada por qualquer analista de testes. É uma

ferramenta patrocinada pela *Radview Software* e atualmente está na versão 8.1.0.118.00 (CAMPOS, 2013).

A aplicação *Apache JMeter* é um *software desktop open source*, desenvolvido em Java, originalmente projetado para testar aplicações *web*. *Apache JMeter* pode ser usado para testar o desempenho tanto em recursos estáticos como dinâmicos (arquivos, *Servlets*, *scripts Perl*, *Java* objetos, bases de dados e consultas, servidores FTP e muito mais). Ele pode ser usado para simular uma carga pesada num servidor, numa rede ou em um objeto para testar a sua resistência ou para analisar o desempenho geral sob diferentes tipos de carga (APACHE *SOFTWARE* FOUNDATION, 2013).

A técnica funcional é melhor aplicada sobre componentes de sistema e realizada por uma equipe de testes, enquanto a abordagem estrutural é melhor aplicada a componentes individuais ou a coleções de componentes dependentes e realizada pela equipe de desenvolvimento (SIXPENCE *et al*, 2011).

2.5 Automação de testes

A realização dos testes é um processo com alto nível de complexidade, que demanda muito tempo e possui custo significativo. Esses aspectos constituem um desafio para a implantação de testes de maneira efetiva nas empresas. A execução de testes manualmente, apesar de descobrir erros, pode não ser suficientemente eficaz, ocorrendo uma verificação superficial.

> Automação de teste é o uso de *software* para controlar a execução do teste de *software*, a comparação dos resultados esperados com os resultados reais, a configuração das precondições de teste e outras funções de controle e relatório de teste (SILVA, 2011, p.19).

Silva (2011) salienta que com a utilização de ferramentas para apoiar as atividades de teste, ocorre a redução do tempo de execução e os testes podem ser repetidos sempre que necessário sem representar uma tarefa árdua. Também expõe o custo benefício da automação, citando que acarreta a elevação da qualidade do *software*, pois permite realizar maior quantidade de testes em menos tempo. Desta forma, é possível uma maior área de abrangência dos testes, além disso, permite a execução de outras atividades do processo de testes realizados pela equipe.

Existe um vasto número de ferramentas no procedimento de automação que

podem ser aplicadas em diferentes níveis de teste ao decorrer do desenvolvimento de *software* (FEWSTER *et al apud* OLIVEIRA, 2007). Alguns tipos de ferramentas estão brevemente apresentados no Quadro 3.

Quadro 3 – Tipos de ferramentas utilizadas no processo de automação

FERRAMENTA	FUNCIONALIDADE
Ferramentas de Análise Dinâmica	Avaliam o comportamento do sistema enquanto o *software* é executado.
Simuladores	Possibilita a execução de partes de um sistema por meio de simulação de condições.
Ferramentas de Gerenciamento	Apoiam o plano de testes, mantendo assim o relacionamento dos testes que serão executados. Ferramentas para ajudar na rastreabilidade de testes e ferramentas de rastreamento de defeitos também estão incluídas nesta categoria.
Ferramentas de Execução e Comparação	Usadas para automatizar a execução de testes, comparando as saídas do teste com as saídas esperadas. São utilizadas em todos os estágios (testes unitários, testes de integração, testes de sistema e testes de aceitação).

Fonte: Adaptado de OLIVEIRA, 2007

Ferramentas *Record and Playback* são ferramentas de execução e comparação, o *Selenium* é um exemplo dessa categoria de ferramenta.

Especificamente em automação de testes funcionais, a execução dos testes é realizada através da criação e da execução de *scripts* de teste a partir de ferramentas *Record and Playback*. Tais ferramentas proporcionam a programação de *scripts*, ou gravação, e podem ser alterados posteriormente, e são frequentemente utilizados em testes de regressão.

A combinação de ferramentas de tipos diferentes pode ser uma opção para suprir melhor a necessidade de teste e obter maior abrangência.

2.6 Ferramenta de teste *Selenium*

O *Selenium* é um conjunto de ferramentas *open source*, composto por um rico conjunto de funções de teste especificamente orientados para as necessidades de testes de aplicações *web* de todos os tipos. (SANTOS; NETO, 2009).

A ideia primária de construção da ferramenta Selenium foi instituída em 2004

por Jason Huggins. Ao testar um aplicativo na ThoughtWorks10, ele percebeu que havia uma forma melhor de utilizar seu tempo automatizando tais testes. Então desenvolveu o núcleo do Selenium, uma biblioteca JavaScript11 que conduzia interações com a página web, permitindo-lhe reexecutar automaticamente testes em vários navegadores. (SELENIUMHQ, 2013).

Em 2006 um engenheiro do Google chamado Simon Stewart começou a trabalhar em um projeto que chamou de *WebDriver.* O intuito de Simon era criar uma ferramenta de teste que comunicasse diretamente com o navegador usando o método nativo para o navegador e sistema operacional, evitando assim as restrições de um ambiente de *JavaScript* em área restrita. (SELENIUMHQ, 2013).

Em 2008, ocorreu a fusão do *Selenium* com o *WebDriver*. A junção destas duas ferramentas trouxe uma das mais brilhantes ferramentas em automação de teste com um conjunto comum de recursos para todos os usuários. (SELENIUMHQ, 2013).

A fusão dos projetos ocorreu porque uma ferramenta complementa a outra. Desta forma, o *Webdriver* aborda algumas deficiências presentes no *Selenium* e vice-versa. Além disso, esta foi a melhor maneira de oferecer aos usuários das ferramentas a melhor estrutura de teste possível. (SELENIUMHQ, 2013).

O *Selenium* foi escrito utilizando *JavaScript* e DHTML (*Dynamic HyperText Markup Language*). Desta forma, seus testes rodam diretamente a partir do navegador e podem ser executados virtualmente em qualquer navegador que suporte *JavaScript*. (CAETANO, 2007).

O *Selenium* vem com um conjunto de comandos para: controle de operações de testes, navegador e operações de *cookie*, *pop-up*, botão, lista, campo de edição, teclado, *mouse* e operações de formulário. Também fornece operações de acesso para examinar a aplicação *web*. (COHEN, 2009).

As operações realizadas pela ferramenta são altamente flexíveis, permitindo muitas opções para a localização de elementos de interface do usuário e comparando os resultados dos testes previstos com o real comportamento da aplicação. (SELENIUMHQ, 2013).

Basicamente, os testes do *Selenium* são escritos em tabelas HTML (*HyperText Markup Language*). Nestas tabelas, são informadas as operações ou asserções de

[10] Empresa de consultoria global em tecnologia de informação que tem como foco o desenvolvimento ágil de *software*.
[11] Linguagem de criação de *scripts* desenvolvida pela *Netscape* em 1995.

um teste e os seus respectivos argumentos. O *Selenium* é responsável por interpretar os comandos das tabelas HTML e executar as ações, simulando um usuário real. (CAETANO, 2007).

Segundo Cohen 2009, para identificar os elementos dentro de uma página da *web* o *Selenium* utiliza os seguintes comandos descritos no Quadro 4.

Quadro 4 – Comandos *Selenium* para Identificação de Elementos *Web*

COMANDOS	DESCRIÇÃO
identifier=id	Seleciona o elemento especificado com o atributo @id. Se nenhuma correspondência for encontrada, seleciona o primeiro elemento @name cujo atributo é id.
name=nome	Seleciona o primeiro elemento especificado com o atributo @name. O nome pode, opcionalmente, ser seguido por um ou mais elementos-filtros, separados por espaço em branco após o nome. Se o tipo de filtro não for especificado, o *value* é assumido.
dom=expressão *JavaScript*	Encontra um elemento HTML usando *JavaScript* como passagem do DOM (Document Object Model). Localizadores DOM devem começar com "document". Por exemplo: dom=document.forms[‘form1’].myList dom=document.images[1]
xpath= expressão *Xpath*[12]	Localiza um elemento usando uma expressão XPath (XML Path Language). Exemplo: xpath=//img[@alt=’The image alt text’]
link= texto padrão	Seleciona o elemento link (âncora), que contém o texto correspondente ao padrão especificado.
css=cssSelectorSyntax	Seleciona o elemento usando seletores CSS.

Fonte: COHEN, 2009

Na documentação do projeto consta a descrição dos comandos típicos do *Selenium*, que são provavelmente os comandos mais usados para a construção de testes. Tais comandos são apresentados no Quadro 5.

Quadro 5 – Comandos Básicos da Ferramenta *Selenium*

(continua)

COMANDOS BÁSICOS	DESCRIÇÃO
Open	Abre uma página usando uma URL (*Uniform Resource Locator*) que é fornecida como parâmetro.
click/clickAndWait	Executa o clique em um botão, *link* ou imagem e espera o carregamento de uma nova página.
verifyTitle/assertTitle	Verifica um título de uma página.
verifyTextPresent/assertTextPresent	Verifica a presença de um texto em qualquer lugar da página.

[12] Conjunto de regras de sintaxe para definir partes de um documento XML(*eXtensible Markup Language*).

(conclusão)

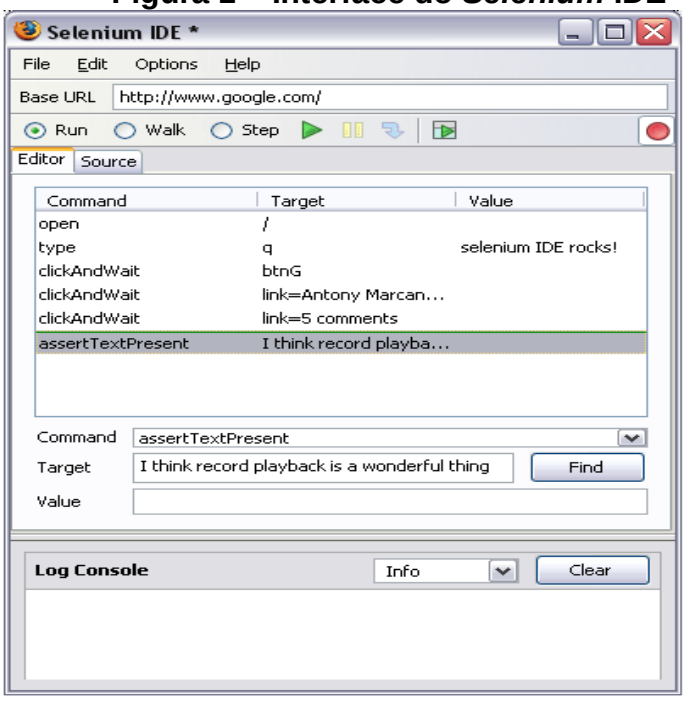
COMANDOS BÁSICOS	DESCRIÇÃO
verifyText/assertText	Verifica se um texto aparece em um determinado local.
verifyTable	Verifica conteúdo de uma tabela.
waitForPageToLoad	Pausa uma execução do teste até que uma nova página seja carregada.
verifyElementPresent	Verifica se um componente gráfico, definido por uma *tag* HTML, está presente na página.
waitForElementPresent	Interrompe a execução até que um componente gráfico definido por uma *tag* HTML esteja presente na página.
Type1	Entra com um valor em um determinado campo da página.
Select	Seleciona um elemento dentre uma lista de opções.

Fonte: SELENIUMHQ, 2011

O *Selenium* está disponível em três versões diferentes: *Selenium* IDE, *Selenium* RC e *Selenium Grid*. Abaixo estão pontuadas algumas características de desses módulos.

O Selenium IDE caracteriza-se por ser um ambiente de desenvolvimento integrado para construção de casos de testes. Sendo uma extensão do *Mozilla Firefox* capaz de fornecer interfaces amigáveis e de fácil utilização para o desenvolvimento e execução d'e conjuntos de testes. A Figura 1 mostra a interface deste módulo.

Figura 1 – Interface do *Selenium* IDE



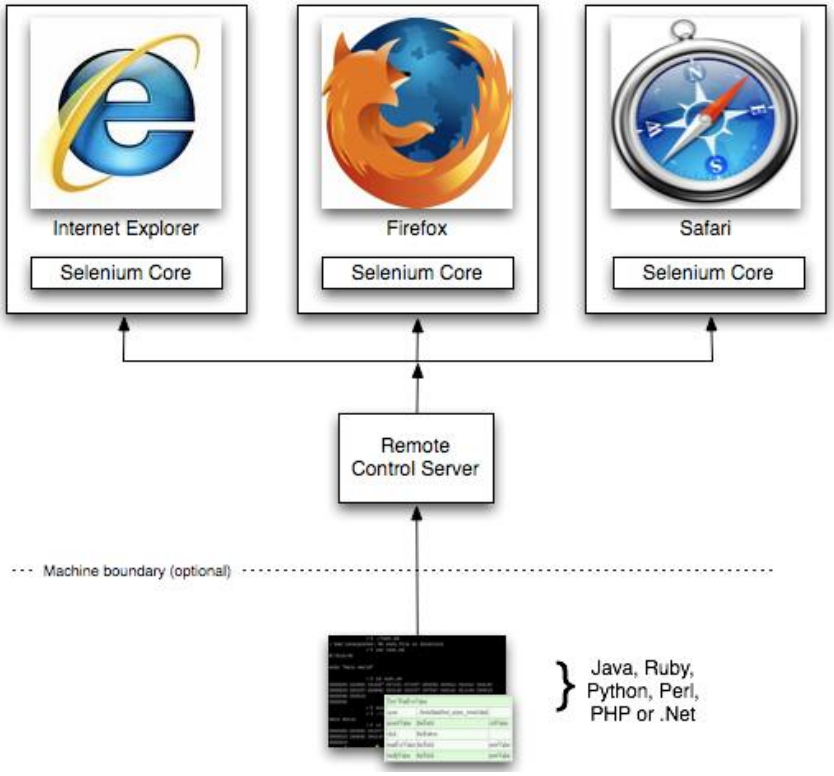
Fonte: SELENIUMHQ, 2013

O *Selenium* IDE se classifica como uma ferramenta do tipo *record-and-playback*, pois é capaz de guardar as ações executadas pelo testador, com isso origina *scripts* que permitem que os testes realizados anteriormente sejam refeitos automaticamente. (SANTOS; NETO, 2009).

O módulo *Selenium* RC (*Remote Control*) é ideal para realizar testes com interação ou que necessitam de condições para ser executados. Permite a utilização de diversas linguagens de programação para a construção de lógicas de teste mais complexas.

O RC possui suporte às linguagens: HTML, Java, C#, Perl, PHP[13], Python e Ruby e apresenta uma API (*Application Programming Interface*) e bibliotecas para cada uma delas. (SANTOS; NETO, 2009). Uma representação simples da arquitetura do RC pode ser visualizada na Figura 2:

Figura 2 – Arquitetura do *Selenium* RC



Fonte: SELENIUMHQ, 2013

Atualmente os mantenedores da ferramenta estão trabalhando na migração do *Selenium* RC para o *Selenium WebDriver.* A maior mudança é a inclusão da API

[13] Acrônimo recursivo para *Hypertext Preprocessor*.

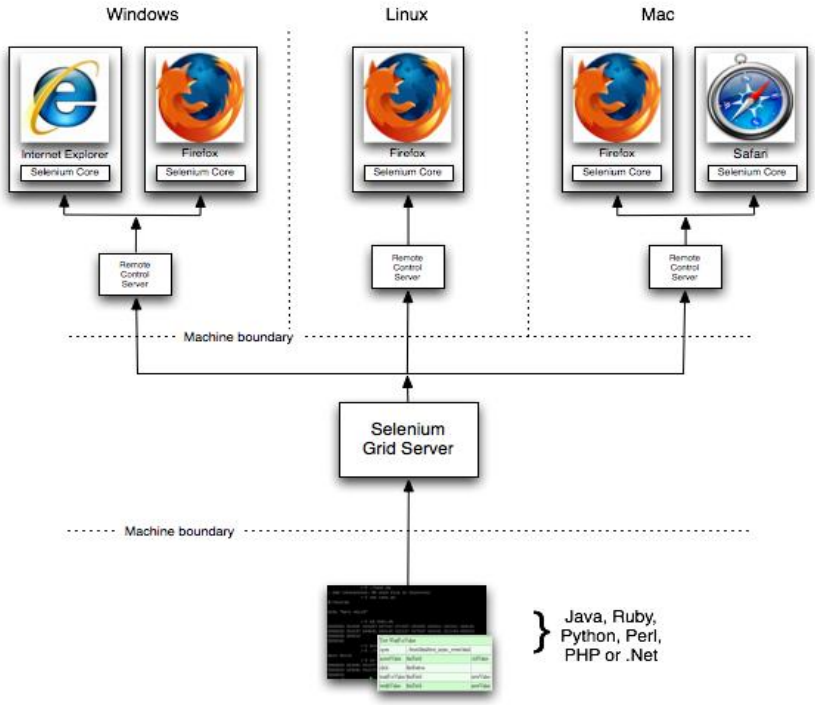
WebDriver, que visa conduzir um navegador de forma natural como um usuário local ou em uma máquina remota usando o *Selenium Server*, o que é uma grande evolução em termos de automação.

O *WebDriver* é projetado em uma interface de programação mais simples e concisa, abordando algumas limitações do *Selenium* RC. *A* API do *WebDriver* é orientada a objeto e pode ser considerada compacta quando comparado com o *Selenium* RC 1.0.

Em relação ao *Selenium Grid*, pode-se mencionar que esta versão permite distribuir os testes em múltiplas máquinas, reduzindo assim o tempo gasto na execução de uma suíte de testes. Ele é ideal para escalonar grandes suítes de testes ou suítes de testes que devem ser executadas em múltiplos ambientes.

O *Selenium Grid* atua executando múltiplas instâncias do *Selenium* RC em paralelo, fazendo com que os testes não precisem se preocupar com a infraestrutura utilizada. (SANTOS; NETO, 2009). Uma demonstração sucinta da estrutura do *Selenium Grid* é mostrada na Figura 3.

FIGURA 3: *Selenium Grid*



Fonte: SELENIUMHQ, 2013

Os módulos apresentados anteriormente podem ser utilizados de forma individualizada ou agrupada dependendo das necessidades exigidas pelos testes a serem executados.

3 ESTADO DA ARTE

Este capítulo apresenta alguns trabalhos relacionados a testes automatizados para sistemas *web* utilizando a ferramenta *Selenium*, através da aplicação de metodologias e perspectivas diferentes.

Machado (2009), no artigo “A implantação de testes automatizados de aplicações *web* num ambiente hostil”, apresenta a implantação de testes automatizados em uma aplicação *web,* mostrando as principais dificuldades para criar um ambiente apropriado para testes. Ele apresenta uma aplicação que já existe há anos e a empresa jamais implantou testes automatizados. São mostradas as dificuldades apresentadas para o início do trabalho, a mudança de paradigma, o alto custo com ferramentas e pessoal. No final ele apresenta o sucesso da automatização de testes passando a identificar falhas no *software* de forma automática, sem custos adicionais, pois se utilizou uma ferramenta gratuita, e promoveu um aumento na qualidade do *software*.

O trabalho apresentado por Gonçalves (2011), “Geração de testes automatizados utilizando o *Selenium*”, mostra o desenvolvimento de uma ferramenta com o intuito de automatizar a etapa de teste, gerando *scripts* para o *Selenium* automaticamente a partir da suíte de testes escritos em linguagem natural (Português). O trabalho de Hiroshi apresenta a demonstração da ferramenta criada em dois testes de formulário, o que é relativamente superficial diante das funcionalidades do *Selenium*.

O trabalho realizado por Silva (2011), “Geração automática de *scripts* de teste utilizando o *Selenium*”, tem o intuito de minimizar o esforço na criação de *scripts* de testes. Para isso, foi desenvolvida uma ferramenta integrada ao *Selenium* que utiliza casos de usos descritos em linguagem natural para a geração automática de *scripts* para testes de aplicações *web*, implementando o mapeamento de frases, linguagem natural, e comandos *Selenium* e assim através dos cenários identificados nos casos de usos são gerados os *scripts* que serão executados no *Selenium* normalmente.

Nazaré (2012) relata a relevância do *software* diante da sociedade moderna,

onde este é utilizado para realizar atividades simples e complexas, caracterizando a importância do seu correto funcionamento. Especifica que a automação da execução dos testes busca reduzir o esforço e facilitar o trabalho de teste, sendo a plataforma *Selenium* um meio para dar os primeiros passos nessa direção. O trabalho apresenta uma ferramenta *web* de criação de *scripts* de teste para o *Selenium* (*SeleniumTG*), com seu desenvolvimento gerenciado com metodologia SCRUM.

Santiago (2013) realizou um estudo de caso sobre testes automatizados com a ferramenta *Selenium*, mostrando as táticas, benefícios e dificuldades dessa técnica. Ele aborda a importância dos testes para a garantia da qualidade de um *software*, e define essa fase como um dos principais fatores para determinar o custo de um sistema. São apresentadas as principais técnicas de testes e ferramentas de automação. Através dos testes realizados, o autor conclui que o uso do *Selenium* contribui para que o desenvolvedor realize uma implementação mais segura, diminuindo o tempo para a realização de testes funcionais.

Os trabalhos apresentados anteriormente expõem estudos diversificados em relação à automação de testes para sistemas *web* através de ferramentas de testes. Pode-se perceber que a maioria dos trabalhos aborda um estudo sobre os diferentes tipos de testes que podem ser realizados com a ferramenta *Selenium*, evidenciando a grande utilidade da mesma. Além disso, complementar suas funcionalidades integrando-o a outras ferramentas.

O diferencial do presente trabalho em relação aos previamente citados é que os testes serão realizados em um sistema *web* coorporativo já em atividade no mercado, onde os testes a serem executados são predefinidos. O intuito é que após os testes seja agregado valor ao *software* e implementado a política de testes na empresa. Outro ponto que diverge é que os conhecimentos adquiridos durante o desenvolvimento do trabalho serão apresentados na forma de um conjunto de boas práticas, demonstrando que a maneira de programar influencia na usabilidade da ferramenta *Selenium* na realização dos testes.

4 METODOLOGIA

Este trabalho consiste em um estudo de caso, que segundo Gil (2008) incide no estudo profundo e exaustivo de um ou poucos objetos, de maneira que permita seu amplo e detalhado conhecimento. O estudo deste trabalho busca trazer algumas explicações sobre testes funcionais utilizando a ferramenta *Selenium* particularmente aplicados ao módulo de Pedidos *Online* do *FalconFV*, mostrando as características desta técnica de teste de maneira aprofundada.

A pesquisa é de abordagem qualitativa, que de acordo com Reneker *apud* Dias (2000) trata-se de uma pesquisa indutiva em que o pesquisador desenvolve conceitos, ideias e entendimentos a partir de padrões encontrados nos dados, ao invés de coletar dados para comprovar teorias, hipóteses e modelos pré-concebidos.

Para chegar ao objetivo desta pesquisa, utilizou-se os seguintes processos: a) Estudo Prévio, b) Fundamentação Teórica, c) Análise de Trabalhos Relacionados, d) Detalhamento das Ferramentas Utilizadas, e) Levantamento de Testes, f) Automação e Aplicação dos Testes, g) Análise dos Resultados.

O Estudo Prévio realizou-se por meio de pesquisas acerca de Engenharia de *Software*, teste de *software* e automação de testes para aplicações *web*. Após o estudo, definiu-se que a automação de testes *web* seria a área de pesquisa, a partir de então se realizou a seleção e a coleta dos dados referentes a esta forma de confecção dos testes bem como das ferramentas que possibilitam automação dos mesmos. Depois das pesquisas, optou-se por estudar e utilizar a ferramenta *Selenium* durante o desenvolvimento do trabalho.

A Fundamentação Teórica contempla o levantamento bibliográfico sobre os conceitos, técnicas, métodos e ferramentas necessárias ao desenvolvimento do trabalho. Também contém a explicação sobre os estágios de testes indispensáveis para a garantia da qualidade de um *software*, citando as técnicas mais utilizadas.

Análise de Trabalhos Relacionados refere-se ao levantamento e análise de trabalhos que propõem a utilização da ferramenta *Selenium* para a aplicação de testes em sistemas *online*. Cada trabalho avaliado é sintetizado para apresentação das suas principais ideias.

O processo de Detalhamento das Ferramentas Utilizadas compõe-se por estudos pertinentes em relação ao módulo IDE da ferramenta *Selenium*, visando conhecer melhor seu funcionamento e sua utilização. O sistema *FalconFV*, é

detalhado dando ênfase ao módulo de Pedidos *Online* com o propósito de entender suas funcionalidades e seu desenvolvimento, para então partir para a fase de testes no mesmo. As ferramentas *Selenium* RC e Eclipse também são descritas.

O Levantamento de Testes refere-se à definição dos testes a serem automatizados pela ferramenta *Selenium* IDE e aplicados no módulo de Pedidos *Online* do sistema *FalconFV*, para isso contou com a participação do engenheiro de *software* da empresa Eagle Tecnologia da Informação LTDA. Este processo ocorreu por meio de entrevistas.

Automação e Aplicação dos Testes consiste em mostrar a automação, através da ferramenta *Selenium,* dos testes definidos no levantamento de testes e depois de aplicá-los no módulo de Pedidos *Online* do sistema *FalconFV*, apresentar os resultados obtidos na execução dos testes automatizados. Esta atividade realizou-se para testar as funcionalidades do sistema, com o intuito de constatar se elas realizam aquilo para o qual foram desenvolvidas.

A Análise dos Resultados contém uma análise crítica sobre o uso da ferramenta *Selenium* IDE juntamente com uma indicação de boas práticas, visando alcançar sua melhor usabilidade para a automação e reutilização de testes predefinidos. Traz também os possíveis benefícios alcançados por empresas da área de desenvolvimento de *software* ao automatizar a fase de teste do produto.

4.1 Ferramentas Utilizadas

Nesta seção estão descritas as ferramentas Eclipse e *Selenium* RC, que foram utilizadas para dar auxílio ao desenvolvimento do trabalho, mas que não compõem o escopo principal deste estudo. As ferramentas *Selenium* IDE e *FalconFV* são detalhadas no próximo capítulo devido sua importância para a pesquisa realizada.

4.1.1 Eclipse

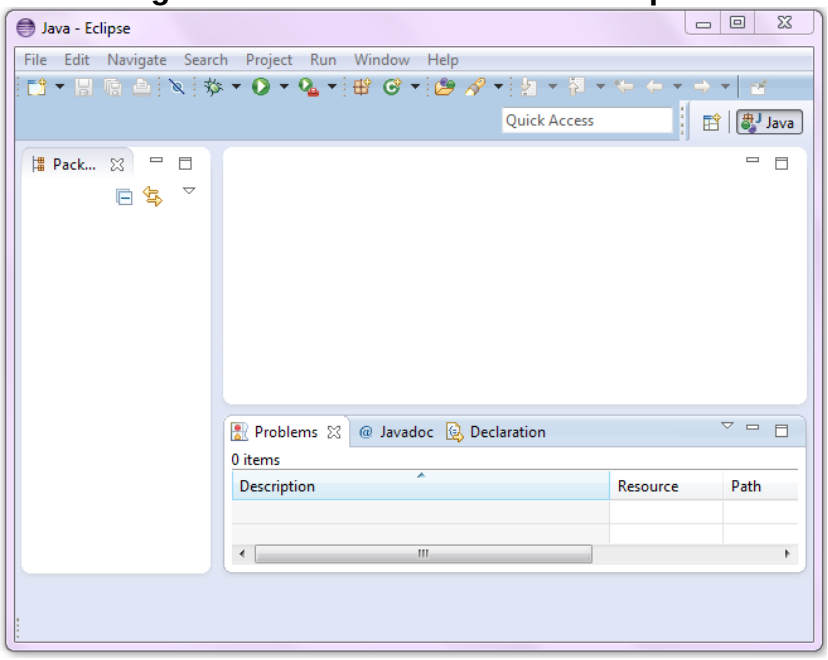
O Eclipse é um IDE *open source* voltado para o desenvolvimento de *software* em linguagem Java, porém a partir da instalação de *plugins*, dá suporte a várias outras linguagens como C, C++, PHP, *ColdFusion*, *Python*, *Scala* e plataforma *Android*. O projeto Eclipse foi iniciado na IBM (*International Business Machines*) que desenvolveu

a primeira versão do produto.

O Eclipse possui como principais características o uso da SWT (*Standard Widget Toolkit*)[14], a forte orientação ao desenvolvimento baseado em *plugins* que fornecem um amplo suporte ao desenvolvedor, procurando atender suas diferentes necessidades. Existem também centenas de *plugins* gratuitos para suporte a servidores de aplicação, visualizadores de banco de dados, criação de diagramas UML (Unified Modeling Language), dentre outros.

O Eclipse é o IDE Java líder de mercado atualmente e seu objetivo é possibilitar o desenvolvimento mais rápido através do *drag-and-drop* do *mouse,* evitando erros de programação. Sua versão mais recente é a 4.3.1 que pode ser baixada no site http://www.eclipse.org. A Figura 4 mostra a área de trabalho do Eclipse.

Figura 4 – Área de trabalho do Eclipse



Fonte: Elaborado pelas autoras

4.1.2 Selenium RC

Como citado previamente, o *Selenium* RC é um dos componentes da ferramenta *Selenium*, com o qual é possível a implementação de testes utilizando uma linguagem de programação, isso possibilita ao testador executar testes complexos.

[14] Biblioteca projetada para fornecer acesso eficiente e portátil para as instalações da interface do usuário dos sistemas operacionais nos quais é aplicada.

Esta versão do *Selenium* é ideal para testar interfaces *web* baseadas em AJAX[15]. E proporciona que os testes feitos com o *Selenium* IDE possam ser escritos de forma mais robusta através da linguagem de programação, sendo mais apresentáveis do que no formato Selenese[16] que os exibe em uma tabela HTML.

O *Selenium* RC apresenta como componentes o *Selenium Server* e as bibliotecas. O *Selenium Server* tem por funções iniciar e fechar os navegadores para a execução dos testes e interpretar e executar os comandos Seleneses. Além disso, atua como um *proxy* HTTP (*Hypertext Transfer Protocol*) que intercepta e verifica mensagens HTTP trocadas entre o navegador e o aplicativo testado. E as bibliotecas são formadas por um conjunto de instruções que fornece a interface entre a linguagem de programação e o *Selenium Server*.

O RC funciona da seguinte maneira: as bibliotecas se comunicam com o servidor passando cada comando *Selenium* para execução, em seguida, o servidor passa o comando Selenium para o navegador usando comandos do *Selenium Core*, um conjunto de funções de *JavaScript* que interpreta e executa os comandos Selenese, o navegador então, usando o seu interpretador de *JavaScript*, executa o comando *Selenium* e a ação Selenese ou verificação especificada no *script* de teste.

O download do *Selenium* RC pode ser realizado no site oficial da ferramenta http://docs.seleniumhq.org/download/.

[15] *Asynchronous JavaScript and XML*
[16] DSL (*Domain Specific Language*), uma linguagem específica do Selenium IDE utilizada para escrever os testes.

5 PROCEDIMENTOS E RESULTADOS

Este capítulo descreve os procedimentos utilizados e os resultados obtidos durante o desenvolvimento do trabalho. A seção 5.1 detalha as ferramentas *Selenium* IDE e *FalconFV*, a 5.2 contém os casos de testes que especificam os testes realizados no módulo de Pedidos *Online* do *FalconFV*. Na seção 5.3 encontra-se a automação e aplicação dos testes realizados juntamente com os resultados obtidos.

5.1 Detalhamento das Ferramentas Utilizadas

Esta seção apresenta a descrição das ferramentas utilizadas para o desenvolvimento do trabalho. A seção 5.1.1 descreve a ferramenta *Selenium* IDE e a seção 5.1.2 descreve o sistema *FalconFV*.

5.1.1 Selenium IDE

O *Selenium* IDE é uma extensão para o navegador *Mozilla Firefox* e apresenta um *design* simples. Por apresentar uma interface amigável e intuitiva, como pode ser visualizado na Figura 5, é de fácil utilização.

É uma ferramenta extremamente útil para a automação de testes, pois pode reproduzir tudo o que um usuário pode fazer interagindo com uma página *web*.

A ferramenta gera códigos em *JavaScript* que manipulam qualquer tipo de elemento, desta forma possui a capacidade desde preencher valores até verificar itens com valores específicos. (MARINS, 2009).

Para a instalação do *Selenium* IDE é necessário que o navegador *Mozilla Firefox* esteja instalado sendo que a versão 0.2 da ferramenta é compatível com o *Firefox* a partir da versão 1.5 e as posteriores funcionam a partir da versão 3.5 do navegador.

Para utilizar o *Selenium* é necessário fazer sua instalação e o primeiro passo a fazer é realizar seu *download* na página http://seleniumhq.org/download/. No site é apresentado um link para o download da versão mais recente desta ferramenta de automação de testes, a 2.4. Após sua instalação é necessário reiniciar o navegador e o *Selenium* já está pronto para ser utilizado.

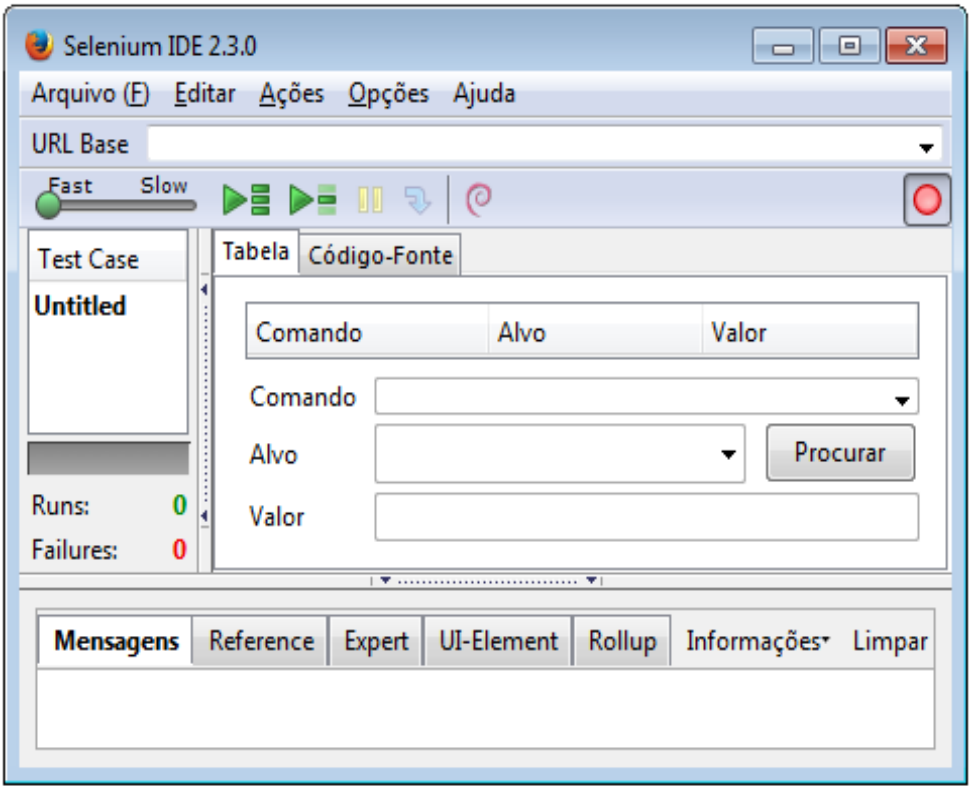
Por *default*, o modo de gravação dos procedimentos de interação que são

realizados pelos usuários do *website* no *Selenium* já está ativado. Na interface mostrada anteriormente pode-se visualizar os itens que compõe o *Selenium* IDE, as funcionalidades destes itens estão explicitadas a seguir.

O campo URL[17] Base é o *path* do sistema, onde fica o endereço do site que se pretende realizar os testes. Abaixo deste item tem a opção *Speed Control* onde se define a velocidade desejada para a execução dos testes sendo delimitada pelas velocidades *Fast* e *Slow*. O *Fast* é a opção padrão e representa a velocidade mais alta e o *Slow* a mais baixa. Esta opção é muito útil para a observação dos testes no momento em que estão executando.

Posteriormente, é encontrado o primeiro botão *Run All* que executa uma suíte de testes, onde são executados vários testes na ordem em que estão abertos. O segundo botão de *Run* serve para rodar apenas o arquivo de teste que está aberto atualmente, ele roda todos os comandos do teste a partir do ponto inicial.

Figura 5 – Interface do Selenium IDE



Fonte: OPENQA, 2013

O botão *Pause* serve para interromper a execução do teste. Existe também o

[17] *Uniform Resource Locator*

botão de iteração que serve para debugar o teste passo a passo. Ao lado está o botão que serve para agrupar comandos como regras simples. Em seguida há o botão de *Record* que serve para iniciar e parar a gravação das interações realizadas pelo usuário nas página.

No campo *Test Case* ficam os títulos dos casos de teste que estão abertos, desta forma quando há dois ou mais casos de testes abertos, a ferramenta possibilita que sejam salvos como uma suíte de testes.

Além do campo existe também a aba Tabela que permite que os testes sejam feitos através do uso do assistente. Na aba Código-Fonte é possível editar os testes pelo código-fonte, onde este pode ser na linguagem padrão HTML ou nas outras linguagens disponíveis na ferramenta (Java, C#, Perl, PHP, Python e Ruby).

Os campos Comando, Alvo e Valor são campos de entrada que permitem inserção e a edição de comandos para o teste. Em Comando é onde fica as ações a serem executadas durante o teste como preencher um campo, abrir a URL. O alvo é um valor que serve de parâmetro para um comando e varia de acordo com o comando especificado, e este campo tem como complemento o botão procurar que indica na página testada o que está sendo testado. O valor serve para setar algum valor em um campo sendo utilizado quando a ação apresenta mais de um parâmetro.

Posteriormente são apresentadas as abas de Mensagem que apresenta execução do passo a passo do teste mostrado inclusive os logs de erros. A guia *Reference* traz uma explicação sobre o comando na qual sua sintaxe é descrita.

A guia *UI-Element* serve para tornar os testes mais legíveis e fáceis de atualizar e entender, pois através deste o testador pode usar mapeamentos utilizando JSON[18], também proporciona a fácil visualização da descrição dos elementos que se encontra na guia *Reference*.

A guia *Rollup* tem como objetivo realizar o agrupamento de comandos. Onde a partir da definição das regras de agrupamento (*Rollup rules*), o testador pode reunir comandos em grupos possibilitando que eles sejam invocados com o comando *rollup* (SANTOS; NETO, 2009).

Na ferramenta também estão disponíveis os menus Arquivo que permite criar, exportar, editar, adicionar e salvar testes. O Editar contém as opções que permitem editar os casos de testes (refazer, desfazer, copiar, recortar, colar, excluir). O menu

[18] Acrônimo de *JavaScript Object Notation*

Opções permite mudar as configurações da ferramenta, como por exemplo, tempo limite que a ferramenta deve esperar por uma resposta (*timeout*) e linguagem será usada para salvar os casos de testes. E o menu Ações é onde estão as funcionalidades dos botões *Record*, *Run All*, *Run*, *Speed Control* e *Pause* em forma de texto.

5.1.2 FalconFV

O sistema *FalconFV* é o produto mais recente lançado no mercado pela empresa Eagle Tecnologia LTDA. Uma empresa que está há cinco anos no mercado atuando com tecnologia da informação e desenvolvimento de sistemas. Dentre os serviços prestados, a empresa desenvolve sistemas para as plataformas *web*, *desktop* e *mobile* (*Android*), utilizando as tecnologias J2SE (*Java Platform*, *Standard Edition*), J2EE (*Java Platform*, *Enterprise Edition*), PHP e C/C++, em conjunto com os bancos de dados Postgres, MySql, *Firebird* e *SQLite*.

O *FalconFV* é um sistema completo para apoio a vendas. Foi criado com o objetivo de agilizar o processo de atendimento e geração de pedidos em tempo real da venda para reduzir o trabalho deste processo. O *FalconFV* é composto por um *Web Service* - um *middleware*[19] -, a App[20] *Mobile*, o monitor *FalconFV*, o módulo de Pedidos *Online*, e é necessário uso de um ERP (*Enterprise Resource Planning*), atualmente a empresa utiliza o Eagle ERP. Nas seções abaixo cada módulo do sistema *FalconFV*, citados anteriormente, estão detalhados.

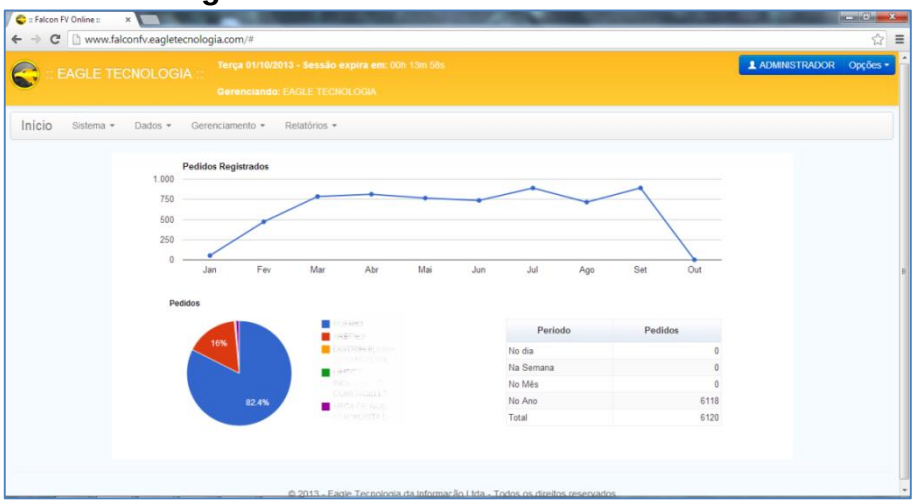
5.2.1.1 Web Service

O *Web Service* como especificado anteriormente é um *middleware*, solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. A Figura 6 mostra a tela inicial deste módulo do *FalconFV*. O termo *Web Service* é utilizado no *FalconFV* não somente como um *middleware*, mas como um gerenciador de dados para os gestores das empresas e no contexto deste trabalho não se aplica a definição de locais para armazenamento de dados ou site nas nuvens. Os dados do

[19] Sistemas que ajudam a gerenciar a complexidade e as diferenças intrínsecas ao desenvolvimento de aplicações e sistemas distribuídos
[20] Aplicativo móvel

Eagle ERP são mandados para o *Web Service*, onde podem ser consultados, extraídos relatórios e sincronizar com o *FalconFV* App *Mobile*.

Figura 6 – Tela inicial do *Web Service*

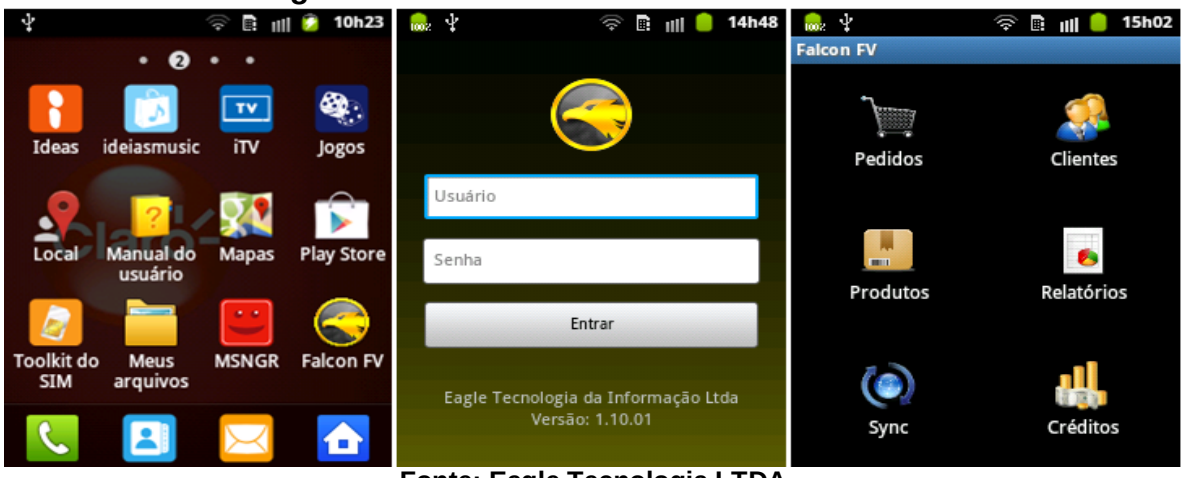


Fonte: Eagle Tecnologia LTDA

5.2.1.2 App Mobile

O App *Mobile* é 100% nativo na plataforma *Android* 2.3 ou superior. O funcionamento está vinculado às configurações do ERP, onde os dados enviados ao ERP são sincronizados. Sincronizar neste caso significa baixar os dados enviados ao *Web Services* para o *FalconFV Mobile*. Na Figura 7 estão as telas iniciais da App *Mobile* do *FalconFV*

Figura 7 - Telas iniciais do *FalconFV Mobile*



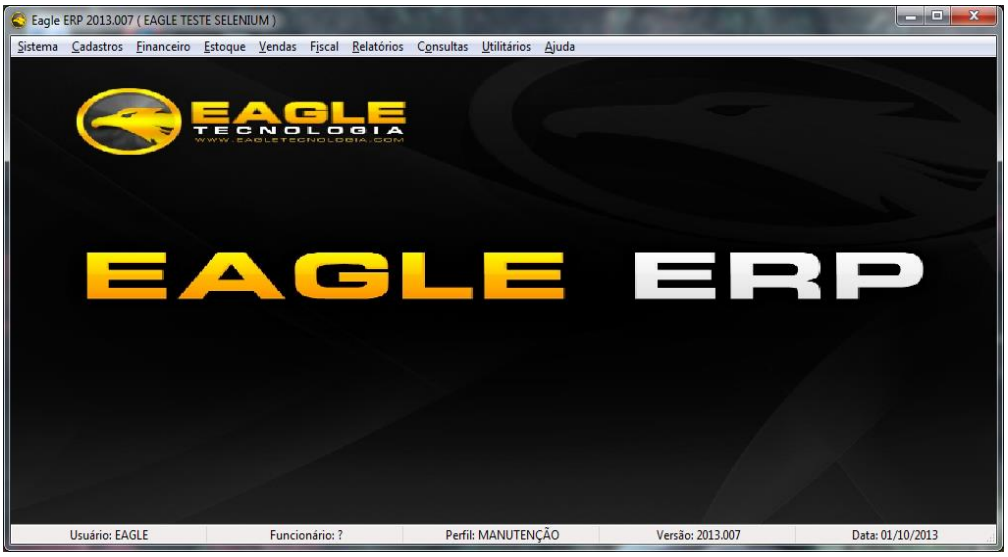
Fonte: Eagle Tecnologia LTDA

5.2.1.3 Eagle ERP

O Eagle ERP é o Sistema Integrado de Gestão Empresarial desenvolvido pela Eagle Tecnologia LTDA projetado para atender as necessidades das empresas sem interferir no seu modelo de negócio. Os requisitos deste sistema serviram de base para o desenvolvimento do *FalconFV*.

Os cadastros e políticas de segurança, referentes ao *FalconFV*, são configurados no Eagle ERP e enviados ao *Web Services*. A tela principal deste sistema é representada na Figura 8.

Figura 8 – Tela principal do Eagle ERP



Fonte: Eagle Tecnologia LTDA

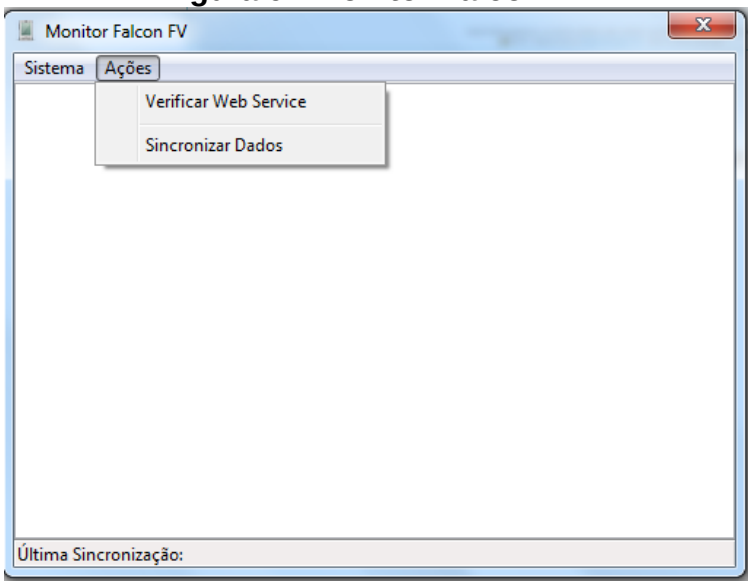
5.2.1.4 Monitor FalconFV

O monitor *FalconFV*, cuja a tela inicial é apresentada na Figura 9, é um integrador do *Web Service* e o ERP da empresa. Através do monitor é possível verificar o *Web Service* e a sincronização de dados.

A função do monitor é realizar tanto o envio dos dados alterado no ERP para o *Web Service* como fazer o processo reverso, onde os dados do mesmo são baixados pelo ERP.

No caso deste trabalho é utilizado o Eagle ERP, porém, é possível integrar o *FalconFV* a ERPs das empresas que utilizam o sistema.

Figura 9 - Monitor Falcon FV



Fonte: Eagle Tecnologia LTDA

5.2.1.5 Módulo de Pedidos Online do FalconFV

O *FalconFV online* foi desenvolvido usando estratégias para melhorar o desempenho, oferecer maior dinamismo, interação, estabilidade e produtividade. A concepção do *software* foi baseada no desenvolvendo em três camadas MVC[21] (*Model*, *View*, *Controller*) utilizando a linguagem PHP e requisições AJAX. Utilizou-se as tecnologias *Zend*, *framework* para aplicações *web* de código aberto e orientado a objetos, o *Twitter Bootstrap*, um *framework* de redesenho de interface, a biblioteca *JavaScript Jquery/JqueryUI* e o banco de dados *MySql*.

Flanagan (2002) define que AJAX trata-se de uma técnica de carregamento de conteúdos em uma página *web* com uso de *JavaScript* e XML, HTML, PHP, ASP, JSON ou qualquer linguagem de marcação ou de programação capaz de ser recuperada de um servidor.

O AJAX foi utilizado visando o melhor desempenho do sistema, pois esta tecnologia utiliza o *JavaScript* e o XML para carregar e renderizar uma página com o intuito de tornar o navegador mais interativo com o usuário. Para isso, utiliza recursos de *scripts* rodando pelo lado cliente, buscando e carregando dados em *background*

[21] Arquitetura de projeto que tem por objetivo separar o código em três camadas. A camada *Model* (Modelo) contém as regras de negócio do sistema, a camada *View* (Visão) é a interface entre usuário e o sistema, e os *Controller*s (Controladores), são utilizados para controlar o fluxo da aplicação. Desta forma cada área só trabalha com itens que a competem.

sem a necessidade de recarregar a página.

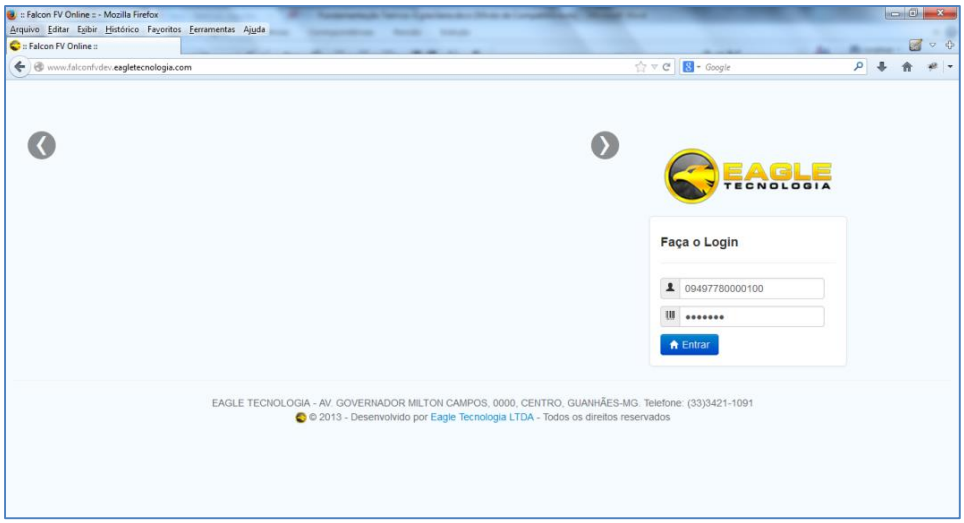
Sempre que são utilizadas as requisições assíncronas no caso do *FalconFV* feitas em AJAX, não será necessário o carregamento total da página para que os resultados sejam retornados, os dados trafegados pela rede são reduzidos e o usuário não precisa aguardar a página ser recarregada a cada requisição com o servidor.

O método utilizado para desenvolver o sistema foi a metodologia ágil, por permitir um desenvolvimento rápido e interativo com a participação e testes do cliente, já que este módulo foi desenvolvido atendendo as necessidades de um cliente que já utilizava o modulo *FalconFV mobile*.

O módulo de Pedidos *Online* do sistema *FalconFV* consiste em um sistema *web*, voltado para empresas distribuidoras, que possibilita venda de produtos *online*. Os clientes das distribuidoras realizam os pedidos de acordo com a regra de negócio estabelecida, onde cada empresa possui o período para realização e prazos para entrega dos mesmos.

O acesso ao sistema é realizado pela inserção do *login* e senha dos clientes. A tela de *login* do sistema *FalconFV* encontra-se na Figura 10.

Figura 10 - Tela de *login*

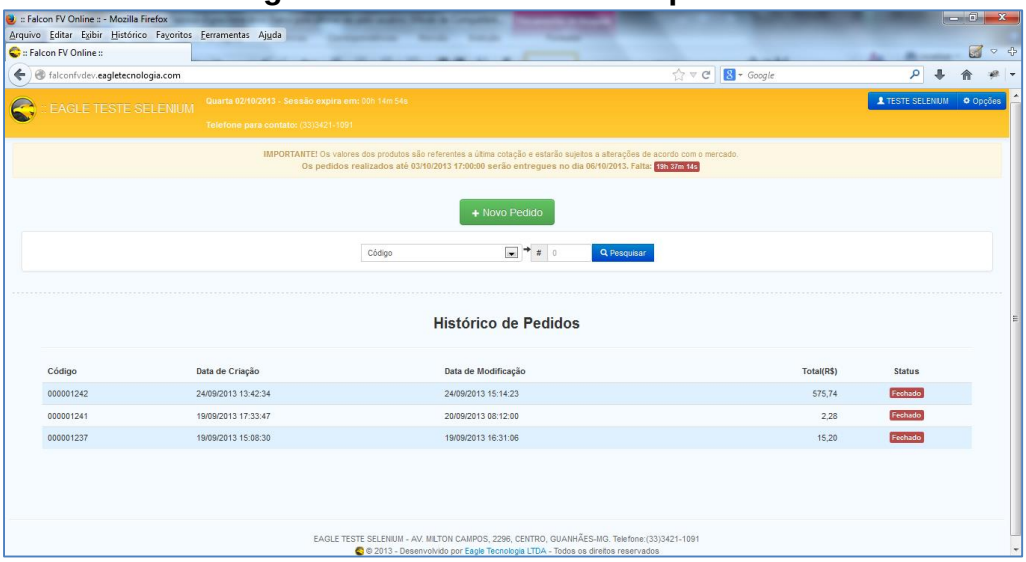


Fonte: Eagle Tecnologia LTDA

Após logar no sistema o usuário é direcionado para a tela de acordo com a situação dos pedidos existentes. Se não existir pedido com status aberto o sistema direciona o usuário para a tela de histórico de pedidos, Figura 11, onde exibe os pedidos realizados anteriormente e o botão novo pedido, que permite aos usuários a

realização de um novo pedido.

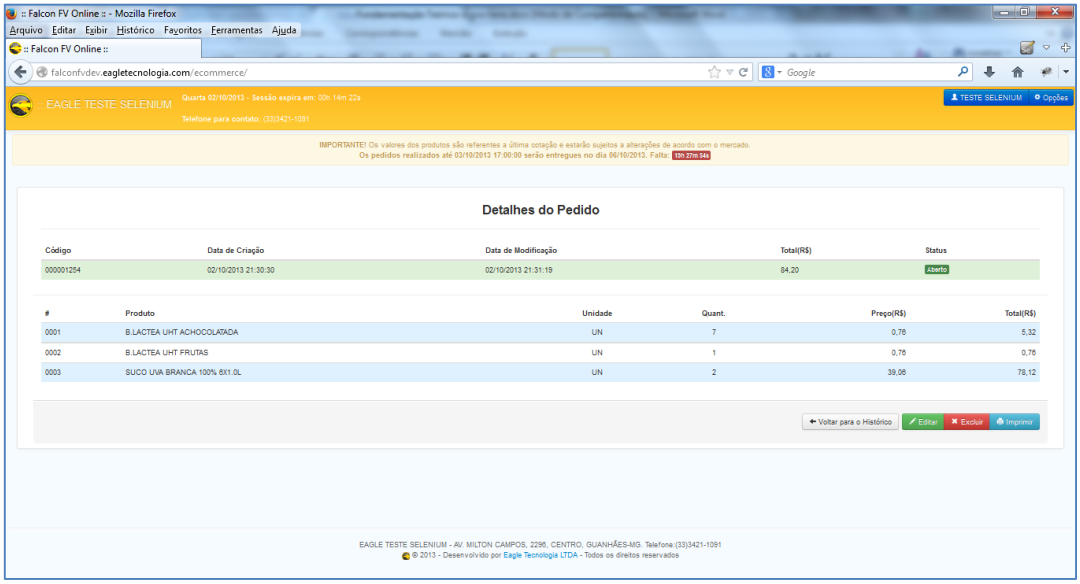
Figura 11 - Tela histórico de pedidos



Fonte: Eagle Tecnologia LTDA

Caso existam pedidos abertos o sistema direciona o usuário para a tela de detalhes de pedido, Figura 12,onde é possível editar, excluir ou imprimir o pedido além da opção voltar ao histórico que leva o usuário a tela do histórico de pedidos.

Figura 12 - Tela Detalhes do Pedido



Fonte: Eagle Tecnologia LTDA

Ao clicar na opção editar o usuário será direcionado para a tela de realização do pedido, onde é possível excluir, adicionar ou alterar a quantidade de produtos inclusa no pedido. Para adicionar produtos ao pedido, é necessário inserir a

quantidade e clicar no botão correspondente para adicionar o produto.

A opção alterar é válida somente para modificar a quantidade de produtos. Para excluir um produto o usuário clica no ícone que aparece a direita do produto. A opção de salvar é acionada quando se deseja salvar a edição do pedido, caso contrário a opção cancelar é escolhida.

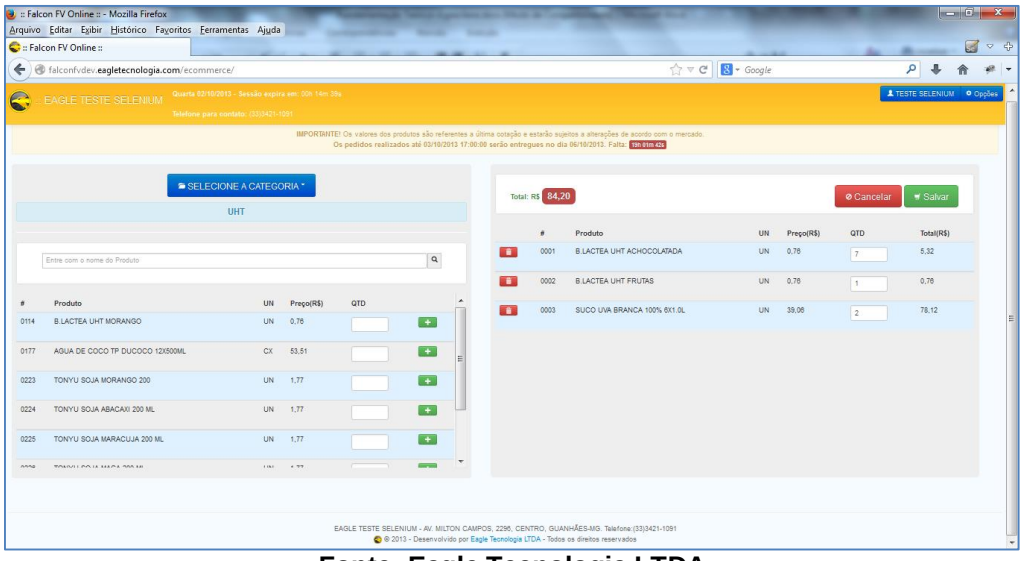
Quando o botão de novo pedido é acionado, a página para que sejam selecionados os produtos desejados para compor o pedido é exibida, a página é apresentada conforme a Figura 13. Para facilitar a busca pelos produtos existem filtros por categoria ou por nome do produto.

Para adicionar um produto o usuário deve inserir a quantidade desejada e clicar no botão de adicionar. Os produtos adicionados podem ser visualizados em uma tabela em paralelo a lista de produtos. Após salvar os pedidos, acontece o redirecionamento para a página que mostra os detalhes do pedido e o cliente pode editar, excluir, imprimir o pedido ou retornar a página de histórico de pedidos.

Os pedidos realizados possuem como dados: código, data de criação, data de modificação, valor e status. O status pode assumir os valores aberto ou fechado.

O status em aberto significa que o período para realizar pedidos ainda não terminou e que os clientes ainda podem acrescentar mais produtos ao pedido. Após o período de realização de pedido expirar, seu status assume o valor de fechado, ficando assim os clientes impossibilitados de acrescentarem produtos aquele pedido.

Figura 13 - Tela de realização de pedido



Fonte: Eagle Tecnologia LTDA

5.3 Casos de Teste

Caso de teste segundo Souza *apud* Heumann (2001) pode ser definido como procedimento para identificar e notificar formalmente as ações e condições específicas detalhadas, que serão validadas para permitir a avaliação de determinados aspectos dos itens a serem testados.

Os casos de testes são usados principalmente para listar um número adequado de testes específicos garantindo a abrangência da avaliação, nortear execução dos testes manualmente ou automatizados. Eles devem cobrir o máximo de situações possíveis e fornecer uma descrição dos pontos-chave de observação e controle de qualquer pós ou precondição. Para elaboração dos casos de testes apresentados neste trabalho, será utilizado um modelo baseado no padrão elaborado por FINAMORE *et. al.* (2009), o qual está representado no Quadro 6.

Quadro 6 – Formato padrão de caso de teste.

Identificador do Caso de Teste	Título do Caso de Teste
Importância	Importância do caso de teste: Alta, Média ou Baixa.
Propósito	Descrição do que se espera testar com cada caso de teste. Parte do Sistema que será testada.
Precondições	Condições que devem ser satisfeitas antes de iniciar o caso de teste.
Passos	Passos que são executados no teste.
Resultados Esperados	Descrição dos resultados que se espera para o sucesso do caso de teste.
Situações de Erros	Descrição do comportamento do sistema para o teste com dados incorretos ou ausentes.

Fonte: FINAMORE *et. al*, 2009

5.3.1 Requisitos de Testes do Módulo de Pedidos Online

Esta seção apresenta o levantamento de testes realizado na empresa Eagle Tecnologia juntamente com o engenheiro de *software* do *FalconFV*. A entrevista focalizada foi o processo utilizado para listar os aspectos do sistema que foram testados. Este modelo de entrevista foi aplicado por ser semiestruturado e permitir que o entrevistado fale livremente sobre um tema específico.

Os dados coletados na entrevista realizada definiram os aspectos testados nos quais estão: realização de *login*, tempo para a seção expirar, botão novo pedido, botão

repetir pedido, botão salvar, botão editar, botão excluir, opção retornar ao histórico, exclusão de produtos do pedido. Tais aspectos serviram de base para a criação dos casos de testes descritos na seção subsequente.

5.3.2 Casos de Testes do Módulo de Pedidos Online

A seguir, nos Quadros 7 a 15, estão descritos os casos de teste do módulo de Pedidos *Online* do sistema *FalconFV* , de acordo com o modelo apresentado no Quadro 6. Os casos de testes refere-se à descrição do correto funcionamento de cada teste realizado, ou seja, consiste em um passo-a-passo das ações realizadas.

Quadro 7 – Caso de teste 01

CT01	Realizar *login* no sistema
Importância	Alta.
Propósito	Testar se os usuários conseguem acessar o sistema.
Precondições	Usuário deve ser cadastrado no sistema.
Passos	1. O usuário insere seu *login*. 2. O usuário insere sua senha. 3. O sistema valida o usuário e permite o acesso ao sistema.
Resultados Esperados	O usuário acessa o sistema.
Situações de Erros	O usuário não acessa o sistema.

Fonte: Elaborado pelas autoras

Quadro 8 – Caso de teste 02

CT02	Realização de um novo pedido
Importância	Alta
Propósito	Testar a inclusão de um novo pedido no sistema.
Precondições	O usuário deve estar logado no sistema. Não deve existir pedido com status aberto.
Passos	1. O usuário clica no botão novo pedido. 2. O sistema redireciona o usuário para a página de pedidos. 3. O usuário realiza o pedido adicionando os itens especificando a quantidade desejada. 4. Os itens adicionados são ocultados da lista de produtos. 5. O sistema mostra os produtos adicionados ao pedido. 6. O sistema mostra o valor total de cada item pedido. 7. O sistema mostra o valor total do pedido. 8. O usuário realiza os passos de **CT03**. 9. O usuário é redirecionado para a página de detalhes do pedido.
Resultados Esperados	O pedido aparece no histórico. O botão novo pedido é ocultado.
Situações de Erros	O botão novo pedido não é ocultado.

Fonte: Elaborado pelas autoras

Quadro 9 – Caso de teste 03

CT03	Salvar pedido
Importância	Alta.
Propósito	Testar se após clicar no botão salvar o pedido é salvo e o usuário redirecionado para a página de detalhes de pedidos.
Precondições	O usuário deve estar logado no sistema. O usuário deve estar realizando um pedido.
Passos	1. Usuário clica no botão “salvar”; 2. O sistema retorna mensagem de confirmação para salvar; 3. Usuário clica no botão “sim”; 4. O sistema retorna mensagem de pedido salvo com sucesso, e redireciona para tela de “Detalhes do pedido”, com todas as informações que foram inseridas.
Resultados Esperados	Os dados inseridos no pedido permanecem após salvar.
Situações de Erros	As alterações no pedido não são salvas.

Fonte: Elaborado pelas autoras

Quadro 10 – Caso de teste 04

CT04	Editar pedido
Importância	Alta.
Propósito	Testar se após salvar o pedido o botão editar fica habilitado e permite ao usuário fazer alterações no pedido.
Precondições	O usuário deve estar logado no sistema. Ter um pedido com status aberto.
Passos	1. O usuário clica no botão editar. 3. O sistema abre o pedido. 4. O usuário edita o pedido adicionando ou removendo produtos. 5. O usuário salva o pedido. 6. O sistema retorna mensagem de pedido salvo com sucesso, e redireciona para tela de “Detalhes do pedido”, com todas as informações que foram inseridas.
Resultados Esperados	As alterações estão presentes no pedido editado.
Situações de Erros	Botão editar não está ativo para pedidos em abertos. Não ser aberta a janela para alterar o pedido após botão editar pedido ser acionado. Não salvar as alterações realizadas.

Fonte: Elaborado pelas autoras

Quadro 11 – Caso de teste 05

(continua)

CT05	Excluir pedido
Importância	Alta.
Propósito	Verificar se após clicar no botão excluir o pedido é deletado do sistema.
Precondições	O usuário estar logado no sistema. Ter um pedido aberto. Estar na tela de detalhes do pedido.

(conclusão)

CT05	Excluir pedido
Passos	1. O usuário clica no botão excluir. 2. O sistema pede confirmação da exclusão. 3. O usuário confirma a exclusão. 3. O pedido é retirado do sistema. 4. O sistema retorna para a tela de histórico de pedidos.
Resultados Esperados	Retirada do pedido do sistema.
Situações de Erros	Botão excluir não está ativo para pedidos em abertos. Pedido estar ativo no sistema apos ser excluido.

Fonte: Elaborado pelas autoras

Quadro 12 – Caso de teste 06

CT06	Botão histórico de pedidos
Importância	Média.
Propósito	Verificar se ao acionar o botão voltar ao histórico o sistema retorna para a tela de histórico de pedidos.
Precondições	Estar logado no sistema. Estar na tela de detalhes de pedido.
Passos	1. O usuário clicar no botão voltar para o histórico. 2. O sistema retorna a tela do histórico do pedido. 3. O sistema mostra o relatório de pedidos realizado, e o primeiro pedido deve esta com status aberto.
Resultados Esperados	O sistema retorna a tela de histórico de pedidos.
Situações de Erros	O sistema não exibe o histórico de pedidos.

Fonte: Elaborado pelas autoras

Quadro 13 – Caso de teste 07

CT07	Repetir pedido
Importância	Alta.
Propósito	Testar se o pedido com status fechado pode ser refeito.
Precondições	O usuário deve estar logado no sistema. O usuário deve estar com todos pedidos com status fechado.
Passos	1. O usuário seleciona o pedido com status fechado; 2. O usuário clica no botão repetir. 3. O sistema mostra aviso sobre o processo da ação de repetir pedido. 4. O usuário confirma clicando em ok. 5. O sistema redireciona para página de realização de pedido contendo os produtos do pedido que foi repetido. 6. O **CT04** é repetido a partir do passo 4.
Resultados Esperados	O pedido é salvo com as alterações realizadas.
Situações de Erros	Não ser gerado uma cópia do pedido.

Fonte: Elaborado pelas autoras

Quadro 14 – Caso de teste 08

CT08	Opção excluir produtos do pedido
Importância	Alta.
Propósito	Verificar se ao acionar a opção excluir produtos o mesmo é removido da lista do pedido e volta a ser uma opção para a venda.
Precondições	O usuário estar logado no sistema. Ter um pedido com status aberto. O usuário estar na tela de realizar pedido. Ter produto adicionado ao pedido.
Passos	1. O usuário clica em excluir produto. 2. O sistema exclui produto do pedido. 3. O produto aparece na lista de opções de venda.
Resultados Esperados	O produto excluído é deletado do pedido.
Situações de Erros	Opção excluir não está ativa para pedidos em abertos. O produto não é excluído. O produto excluído não fica disponível para venda.

Fonte: Elaborado pelas autoras

Quadro 15 – Caso de teste 09

CT09	Retorno ao sistema após expiração da sessão
Importância	Alta.
Propósito	Testar se após a sessão expirar o usuário consegue realizar *login* novamente para utilizar o sistema.
Precondições	O usuário deve estar logado no sistema.
Passos	1. O usuário acessa o sistema. 2. O sistema mostra o tempo que resta para a expiração da sessão no cabeçalho da página. 3. Após o tempo passar o sistema exibe uma mensagem para o usuário informando que sua sessão expirou. 4. O usuário insere seu *login* e senha para ter acesso ao sistema novamente.
Resultados Esperados	O usuário acessa novamente o sistema e usa-o normalmente.
Situações de Erros	A seção não expira depois do tempo determinado. A seção expira e não possibilita refazer *login*.

Fonte: Elaborado pelas autoras

5.4 Automação e Aplicação dos Testes

Nesta seção apresentam-se os passos para automação dos testes realizados e os resultados obtidos após sua execução. Os testes foram realizados de acordo com

a descrição dos casos de testes apresentados na subseção 5.2.2.

Primeiramente para automatizar os testes o *Selenium* IDE versão 2.4, foi instalado no navegador *Mozilla Firefox* 24.0, posteriormente se iniciou a gravação dos testes em linguagem Selenese, que grava o teste em formato de uma tabela HTML.

Após a gravação de cada teste, o mesmo foi executado, verificado e exportado para a linguagem Java para uma melhor apresentação do código. O teste exportado para Java foi realizado novamente utilizando o *Selenium* RC para verificar se executaria corretamente após a exportação.

Os testes exportados para Java são compostos por três anotações principais: *@After*, *@Before*, *@Test* que fazem parte do pacote *JUni* e são úteis para indicar atividades que devem ser executadas em momentos distintos da execução dos testes.

A *@After* é responsável por criar os objetos que são necessários antes que inicie-se o teste. O método que compõe esta anotação é o *setUp* que tem por função abrir a conexão com o servidor Selenium.

Neste método é instanciada a classe *DefaultSelenium* utilizada para a conexão, esta classe é composta por quatro parâmetros. O primeiro corresponde à máquina onde está instalado o *Selenium Server*. O segundo corresponde a porta de acesso, cujo padrão do *Selenium* é a 4444. O terceiro refere-se ao navegador quando chamado para a execução dos testes. E por último é especificado a url do site a ser testado.

No caso deste trabalho o método *setUp* ficou conforme mostra a Figura 14.

Figura 14 – Método *setUp*

```
@Before
public void setUp() throws Exception {
    selenium = new DefaultSelenium("localhost", 4444, "*chrome",
            "http://falconfvdev.eagletecnologia.com/");
    selenium.start();
}
```

Fonte: Elaborada pelas autoras

A *@After* é incumbido de liberar os recursos alocados pela *@Before*, após o teste ser executado, é composta pelo método *tearDown* que fecha a conexão com o *Selenium server*. A Figura 15 mostra este método como utilizado neste trabalho.

Figura 15 – Método *tearDown*

```
@After
public void tearDown() throws Exception {
    selenium.stop();
}
```

Fonte: Elaborada pelas autoras

A @*Test* informa que o método ao qual está ligado pode ser executado como um caso de teste. Para executar o método, primeiramente o *JUnit* constrói uma instância da classe, em seguida, chama o método anotado.

A @*Test* suporta dois parâmetros que são opcionais. O primeiro, *expect*, declara que um método de teste deve executar uma exceção. Se ele não executar uma exceção ou se executa uma exceção diferente do declarado, o teste falha. O segundo parâmetro, *timeout*, refere-se ao tempo determinado para a execução de um teste e caso demore mais que o tempo especificado o teste falha, este tempo é medido em milissegundos.

Para testar o módulo de Pedidos *Online* do sistema *FalconFV*, realizou-se os seguintes testes: fazer *login*, realizar pedidos, excluir, editar e repetir pedido, salvar pedido, excluir produtos do pedido e refazer *login* após expirar o tempo da sessão por inatividade. Os códigos gerados após a realização dos testes, tanto em HTML como em Java, são apresentados após a descrição de cada teste realizado.

Todos os testes realizados foram inicializados a partir do *login* do usuário, prevendo a execução dos testes com o *Selenium* RC, pois como explicado anteriormente esta versão da ferramenta faz uma chamada ao navegador e URL especificados em @After para então começar a execução dos testes, por isso há a necessidade de realizar o *login* em cada teste. Este tipo de exceção pode ser tratado quando é realizada uma suíte de teste, onde os testes são executados em sequência, o que não foi realizado neste trabalho porque o objetivo era testar cada componente separadamente.

O primeiro teste automatizado consiste na função de realizar *login* no sistema. Para realizar este teste foram consideradas as seguintes situações: a) inserção do usuário e senha corretos; b) inserção de senha inválida; c) inserção da senha com quantidade de caracteres inferior a especificada; d) tentativa de *login* com campos vazios. De acordo com as situações descritas foi efetuada a automatização deste

teste, os códigos gerados encontram-se nos Quadros 16 e 17:

Quadro 16 – Teste realizar *login* no sistema (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" />
<title>CT01</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT01</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
        <td>id=senha</td>
        <td>123@abc</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=Opções</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=Sair</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
```

(conclusão)

Código do teste em HTML

```
        <td>id=senha</td>
        <td>123456678</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>assertText</td>
        <td>id=div_error</td>
        <td>Credências inválidas!</td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
        <td>id=senha</td>
        <td>123</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>assertText</td>
        <td>//form[@id='form_autentica']/div/p[2]/label/span</td>
        <td>exact:O tamanho de '***' é inferior a 4 caracteres</td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>selectWindow</td>
        <td>null</td>
        <td></td>
</tr>
<tr>
        <td>assertText</td>
        <td>//form[@id='form_autentica']/div/p[2]/label/span</td>
        <td>O valor é obrigatório e não pode estar vazio</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 17 – Teste realizar *login* no sistema (Java)

Código do teste em Java

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

public class CT01 {
	private Selenium selenium;

	@Before
	public void setUp() throws Exception {
		selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
		selenium.start();
	}

	@Test
	public void testCT01() throws Exception {
		selenium.open("/");
		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123@abc");
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("50000");
		selenium.click("link=Opções");
		selenium.waitForPageToLoad("50000");
		selenium.click("link=Sair");

		selenium.waitForPageToLoad("50000");
		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123456678");
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("50000");
		assertEquals("Credências inválidas!", selenium.getText("id=div_error"));

		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123");
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("50000");
		assertTrue(selenium.getText("//form[@id='form_autentica']/div/p[2]/label/span").ma
	tches("^exact:O tamanho de '[\\s\\S]*[\\s\\S]*[\\s\\S]*' é inferior a 4 caracteres$"));

		selenium.type("id=login", "05006141000135");
		selenium.selectWindow("null");
		assertEquals("O valor é obrigatório e não pode estar vazio",
selenium.getText("//form[@id='form_autentica']/div/p[2]/label/span"));
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("50000");
	}

	@After
	public void tearDown() throws Exception {
		selenium.stop();
	}
}
```

Fonte: Elaborado pelas autoras

A execução de todas as situações especificadas para a efetuação do *login* foram realizadas corretamente. Como pode ser visualizado no código acima para a situação a) o *login* foi realizado com sucesso; para a b) mostrou a mensagem de "Credenciais Inválidas". Quando executada a situação c) a mensagem "O tamanho *** é inferior a 4 caracteres" e para a d) apareceu a mensagem "O valor é obrigatório e não pode estar vazio".

O próximo teste, realizar novo pedido, foi efetuado para testar a funcionalidade do botão novo pedido. E juntamente com ele fez-se o teste correspondente ao do botão salvar pedido. Para fazer este teste especificaram-se os produtos e a quantidade desejada e posteriormente, acionado o botão de adicionar itens ao pedido. Estes passos geraram os códigos que aparecem nos Quadros 18 e 19:

Quadro 18 – Teste realizar novo pedido (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" />
<title>CT02</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT02</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
        <td>id=senha</td>
        <td>123@abc</td>
</tr>
<tr>
```

(continua)

Código do teste em HTML

```
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>id=novo_pedido</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='28']</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=#myModalNulo &gt; div.modal-footer &gt; #cancela_modal</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd28</td>
        <td>5</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='28']</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd114</td>
        <td>25</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='114']</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd177</td>
        <td>4</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='177']</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=Salvar</td>
        <td></td>
</tr>
```

(conclusão)

Código do teste em HTML
<tr> <td>clickAndWait</td> <td>id=salva_pedido</td> <td></td> </tr> </tbody></table> </body> </html>

Fonte: Elaborado pelas autoras

Quadro 19 – Teste realizar novo pedido (Java)

Código do teste em Java

package com.example.tests;
import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
 	
public class CT02 {
private Selenium selenium;
@Before
public void setUp() throws Exception {
selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
selenium.start();
}
@Test
public void testCT02() throws Exception {
selenium.open("/");
selenium.type("id=login", "05006141000135");
selenium.type("id=senha", "123@abc");
selenium.click("css=button.btn.btn-primary");
selenium.waitForPageToLoad("50000");
selenium.click("id=novo_pedido");
selenium.waitForPageToLoad("50000");
selenium.click("id=qtd28");
selenium.waitForPageToLoad("50000");
selenium.click("id=qtd28");
selenium.waitForPageToLoad("50000");
selenium.click("//button[@id='28']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd114", "5");
selenium.click("//button[@id='114']");
selenium.waitForPageToLoad("50000");
selenium.click("link=Salvar");
selenium.waitForPageToLoad("50000");
selenium.click("id=salva_pedido");
selenium.waitForPageToLoad("50000");
}
@After
public void tearDown() throws Exception {
selenium.stop();
}
}

Fonte: Elaborado pelas autoras

Este teste também executou sem apresentar nenhum tipo de erro. O campo onde se especifica a quantidade desejada do produto foi programado prevendo as possíveis situações de erro, não permite desta forma a digitação de letras ou valor nulo. Ao clicar no botão de adicionar, caso o campo de quantidade esteja vazio é exibida uma mensagem indicando que o preenchimento do campo é obrigatório. Outra observação é que o botão salvar só é exibido após a adição de um produto no pedido. O botão cancelar pode ser acionado a qualquer momento para interromper esta ação.

O teste realizado para verificar a função de editar um pedido, cujos códigos estão nos Quadros 20 e 21, só é possível em pedidos com o status aberto, desta forma, primeiramente fez-se o processo de realizar um pedido. Posteriormente, realizou-se a edição deste pedido clicando no botão editar, ao realizar esta ação o sistema direciona o usuário para a página que permite a adição ou exclusão de produtos ao pedido. Então foi excluído um produto da lista e acionado o botão salvar, após a confirmação de salvar as alterações realizadas, houve o redirecionamento para a página de detalhes do pedido. Não houve nenhum tipo de erro a executar o teste gravado. Os códigos referentes a este teste localizam-se nos Quadros 20 e 21.

Quadro 20 – Teste editar pedido (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" />
<title>CT04</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT04</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/</td>
        <td></td>
</tr>
```

(continua)

Código do teste em HTML

```
<tr>
	<td>type</td>
	<td>id=login</td>
	<td>05006141000135</td>
</tr>
<tr>
	<td>type</td>
	<td>id=senha</td>
	<td>123@abc</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=button.btn.btn-primary</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=novo_pedido</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=qtd28</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=qtd28</td>
	<td>7</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>//button[@id='28']</td>
	<td></td>
</tr>
<tr>
	<td>type</td>
	<td>id=qtd114</td>
	<td>5</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>//button[@id='114']</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=Salvar</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=salva_pedido</td>
	<td></td>
</tr>
```

(conclusão)

Código do teste em HTML

```
<tr>
	<td>click</td>
	<td>id=btn_edita_pedido</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>css=td &gt; #179 &gt; i.icon-trash.icon-white</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>link=Salvar</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>id=salva_pedido</td>
	<td></td>
</tr>

</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 21 – Teste editar pedido (Java)

(continua)

Código do teste em Java

```
package CT04;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CT04 {
	private Selenium selenium;

	@Before
	public void setUp() throws Exception {
		selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
		selenium.start();
	}

	@Test
	public void testCT04() throws Exception {
		selenium.open("/");
		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123@abc");
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("20000");
		selenium.click("id=novo_pedido");
```

(conclusão)

Código do teste em Java

```
                selenium.waitForPageToLoad("20000");
                selenium.click("id=qtd28");
                selenium.waitForPageToLoad("20000");
                selenium.click("id=qtd28");
                selenium.waitForPageToLoad("20000");
                selenium.click("//button[@id='28']");
                selenium.waitForPageToLoad("20000");
                selenium.type("id=qtd114", "5");
                selenium.click("//button[@id='114']");
                selenium.waitForPageToLoad("20000");
                selenium.click("link=Salvar");
                selenium.waitForPageToLoad("20000");
                selenium.click("id=salva_pedido");
                selenium.waitForPageToLoad("20000");
                selenium.click("id=btn_edita_pedido");
                selenium.click("css=td > #179 > i.icon-trash.icon-white");
                selenium.click("link=Salvar");
                selenium.click("id=salva_pedido");
        }

@After
        public void tearDown() throws Exception {
                selenium.stop();
        }
```

Fonte: Elaborado pelas autoras

A exclusão de pedidos só é permitida para produtos com status aberto, quando o status está fechado a opção é ocultada ao usuário. Para testar esta função efetuou-se o *login* no sistema, e como existia pedidos com status aberto, a página de detalhes do pedido foi carregada. Foi realizada uma edição no pedido e em seguida o botão excluir foi acionado. O sistema então solicitou a confirmação da ação e o pedido foi excluído da lista de pedidos. Automaticamente o sistema redireciona o usuário para a página de histórico de pedidos e o botão novo pedido é exibido. Os códigos referentes à gravação deste teste podem ser visualizados nos Quadros 22 e 23. O teste não apresentou falhas ao ser realizado.

Quadro 22 – Teste excluir pedido (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://www.falconfvdev.eagletecnologia.com/" />
```

(continua)

Código do teste em HTML

```
<title>CT05</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT05</td></tr>
</thead><tbody>
<tr>
	<td>open</td>
	<td>/</td>
	<td></td>
</tr>
<tr>
	<td>type</td>
	<td>id=login</td>
	<td>05006141000135</td>
</tr>
<tr>
	<td>type</td>
	<td>id=senha</td>
	<td>123@abc</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=button.btn.btn-primary</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>id=btn_edita_pedido</td>
	<td></td>
</tr>
<tr>
	<td>type</td>
	<td>id=qtd114</td>
	<td>3</td>
</tr>
<tr>
	<td>click</td>
	<td>//button[@id='114']</td>
	<td></td>
</tr>
<tr>
	<td>type</td>
	<td>id=qtd177</td>
	<td>5</td>
</tr>
<tr>
	<td>click</td>
	<td>//button[@id='177']</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>link=Salvar</td>
	<td></td>
</tr>
```

(conclusão)

Código do teste em HTML

```
<tr>
	<td>click</td>
	<td>id=salva_pedido</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>id=btn_voltar_historico</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>css=span.label.label-success</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>id=btn_deletaPedido</td>
	<td></td>
</tr>
<tr>
	<td>click</td>
	<td>id=deleta_pedido</td>
	<td></td>
</tr>

</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 23 – Teste excluir pedido (Java)

(continua)

Código do teste em Java

```
package CT05;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.regex.Pattern;

public class CT05 {
	private Selenium selenium;

	@Before
	public void setUp() throws Exception {
		selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://www.falconfvdev.eagletecnologia.com/");
		selenium.start();
	}
```

(conclusão)

Código do teste em Java

```
@Test
	public void testCT05() throws Exception {
		selenium.open("/");
		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123@abc");
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("20000");
		selenium.click("id=btn_edita_pedido");
		selenium.type("id=qtd114", "3");
		selenium.click("//button[@id='114']");
		selenium.type("id=qtd177", "5");
		selenium.click("//button[@id='177']");
		selenium.click("link=Salvar");
		selenium.click("id=salva_pedido");
		selenium.click("id=btn_voltar_historico");
		selenium.click("css=span.label.label-success");
		selenium.click("id=btn_deletaPedido");
		selenium.click("id=deleta_pedido");
	}

	@After
	public void tearDown() throws Exception {
		selenium.stop();
	}
}
```

Fonte: Elaborado pelas autoras

A funcionalidade embutida no botão voltar ao histórico permite o retorno do usuário ao histórico de pedidos. A função é ativada somente quando existe pedido com status aberto, onde pode ser utilizada ao finalizar um pedido, ou quando o usuário efetuar *login* no sistema, e for direcionado para a página de detalhes do pedido. O retorno ao histórico permite a visualização dos pedidos feitos estejam com status fechado ou não, e são organizados pela data de criação em ordem cronológica decrescente. Este teste foi um dos mais simples de ser realizado e não foram encontrados erros nesta função e seus códigos são exibidos nos Quadros 24 e 25.

Quadro 24 – Teste voltar ao histórico de pedidos (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

(conclusão)

Código do teste em HTML

```
<link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" />
<title>CT06</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT06</td></tr>
</thead><tbody>
<tr>
	<td>open</td>
	<td>/</td>
	<td></td>
</tr>
<tr>
	<td>type</td>
	<td>id=login</td>
	<td>05006141000135</td>
</tr>
<tr>
	<td>type</td>
	<td>id=senha</td>
	<td>123@abc</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=button.btn.btn-primary</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=#pedido_unico &gt; div.modal-header &gt; #myModalLabel</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=btn_voltar_historico</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=div.hero-unit.style2 &gt; h3</td>
	<td></td>
</tr>
</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 25 – Teste voltar ao histórico de pedidos (Java)

(continua)

Código do teste em Java

```
package CT07;
import com.thoughtworks.selenium.*;
import org.junit.After;
```

(conclusão)

Código do teste em Java

```
import org.junit.Before;
import org.junit.Test;


public class CT07 {
	private Selenium selenium;

	@Before
	public void setUp() throws Exception {
		selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
		selenium.start();
	}

	@Test
	public void testCT07() throws Exception {
		selenium.open("/ecommerce/");
		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123@abc");
		selenium.click("css=button.btn.btn-primary");
		selenium.waitForPageToLoad("50000");
		selenium.click("css=#status > span.label.label-important");
		selenium.waitForPageToLoad("50000");
		selenium.click("link=Repetir");
		selenium.waitForPageToLoad("50000");
		selenium.click("id=btn_repete_pedido");
		selenium.waitForPageToLoad("50000");
		selenium.click("link=Salvar");
		selenium.waitForPageToLoad("50000");
		selenium.click("id=salva_pedido");
		selenium.waitForPageToLoad("50000");
	}

	@After
	public void tearDown() throws Exception {
		selenium.stop();
	}
}
```

Fonte: Elaborado pelas autoras

O teste repetir pedido requer que na tela de históricos não exista pedidos com status aberto, mas tenha pedidos com status fechados. Como próprio nome indica, permite que o usuário reproduza um pedido. Não necessariamente o pedido tem que ser igual ao de origem, pois é permitida a edição do pedido. Para realizar este teste, logou-se no sistema e, logo após, clicou-se no status do pedido, este processo abre a página de detalhes de pedido que possui o botão repetir. Este botão foi acionado e uma mensagem de alerta foi exibida informando que os produtos indisponíveis no estoque são excluídos automaticamente do pedido. Durante este teste optou-se somente por salvar o pedido e não realizar alterações. Não foi detectado erros neste

teste. Nos Quadros 26 e 27 estão expostos os códigos provenientes deste teste.

Quadro 26 – Teste repetir pedido (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" />
<title>CT07</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT07</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/ecommerce/</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
        <td>id=senha</td>
        <td>123@abc</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=#status &gt; span.label.label-important</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=Repetir</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>id=btn_repete_pedido</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=Salvar</td>
```

(conclusão)

Código do teste em HTML

```
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>id=salva_pedido</td>
        <td></td>
</tr>
</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 27 – Teste repetir pedido (Java)

(continua)

Código do teste em Java

```
package CT07;
import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CT07 {
        private Selenium selenium;
        @Before
        public void setUp() throws Exception {
                selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
                selenium.start();
        }
        @Test
        public void testCT07() throws Exception {
                selenium.open("/ecommerce/");
                selenium.type("id=login", "05006141000135");
                selenium.type("id=senha", "123@abc");
                selenium.click("css=button.btn.btn-primary");
                selenium.waitForPageToLoad("50000");
                selenium.click("css=#status > span.label.label-important");
                selenium.waitForPageToLoad("50000");
                selenium.click("link=Repetir");
                selenium.waitForPageToLoad("50000");
                selenium.click("id=btn_repete_pedido");
                selenium.waitForPageToLoad("50000");
                selenium.click("link=Salvar");
                selenium.waitForPageToLoad("50000");
                selenium.click("id=dataCri");
                selenium.waitForPageToLoad("50000");
                selenium.click("id=btn_deletaPedido");
                selenium.waitForPageToLoad("50000");
                selenium.click("id=deleta_pedido");
                selenium.waitForPageToLoad("50000");
                selenium.click("id=salva_pedido");
                selenium.waitForPageToLoad("50000");
        }
```

(conclusão)

Código do teste em Java

@After
public void tearDown() throws Exception {
selenium.stop();
}
}

Fonte: Elaborado pelas autoras

Outro teste realizado foi o excluir produtos do pedido. Este teste verifica se a exclusão de produtos do pedido com status aberto é efetuada corretamente. Para ser realizado de forma certa após a deleção do produto do pedido, o item deve aparecer novamente como uma das opções de produtos que pode ser inserida no mesmo. Neste teste após *login* do usuário, foi criado um novo pedido inserindo os produtos e o pedido então foi salvo. Após carregar a página de detalhes do pedido, a opção editar foi acionada, o produto a ser excluído foi escolhido na lista pelo usuário e clicou-se no ícone responsável por fazer a exclusão dos itens. O item excluído então apareceu novamente na lista de produtos disponíveis e desta forma poderá ser adicionada ao pedido posteriormente se necessário. O teste não retornou nenhum tipo de erro. Os códigos dos Quadros 28 e 29 referem-se a este teste.

Quadro 28 – Teste excluir produtos do pedido (HTML)

(continua)

Código do teste em HTML
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"> <head profile="http://selenium-ide.openqa.org/profiles/test-case"> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /> <link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" /> <title>CT08</title> </head> <body> <table cellpadding="1" cellspacing="1" border="1"> <thead> <tr><td rowspan="1" colspan="3">CT08</td></tr> </thead><tbody> <tr> <td>open</td> <td></td> <td></td> </tr> <tr> <td>type</td> <td>id=login</td>

(continua)

Código do teste em HTML

```
<td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
        <td>id=senha</td>
        <td>123@abc</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>id=novo_pedido</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd28</td>
        <td>2</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='28']</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd115</td>
        <td>2</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='115']</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd114</td>
        <td>2</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='114']</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=qtd179</td>
        <td>1</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//button[@id='179']</td>
        <td></td>
```

(continua)

Código do teste em HTML

```
</tr>
<tr>
	<td>type</td>
	<td>id=qtd224</td>
	<td>1</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>//button[@id='224']</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=Salvar</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=salva_pedido</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=btn_edita_pedido</td>
	<td></td>
</tr>
<tr>
	<td>assertText</td>
	<td>xpath=(//span[@id='produto'])[5]</td>
	<td>TONYU SOJA ABACAXI 200 ML</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=td &gt; #224 &gt; i.icon-trash.icon-white</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>css=td &gt; #179 &gt; i.icon-trash.icon-white</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=Salvar</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=salva_pedido</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=btn_edita_pedido</td>
	<td></td>
</tr>
<tr>
```

(conclusão)

Código do teste em HTML

```
        <td>assertText</td>
        <td>//tr[@id='115']/td[3]</td>
        <td>B.LACTEA UHT FRUTAS</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=td &gt; #115 &gt; i.icon-trash.icon-white</td>
        <td></td>
</tr>
<tr>
        <td>verifyTable</td>
        <td>id=tab_esquerda.1.1</td>
        <td>B.LACTEA UHT FRUTAS</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=Salvar</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>id=salva_pedido</td>
        <td></td>
</tr>
</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 29 – Teste excluir produtos do pedido (Java)

(continua)

Código do teste em Java

```
package CT08;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CT08 {
        private Selenium selenium;

        @Before
        public void setUp() throws Exception {
                selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
                selenium.start();
        }

        @Test
        public void testCT10() throws Exception {
                selenium.open("/");
```

(conclusão)

Código do teste em Java

selenium.type("id=login", "05006141000135");
selenium.type("id=senha", "123@abc");
selenium.click("css=button.btn.btn-primary");
selenium.waitForPageToLoad("90000");
selenium.click("id=novo_pedido");
selenium.waitForPageToLoad("90000");
selenium.type("id=qtd28", "1");
selenium.click("//button[@id='28']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd115", "3");
selenium.click("//button[@id='115']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd114", "1");
selenium.click("//button[@id='114']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd177", "4");
selenium.click("//button[@id='177']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd179", "1");
selenium.click("//button[@id='179']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd223", "5");
selenium.click("//button[@id='223']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd224", "5");
selenium.click("//button[@id='224']");
selenium.waitForPageToLoad("50000");
selenium.type("id=qtd225", "12");
selenium.click("//button[@id='225']");
selenium.waitForPageToLoad("50000");
selenium.click("link=Salvar");
selenium.waitForPageToLoad("50000");
selenium.click("id=salva_pedido");
selenium.waitForPageToLoad("50000");
selenium.click("id=btn_edita_pedido");
selenium.waitForPageToLoad("50000");
selenium.click("css=td > #177 > i.icon-trash.icon-white");
selenium.waitForPageToLoad("50000");
selenium.click("css=td > #115 > i.icon-trash.icon-white");
selenium.waitForPageToLoad("50000");
selenium.click("//a[@id='114']");
selenium.waitForPageToLoad("50000");
selenium.click("link=Salvar");
selenium.waitForPageToLoad("50000");
selenium.click("id=salva_pedido");
selenium.waitForPageToLoad("50000");

Fonte: Elaborado pelas autoras

O teste, retorno ao sistema após expiração da sessão, verifica se quando o sistema fica ocioso por 15 minutos, a sessão expira e pede ao usuário para inserir *login* e senha novamente voltando para a tela onde o usuário encontrava-se. O teste procedeu da seguinte forma, foi realizado o *login* no sistema, como existia um pedido

com status aberto, a página de detalhes do pedido foi carregada. A opção editar pedido foi acessada e o sistema ficou 15 minutos inativo, após este tempo, a opção término da sessão foi ativada solicitando um novo *login* do usuário. Ao realizar o teste com o usuário e senha válidos foi efetuado o *login* novamente e o sistema pode ser utilizado normalmente. Os códigos deste teste podem ser vistos nos Quadros 30 e 31.

Quadro 30 – Teste retorno ao sistema após expiração da sessão (HTML)

(continua)

Código do teste em HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://falconfvdev.eagletecnologia.com/" />
<title>CT09</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">CT09</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>id=login</td>
        <td>05006141000135</td>
</tr>
<tr>
        <td>type</td>
        <td>id=senha</td>
        <td>123@abc</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>css=button.btn.btn-primary</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>id=btn_edita_pedido</td>
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=OK</td>
```

(conclusão)

Código do teste em HTML

```
<td></td>
</tr>
<tr>
	<td>verifyText</td>
	<td>css=#winExpiredSession &gt; div.modal-header &gt; #myModalLabel</td>
	<td>Sessão Expirada</td>
</tr>
<tr>
	<td>type</td>
	<td>id=login</td>
	<td>05006141000135</td>
</tr>
<tr>
	<td>type</td>
	<td>id=senha</td>
	<td>123@abc</td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>id=btnLogin</td>
	<td></td>
</tr>
</tbody></table>
</body>
</html>
```

Fonte: Elaborado pelas autoras

Quadro 31 – Teste Retorno ao sistema após expiração da sessão (Java)

(continua)

Código do teste em Java

```
package CT09;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.regex.Pattern;

public class CT09 {
	private Selenium selenium;

	@Before
	public void setUp() throws Exception {
		selenium = new DefaultSelenium("localhost", 4444, "*firefox",
"http://falconfvdev.eagletecnologia.com/");
		selenium.start();
	}

	@Test
	public void testCT09() throws Exception {
		selenium.open("/");
		selenium.type("id=login", "05006141000135");
		selenium.type("id=senha", "123@abc");
```

(conclusão)

Código do teste em Java

```
        selenium.click("css=button.btn.btn-primary");
                selenium.waitForPageToLoad("50000");
                selenium.click("id=btn_edita_pedido");
                selenium.waitForPageToLoad("50000");
                selenium.click("link=OK");
                selenium.waitForPageToLoad("50000");
                verifyEquals("Sessão Expirada", selenium.getText("css=#winExpiredSession >
div.modal-header > #myModalLabel"));
                selenium.type("id=login", "05006141000135");
                selenium.type("id=senha", "123@abc");
                selenium.click("id=btnLogin");
                selenium.waitForPageToLoad("50000");
        }

        @After
        public void tearDown() throws Exception {
                selenium.stop();
        }
}
```

Fonte: Elaborado pelas autoras

Os testes gerados pelo Selenium IDE, como citado anteriormente, são apresentados em uma tabela html, que o usuário poderá editar caso necessário. A Figura 16 exemplifica como o código-fonte do teste é exibido por meio de um navegador *web*. Os dados mostrados na Figura 16 referem-se ao título, comandos utilizados, o alvo e o valor do teste.

Figura 16 – Teste em HTML

CT01		
open	/	
type	id=login	05006141000135
type	id=senha	123@abc
clickAndWait	css=button.btn.btn-primary	
clickAndWait	link=Opções	
clickAndWait	link=Sair	

Fonte: Eladorado pelas autoras

6 ANÁLISE DOS RESULTADOS

Para realizar os testes utilizou-se uma versão do sistema *FalconFV online*, pois apesar de já estar ativo seu desenvolvimento ainda não foi finalizado, atualmente estão sendo implementadas algumas regras de negócio.

Inicialmente foram encontradas algumas situações de erro que refletiam no funcionamento do sistema e impediam a realização dos testes. Tais problemas foram reportados ao engenheiro de *software* da empresa, para aplicar as correções.

O principal erro do sistema estava na configuração da Sessão de pedidos, Figura 17, opção que permite ao gestor determinar quando o cliente poderá realizar os pedidos e o dia que ele receberá a mercadoria. O erro era que o sistema estava fazendo a análise para a data de fechamento do pedido de forma errada, e assim que o pedido era realizado o seu status ficava fechado.

Para corrigir este impasse, segundo o engenheiro de *software*, a *trigger* responsável por controlar o tempo de duração da sessão de pedidos foi alterada. Este módulo do sistema não estava disponível para o acesso durante a realização da pesquisa por ter grande influência no funcionamento do sistema *online*, não sendo possível aprofundar os estudos sobre o mesmo.

Figura 17 – Tela de configuração da sessão de pedidos

Fonte: Eagle Tecnologia, 2013

Um fator que influenciou na análise dos resultados, após a execução de cada teste, foi a não existência da documentação do sistema que facilitaria a verificação dos resultados, ou seja, se as respostas obtidas eram as esperadas.

Como o sistema testado está alocado em um servidor *web*, a dependência da velocidade da *internet*, dificultou um pouco a realização dos testes, isso porque o tempo de resposta das requisições AJAX variava de acordo com a taxa de transmissão da conexão de *internet*, podendo ser um tempo elevado para as configurações de *timeout* feitas no *Selenium* e o que colaborava para o aumento do tempo de execução de cada teste.

Apesar dos problemas ocorridos no início do desenvolvimento da pesquisa, através dos testes executados pode-se perceber a grande utilidade da ferramenta *Selenium* IDE para automatizar testes e facilitar a percepção do testador diante dos resultados que a ferramenta expõe a cada execução do teste automatizado.

A ferramenta oferece os mais diversos tipos de recursos em sua linguagem nativa, o Selenese, para a montagem dos testes, além de permitir a exportação dos mesmos para uma linguagem de programação, isso proporciona ao testador a montagem de testes mais robustos que possam testar todos os possíveis resultados dos testes sejam eles positivos ou não.

Uma das grandes vantagens do *Selenium* IDE é a possibilidade armazenar os testes em conjunto, formando o que se pode chamar de suíte de testes, desta forma os testes serão executados em sequência. Porém, é de boa prática primeiramente a automação particular de cada teste, para que as funcionalidades sejam tratadas conforme sua particularidade.

A automação de testes com a ferramenta *Selenium* começa a partir do *Selenium* IDE, isso porque permite a gravação das interações que são realizadas na página, formando a base do teste e este pode ser complementado através de comandos específicos da linguagem de programação. É importante frisar que a elaboração dos *scripts* requer um esforço maior que a execução do mesmo

Uma grande vantagem que pode ser observada é que após a gravação do teste, o testador não precisa mais interagir com o sistema, os passos são executados automaticamente, e os resultados podem ser visualizados através do *feedback* que a ferramenta oferece, pontuando o que acontece durante a execução do teste e, quando um erro é encontrado, informa qual foi o erro.

Outra vantagem da ferramenta é que permite a realização de testes de regressão, que visa garantir que o programa ainda satisfaz seus requisitos após alguma alteração. Esta vantagem é pelo fato de que os testes automatizados não

precisarão ser refeitos, apenas aplicados ao sistema.

Os testes realizados no módulo de Pedidos *Online* do sistema *FalconFV* foram funcionais, porém a ferramenta Selenium permite também a execução de testes de regressão e desempenho, além de testes de compatibilidade entre navegadores e plataformas diferentes, para isso a utilização do RC é indispensável já que o IDE é um complemento do navegador *Mozilla Firefox*.

Os resultados obtidos na realização dos testes serão reportados para a empresa cedente do *software* testado, os testes entregues para o engenheiro de *software* da empresa e, caso seja de interesse da empresa, os testes devem ser complementados para então formar uma suíte de teste e executá-los novamente na versão final do sistema, já que este ainda passa por modificações.

A opção pela exportação dos testes para a linguagem Java foi pela grande popularização da linguagem e sua facilidade de manipulação. Outro fator que influenciou foi que as pesquisas realizadas sempre indicavam a utilização de Java, juntamente com o *Selenium* RC e com a utilização da ferramenta Eclipse.

Para que os *scripts* de testes sejam produzidos com maior facilidade é importante que o *software* seja desenvolvido seguindo uma padronização com o uso da linguagem *web* utilizada, é necessário identificar cada componente da página para facilitar sua localização.

As empresas, ao automatizar seus testes, permitem que estes sejam realizados de forma mais amena e com maior precisão. Os testadores são indispensáveis para construção dos testes, porém não precisam se preocupar em executá-los, ainda possuem relatórios sobre a execução de testes que proporcionam a detecção de forma mais ágil dos erros. Além disso, os testes podem ser reexecutados sem um esforço repetitivo das atividades realizadas pelo testador, ficando este disponível para realizar outras atividades. Esta vantagem é principalmente aproveitada por empresas com pequenas equipes, onde os programadores também são testadores.

A documentação do *software* é importantíssima na fase de teste do sistema, porque evita que os testes fiquem dependentes do desenvolvedor para explicar como o sistema deve funcionar, quais são as entradas e como devem ser as saídas. Também evita que na fase de teste seja gasto tempo para o conhecimento do sistema pelo testador.

Para o uso eficiente de qualquer ferramenta de automação, é necessário o

conhecimento de suas funções e recursos oferecidos, os profissionais devem ser qualificados para usá-las para a elaboração de testes coerentes com o que deseja testar.

O *Selenium* se classifica como a ferramenta de automação de testes mais utilizada atualmente para testar aplicações *web* e ainda está sendo complementada por seus mantenedores com a aglomeração recursos para permitir que os testes sejam desempenhados com alto nível de complexidade, possibilitando a verificação mais detalhada das funções testadas.

7 CONCLUSÃO

Para garantir a qualidade total de um *software* desenvolvido é necessária a implantação da fase de teste como um ciclo de seu desenvolvimento. Porém, o principal fator que influência na execução dos testes e que, muitas vezes é o motivo para a sua não efetuação, é o tempo gasto com testes manuais.

Em sistema *web*, onde geralmente a equipe responsável pelo projeto é pequena, este fator é ainda mais agravante, com isso se faz necessário à utilização de ferramentas que reduzam o tempo dedicado aos testes e permitam sua execução de forma qualificada. Uma boa tática para amenizar o tempo dedicado à execução esta atividade, é a utilização de um bom processo de automação que contribui para o desenvolvimento mais seguro.

A ferramenta *Selenium* IDE pode ser utilizada para automação deste processo, diminuindo o tempo desta atividade e contribuindo para firmar a qualidade do *software*. O estudo e utilização da ferramenta *Selenium* IDE neste trabalho teve como intuito avaliar sua eficiência na automação e execução dos testes para verificar o quanto a ferramenta colabora para auxiliar a atividade de teste.

As dificuldades encontradas no processo de desenvolvimento deste trabalho referem-se ao: referencial a respeito da ferramenta, pois a maior gama de informações encontra-se em língua estrangeira, e a falta de documentação do *software* testado, pois foi necessária a intervenção do engenheiro de *software* da empresa para o conhecimento do sistema, o que delongou o processo de automação devido a sua disponibilidade para atender aos questionamentos propostos.

Pode-se perceber que a automação, apesar de ser uma atividade facilitadora da fase de testes, não é um processo fácil, pois necessita de dedicação e esforço para identificar os testes que podem ser automatizados. Isso influi em um gasto maior de tempo, porém este tempo é recompensado na execução e análise dos resultados após a automação.

A predefinição dos testes facilitou o processo de automação porque não foi necessário pensar nos possíveis testes, apenas fazer seu levantamento junto ao engenheiro de *software* da empresa, para então desenvolver os casos de testes a serem automatizados.

Os objetivos deste trabalho foram atendidos, uma vez que a ferramenta

Selenium permitiu a aplicação dos testes predefinidos e com isso mostrou-se como é o processo de automação de testes através desta ferramenta. Além disso, permitiu o conhecimento do processo de exportação e entendimento dos testes em uma linguagem de programação.

É importante ressaltar que raramente um teste será aplicado apenas uma vez para certificar o funcionamento de uma determinada funcionalidade de um *software,* e este é mais um ponto favorável a automação de teste que enfatiza a grande utilidade de executar testes de forma automática.

A ferramenta garantiu o teste das funcionalidades do sistema de modo prático e preciso, isso facilita a verificação dos problemas que por ventura venha a ocorrer. Os testes realizados no módulo de Pedidos *Online* do sistema *FalconFV* não detectaram nenhum tipo de erro, mas ainda assim comprovaram a utilidade do *Selenium* IDE em automatizar testes.

A pesquisa realizada oferece respaldo para alguns trabalhos futuros os quais são:

1. Executar testes de regressão no módulo de Pedidos *Online* do sistema *FalconFV* após a finalização das alterações que estão sendo realizadas, para verificar a eficiência da ferramenta *Selenium* na execução deste tipo de teste.
2. Realizar uma análise comparativa entre as ferramentas destinadas a automatizar testes *web*, sejam elas comerciais ou *open source*, através da execução de testes em um mesmo sistema *web* e observação dos resultados que cada ferramenta gera, visando apontar suas vantagens e desvantagens.
3. Criar um servidor de teste automatizado utilizando o *Selenium* RC com códigos em linguagem de programação Java vinculado ao desenvolvimento do sistema *FalconFV*.
4. Elaborar uma metodologia de desenvolvimento usando *Ajax* para melhorar o tempo e a agilidade da execução dos testes com a ferramenta *Selenium*.

REFERÊNCIAS

APACHE Software Foundation. **Apache JMeter**, 2013. Disponível em: <http://jmeter.apache.org/index.html>. Acessado em: 03/06/2013.

BADBOY Software. Disponível em: <http://www.badboy.com.au/>. Acessado em: 03/06/2013.

CAELUM. **Eclipse IDE**. Disponível em: <http://www.caelum.com.br/apostila-java-orientacao-objetos/eclipse-ide/>. Acessado em: 06/10/2013.

CAETANO, C. **Automação e Gerenciamento de Testes**, 2007. Disponível em: <www.digitalworks.eti.br/DigitalWorks/Upload/Curso/automa_teste.pdf>. Acessado em: 06/03/2013.

CAMPOS, F.M. **Série Conhecendo Ferramentas de Automação para Teste de Software - WEBLOAD - Parte 1**, 2013. Disponível em: <http://www.linhadecodigo.com.br/artigo/1489/serie-conhecendo-ferramentas-de-automacao-para-teste-de-*software-web*load-parte-1.aspx#ixzz2VBMr0lmZ>. Acessado em: 03/06/2013.

CANOO WebTest. Disponível em: <http://webtest.canoo.com/webtest/manual/WebTestHome.html>. Acessado em: 03/06/2013.

CHASE, N. **Entendendo a Zend Framework**. Disponível em: <http://www.ibm.com/developerworks/br/library/os-php-zend1/>. Acessado em 10/09/2013.

COHEN, F. **Getting Started with Selenium***,* 2009. Disponível em: <http://refcardz.dzone.com/refcardz/getting-started-Selenium>. Acessado em: 22/04/2013.

DANTAS, M.; CAVALCANTE, V. **Pesquisa Qualitativa e Pesquisa Quantitativa**. Disponível em: <http://pt.scribd.com/doc/14344653/Pesquisa-qualitativa-e-quantitativa>. Acessado em: 21/10/2013.

DIAS, C. **Pesquisa qualitativa – características gerais e referências**. <http://www.reocities.com/claudiaad/qualitativa.pdf>. Acessado em: 21/10/2013.

DOCUMENTAÇÃO Selenium, 2013. Disponível em: <http://docs.Seleniumhq.org/docs/>. Acessado em: 01/05/2013.

EAGLE Tecnologia. **A Empresa**, 2012. Disponível em: <http://www.eagletecnologia.com/site/index.php?area=aempresa>. Acessado em: 22/12/2012.

FILHO, A. M. S. **Sobre a Importância da Arquitetura de Software no Desenvolvimento de Sistemas de Software**, 2006**.** Disponível em: <http://www.unibratec.edu.br/tecnologus/wp-content/uploads/2006/08/n1_silvafilho.pdf>. Acessado em: 05/04/2013.

FLANAGAN, D. **JavaScript – O guia definitivo**. Porto Alegre: Editora S.A,2002. Disponível em: <http://books.google.com.br/books?id=ZuTxAKCczMkC&pg=PA28&lpg=PA28&dq=javascript+o+guia+definitivo+download&source=bl&ots=F72KObNFjL&sig=1HsoTcR0SVIj9BuiwZ-pXnm0dgQ&hl=pt-BR&sa=X&ei=H6dWUvywFPPl4AP1sID4Bg&ved=0CC4Q6AEwAA#v=onepage&q=javascript%20o%20guia%20definitivo%20download&f=false>. Acessado em: 10/09/2013.

GIL, R. L. **Tipos de Pesquisa**, 2009. Disponível em: <http://wp.ufpel.edu.br/ecb/files/2009/09/Tipos-de-Pesquisa.pdf>. Acessado em: 21/10/2013.

GINIGE, A.; MURUGESAN, S. **Web Engineering: An Introduction**. University of Western Sydney. Australia, 2001a. Disponível em: <http://www-itec.uni-klu.ac.at/~harald/proseminar/web1.pdf>. Acessado em: 21/03/2013.

GONÇALVES, H. N. **Geração de testes automatizados utilizando o Selenium.** UFPR .2011. Disponível em < tcc.ecomp.poli.br/20111/ >. Acessado em: 21/10/2012.

GUERRA, A. C.; COLOMBO R. M. T. **Tecnologia da Informação - Qualidade do Produto de Software**. Ministério das Ciências e Tecnologia - Secretaria de Politica de Informática, 2009.

HIRAMA, K. **Engenharia de Software – Qualidade e Produtividade com Tecnologia**. Rio de Janeiro, Elservier, 2012. Disponível em: <http://books.google.com.br/books?id=tYhQYH2yiiYC&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false>. Acessado em: 10/01/2013.

ITESTE. **Uso do SIKULI para automação**, 2011. Disponível em: <http://www.iteste.com.br/LinkClick.aspx?fileticket=U5Ihjq-6Rxk%3d&tabid=320&mid=1205>. Acessado em: 03/06/2013.

JUNIT. **Annotation Type After**. Disponível em: <http://junit.sourceforge.net/javadoc/org/junit/After.html>. Acessado em: 08/10/2013.

JUNIT. **Annotation Type Before**. Disponível em: <http://junit.sourceforge.net/javadoc/org/junit/Before.html>. Acessado em: 08/10/2013.

JUNIT. **Annotation Type Test**. Disponível em: <http://junit.sourceforge.net/javadoc/org/junit/Test.html> Acessado em: 08/10/2013.

LUCENA, F. N. **JUnitAnotations**. Disponível em: <http://code.google.com/p/exemplos/wiki/JUnitAnotations>. Acessado em: 08/10/2013.

MACHADO, A. M. **A implantação de testes automatizados de aplicações Web num ambiente hostil**, 2009. Disponível em**:**

<http://www.inf.ufsc.br/~alexmag/artigo_estudo_de_caso.pdf> Acessado em: 06/03/2013.

MANINI, E. P.; LACERDA, G. S**. Héstia – Ferramenta de Apoio a Teste de Software com Base em Casos de Uso**, 2009. Disponível em: http://www.uniritter.edu.br/graduacao/informatica/sistemas/downloads/tcc2k9/TCCII_EduardoManini_2009_2.pdf. Acessado em: 10/06/2013

MARINS, A. F. **Utilizando a Selenium IDE,** 2009. Disponível em: <http://imasters.com.br/artigo/13317/>. Acessado em: 05/03/2013

MYERS, G. **The Art of Software Testing**. 2nd ed. John Wiley & Sons, 2004. Disponível em: <http://www.noqualityinside.com/nqi/nqifiles/The%20Art%20of%20*Software*%20Testing%20-%20Second%20Edition.pdf>. Acessado em: 14/03/2013.

NAZARÉ, E. A. O. **SeleniumTG: Uma Ferramenta web para Geração de Scripts de Teste**, 2012. Disponível em: <http://tcc.ecomp.poli.br/20121/TCC%20-%20EAON.pdf>. Acessado em: 23/03/2013.

NETO, A.; CALAÇA, O. Desenvolvendo em três camadas com PHP, MVC e AJAX. **PHP Magazine**, 5ª ed. p. 17-22, agosto 2008. Disponível em: <http://www.phpmagazine.org.br/portal/?modulo=secao&id=1>. Acessado em: 11/10/2013.

NETO, P. A. S. **Introdução à Engenharia de *Software***, 2010. Disponível em: <http://www.ufpi.br/subsiteFiles/pasn/arquivos/files/IntroducaoEngenhariaDeSoftware.pdf>. Acessado em: 21/03/2013.

NSTI. **NSTI - Núcleo Setorial de Tecnologia de Informação**. Disponível em: <http://www.nsti.com.br/index.cfm?apresentacao=1>. Acessado em: 29/04/2013.

OLIVEIRA, R. B. **Framework FUNCTEST: Aplicando Padrões de *Software* na Automação de Testes Funcionais.** Universidade de Fortaleza, 2007. Disponível em: < https://uol02.unifor.br/oul/conteudosite/F1066342331/Dissertacao.pdf>. Acessado em: 20/12/2012.

OLIVEIRA, R. B; GÓIS, F. N. B; FARIAS, P. P. M. **Automação de Testes Funcionais: Uma Experiência do SERPRO**. Universidade de Fortaleza, 2007. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sast/2007/011.pdf>. Acessado em: 14/03/2013.

PRESSMAN, R. S. **Engenharia de Software: Uma Abordagem Profissional**. 7 ed. São Paulo: Pearson, 2010.

SANTIAGO, M. C. **Estudo de Caso sobre Testes Automatizados com Ênfase na Ferramenta Selenium.** Trabalho apresentado para obtenção do título de Especialização em Desenvolvimento de Sistemas Corporativos, FARN, Natal, 2013.

SANTOS, I. S.; NETO, P. A. S. **Automação de testes funcionais com o Selenium**, 2009. Disponível em:

<http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/smc2.pdf>. Acessado em: 16/03/2013.

SILVA, T. X. L. **Geração Automática de Scripts de Teste Utilizando o Selenium**, 2011. Disponível em: <http://tcc.ecomp.poli.br/> Acessado em: 23/03/2013.

SIXPENCE, E.; ADÃO P.; SMITH C. **Automatização de Casos de Teste como Processo de Melhoria da Qualidade do Software: O Caso da Aplicação E-Learning ISUPAC3 no ISUTC**. In 6º Congresso Luso-Moçambicano de Engenharia (CLME), Maputo, Moçambique, 2011. Disponível em: <http://www.math.ist.utl.pt/~padao/publications/11-SAS-CLME/11-SAS-ISUPAC3.pdf>. Acessado em: 20/12/2012.

SOMMERVILLE, I. **Engenharia de Software**. 9 ed. Pearson, 2011.

SOUZA, L. V. S. S. **Geração automática de casos de teste a partir de casos de Uso: um mapeamento Sistemático da literatura**, 2011. Disponível em: <http://tcc.ecomp.poli.br/20111/TCC-Lamartine_VersaoFinal.pdf> Acessado em: 06/03/2013.

SOUZA, S. R. S *et al*. **Introdução ao Teste de Software**, 2003. Disponível em: <http://www.labes.icmc.usp.br/site/sites/default/files/NotaDidatica65.pdf>. Acessado em: 21/12/2012.

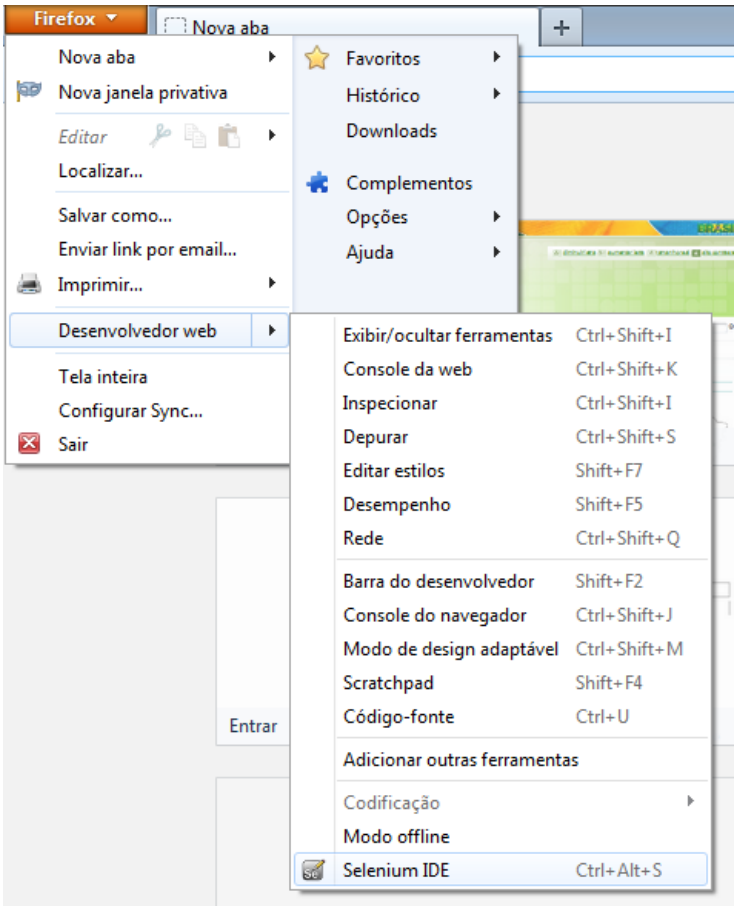
SWEBOK. **Guide to the Software Engineering Body of Knowledge**. Disponível em: <http://webstore.computer.org/Guide-Software-Engineering-Knowledge-version/dp/B009AJGEZW>. Acessado em: 24/08/2012.

APÊNDICE A – PASSO A PASSO DE UM TESTE COM SELENIUM IDE

PASSO 1: Após a instalação do Selenium IDE no navegador Mozilla Firefox, abra o navegador.

PASSO 2: Acesse a IDE, que pode ser acessada através do menu Desenvolvedor web como mostra a Figura 1.

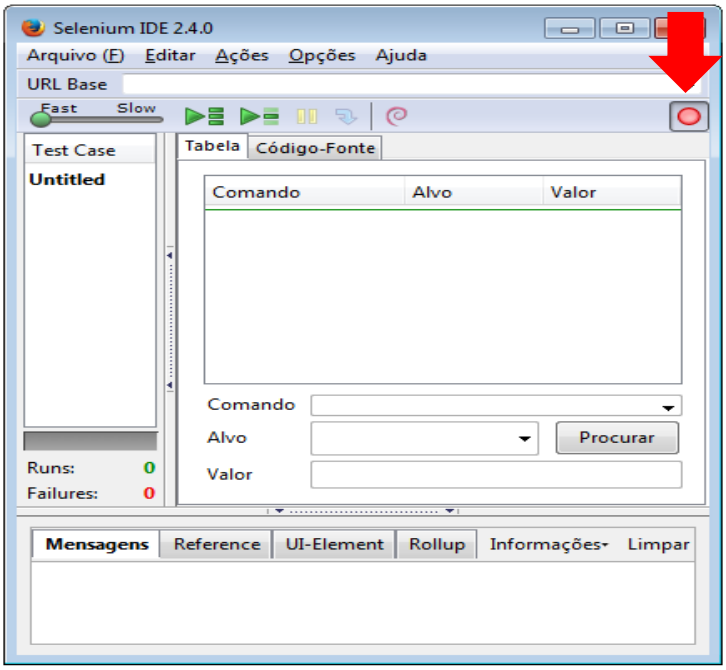
Figura 1 – Acesso ao Selenium IDE



Fonte: Elaborado pelas autoras

Passo 3: Ao ser exibida a interface do Selenium IDE verifique se o botão Record (indicado pela seta vermelha) está ativado como mostra a Figura 2.

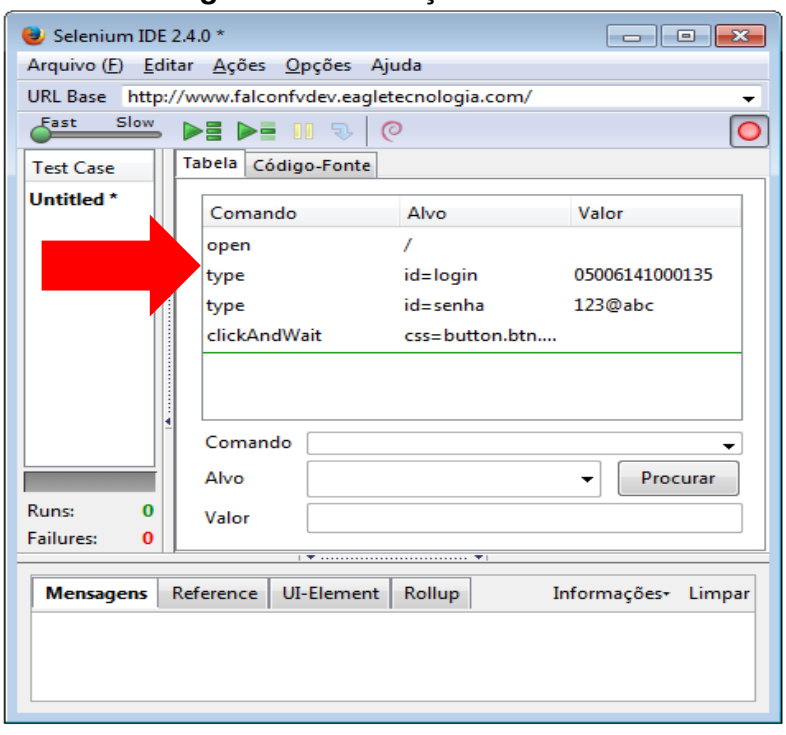
Figura 2 – Interface do Selenium



Fonte: Elaborado pelas autoras

PASSO 4: Comece a interagir com a página de acordo com o caso de teste a ser executado, que o Selenium IDE gravará as ações realizadas que formarão a estrutura do teste em HTML, como mostra a Figura 3.

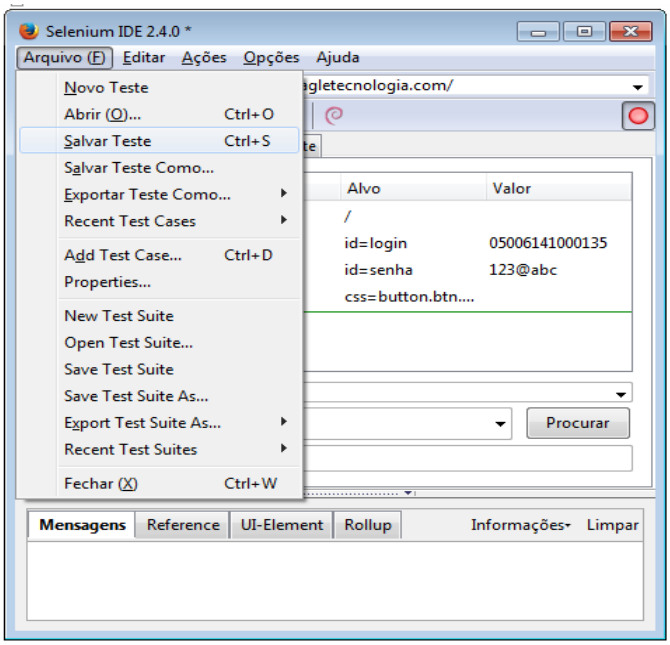
Figura 3 – Gravação do Teste



Fonte: Elaborado pelas autoras

PASSO 5: Salve o teste, através do menu arquivo, selecionando salvar teste e salve em formato HTML.

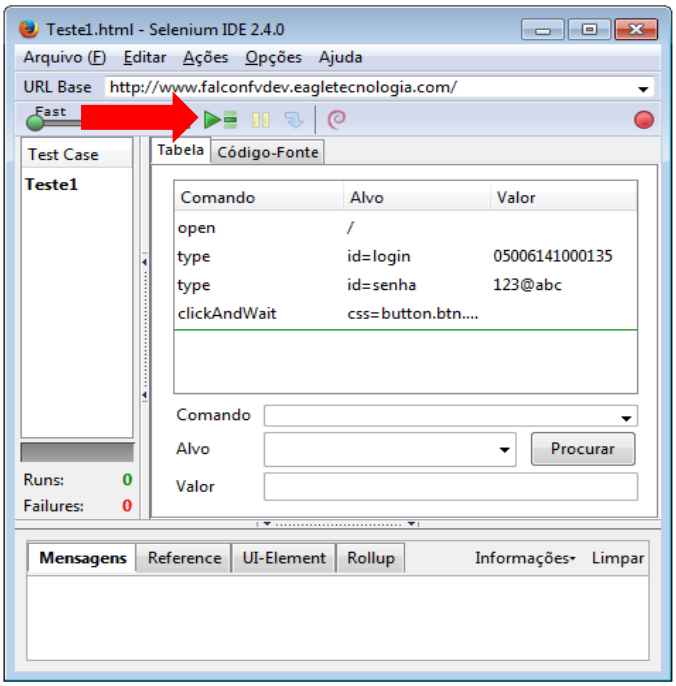
Figura 4 – Gravação do Teste



Fonte: Elaborado pelas autoras

PASSO 6: Desative o botão Record, clicando sobre ele. O teste salvo pode ser executado clicando no botão Run.

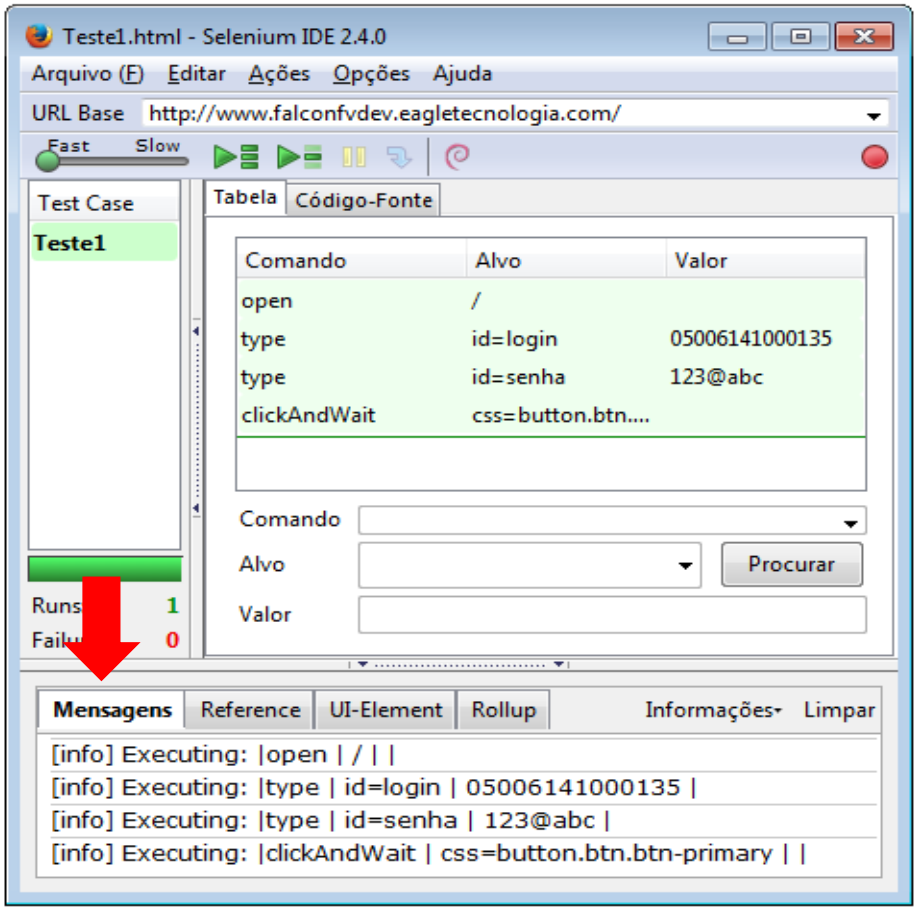
Figura 5 – Executar do Teste



Fonte: Elaborado pelas autoras

Ao clicar no botão Run, os comandos que aparecem na Figura 5, serão executados e o Selenium fornecerá o que aconteceu ao executar cada um deles na guia Mensagens, como mostra a Figura 6.

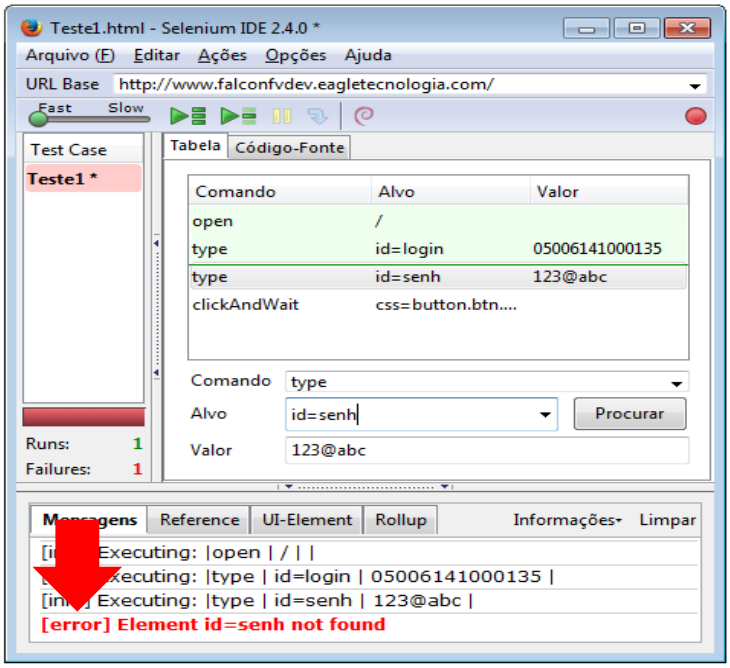
Figura 6 – Execução do Teste



Fonte: Elaborado pelas autoras

Caso ocorra algum erro também será reportado pela ferramenta na guia mensagem como mostra a Figura 7.

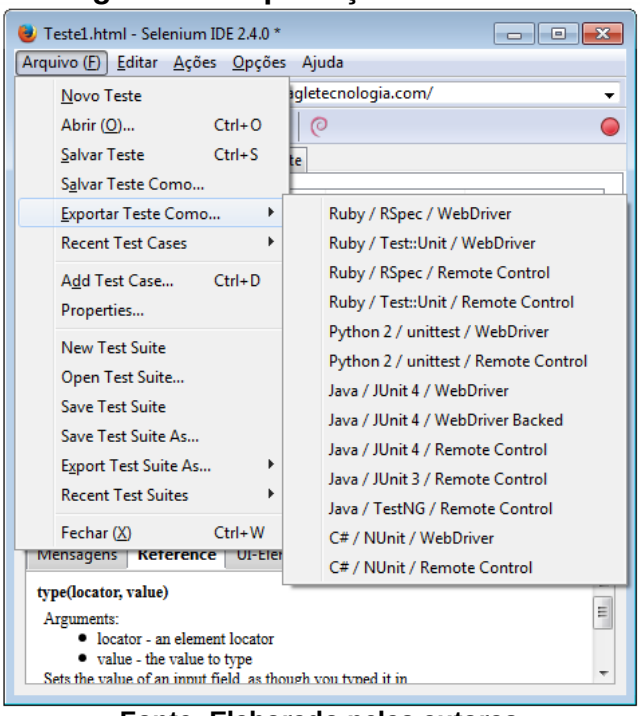
Figura 7 – Reporte de Erro



Fonte: Elaborado pelas autoras

Também é possível exportar os testes para uma das linguagens de programação que a ferramenta oferece suporte através do menu Arquivo, selecionando Exportar teste como, como mostra a Figura 8.

Figura 8 – Exportação dos Testes



Fonte: Elaborado pelas autoras