UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Arthur Moreira Alves

Comparativo de ferramentas de automatização de testes: Selenium IDE e Selenium WebDriver

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Arthur Moreira Alves

Comparativo de ferramentas de automatização de testes: Selenium IDE e Selenium WebDriver

Trabalho de conclusão de curso apresentado à Faculdade de Gestão e Negócios da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Gestão da Informação.

Orientador: José Eduardo Ferreira Lopes

Coorientador: Régis Michel dos Santos Sousa

Universidade Federal de Uberlândia – UFU

Faculdade de Gestão e Negócios

Bacharelado em Gestão da Informação

Uberlândia, Brasil

2018

Agradecimentos

Resumo

O propósito deste trabalho é auxiliar no processo de tomada de decisão na hora da seleção de uma ferramenta de testes automatizados através da análise de parâmetros pré-definidos. Os critérios de análise serão evidenciados e destacados para salientar os pontos positivos e negativos das ferramentas Selenium IDE e Selenium WebDriver. Os dados serão coletados através de testes realizados em uma aplicação Web, tabelados e classificados por meio de uma análise descritiva.

Palavras-chave: Automação; Testes de Sistema; Testes Automatizados; Testes Aplicação Web; Selenium WebDriver; Selenium IDE.

Lista de ilustrações

Sumário

1 Introdução

O termo automação de testes primeiramente deve ser esclarecido, pois existem diversos tipos de tecnologia utilizadas e etapas que podem se beneficiar desse processo. Categorias de automação de testes incluem: Gerência de testes, testes unitários, testes de geração de massa de dados, testes de performance e testes funcionais/de sistema/regressão. Geralmente esses se beneficiam de tecnologias de automação em "execução de testes" (também conhecida como Capturar/Reproduzir ou Gravar/Reproduzir).

Trabalharemos no escopo deste trabalho os testes de sistema, sendo considerado o cenário principal de testes e um processo de grande importância na aferição de qualidade. (WISSINK; AMARO, 2006). Como a automatização de testes está se expandindo, muitas vezes o processo de averiguação de qualidade é feito manualmente, onde um usuário, seguindo um plano de testes efetua as conferências necessárias, segue os passos a serem repetidos e registram os resultados obtidos.

Atualmente a empresa se utiliza de um recurso humano para efetuar os testes, o que implica a necessidade de um colaborador dedicado, planejando, efetuando e validando os resultados todas as vezes que houver essa necessidade, com o uso de um processo automatizado essas tarefas seriam necessarias apenas uma vez e quando surgir novamente a necessidade, os testes automatizados podem ser executados para gerar as validações, além de que o uso de recurso humano para este tipo de trabalho abre margens para erros, o que poderia ser evitado utilizando o processo automatizado. (KUMAR; MISHRA, 2016)

A automatização de teste podem ser aplicadas em aplicações Desktop, que são aquelas executadas diretamente pelo sistema operacional ou em aplicações Web, que são interpretadas através de um navegador. Tendo em vista que a aplicação avaliada é uma aplicação web, os impactos por erros geralmente são bastantes visíveis e dependem de uma rápida resposta de ajuste, já que essa aplicação não pode ficar indisponível para seus utilizadores. (SARAVANAN; PRASAD, 2016)

Este trabalho traz como objetivo a comparação entre duas ferramentas responsáveis pela automatização de testes de sistema, auxiliando no processo de tomada de decisão, permitindo uma maior assertividade e acurácia dos responsáveis pela escolha da ferramenta que melhor atende as necessidades de sua organização.

2 Objetivos

2.1 Objetivo Geral

Objetiva-se com este trabalho, apresentar os resultados comparativos de duas ferramentas de testes automatizados de sistema.

2.2 Objetivo Específico

- Selecionar duas ferramentas utilizadas para testes automáticos de sistema
- Apontar as características de cada uma das ferramentas
- Apresentar a síntese dos resultados comparativos entre as duas ferramentas.

2.3 Justificativa

A proposta que torna viável a apresentação desse trabalho é a necessidade da automatização de testes para substituir um recurso humano nos testes de aplicações web. Sendo necessário a escolha de um software para efetuar essa automatização.

Como o mercado oferece uma grande gama de ferramentas para efetuar automatizações de teste, este trabalho se justifica conforme há a necessidade de escolha de qual dessas ferramentas adotar. Assim, este trabalho poderá contribuir no processo de tomada de decisão, ao apontar pontos fortes e vulnerabilidades em cada uma das soluções testadas.

3 Referencial Teórico

Neste capítulo são apresentados e discutidos conceitos importantes para o entendimento deste trabalho. Na Seção 3.1 é elucidado os conceitos dos testes de Softwares. Já na Seção 3.2, explica-se mais a fundo as técnicas de testes de softwares. Na Seção 3.3 é exposto o conceito dos tipos de testes mais utilizados atualmente e dentro da Seção 3.4 é exposto a diferença de testes em aplicações webs comparado com as aplicações em dispositivos.

3.1 Testes de Software

Os testes de software têm como função identificar, registrar e documentar erros estruturais ou de código em aplicações a partir de roteiros de testes e cenários que possam causar falhas, garantindo assim uma maior qualidade na entrega do produto final.

As empresas buscam identificar os erros e defeitos em códigos, que afetem a funcionalidade da ferramenta antes da entrega final, pois, além de passar uma imagem negativa para o cliente, essas falhas podem causar prejuízos ou não atender alguma funcionalidade que o software deveria possuir. (DELAMARO; JINO; MALDONADO, 2013)

Outro ponto que deve ser levado em consideração é a identificação dos erros antes da avaliação da equipe de qualidade, já que a correção de erros por parte do desenvolvedor é contabilizada como tempo de codificação e pode afetar os prazos de entrega. (ANDRADE; VIEIRA, 2009)

Atualmente o mercado possui diferentes opções de técnicas de testes, geralmente com objetivos e execuções diferentes, os padrões mais comuns para aferição de qualidade são como CMMI (TEAM, 2006) ou MPS.BR (BR, 2011), há também os métodos ágeis, como SCRUM (SCHWABER; BEEDLE, 2002).

3.2 Técnicas de Testes de Software

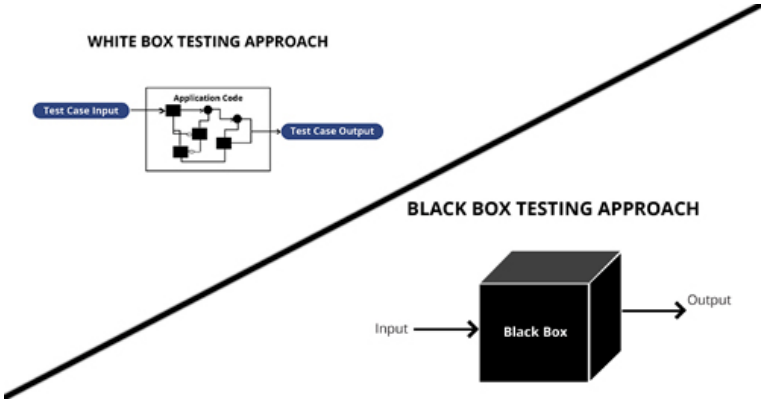
As técnicas de testes possuem diferentes abordagens, sendo classificadas superficialmente como testes estáticos e dinâmicos. Os testes estáticos podem ser feitos durante o desenvolvimento da aplicação e envolve uma parte maior de análise, como *peer-review*, guia de uso e inspeções. Já os testes dinâmicos são executados durante o funcionamento da aplicação, tanto na aplicação completa ou em seus módulos individualmente. Comumente vemos duas técnicas distintas sendo empregadas, a técnica dos testes de caixa e os testes visuais (KOSCIANSKI; SOARES, 2007). Os testes de caixa se subdividem em três

categorias:

- Testes de caixa-branca: Têm como objetivo encontrar problemas no fluxo de dados da aplicação através de códigos de baixa qualidade. Faz uma análise interna no código, considerando cada entrada e qual sua saída, comparando a mesma com a saída desejada. Sua principal função é encontrar problemas na estrutura interna do software por meio de seu fluxo de dados.

- Testes de caixa-preta: Esse tipo de teste desconsidera completamente o código, linguagem e estrutura da aplicação, diferente do teste de caixa-branca. Têm como objetivo capturar erros em entradas e saídas de dados, seu roteiro de teste descreve ao testado apenas o que deve ser esperado do sistema em tempo de execução.

Figura 1 – Estrutura comparativa de Teste de Caixa-Branca e Texte de Caixa-Preta.



Fonte: https://www.invensis.net/

- Testes de caixa-cinza: Considera-se que o testador possui entendimento do código interno da aplicação e dessa forma desenvolve os casos de teste. No entanto sua execução é feita semelhante ao teste de caixa-preta, desencorajando o testador à manipulação de dados.

- Testes visuais: Almeja demonstrar a alguém o problema de um software exibindo a informação ao invés de apenas descreve-la. Portando, essa técnica consiste em gravar o processo que causa o erro.

Os testes visuais possuem a vantagem de integrar melhor a equipe de qualidade com a equipe de desenvolvimento, já que apresentar o problema dessa forma permite que o desenvolvedor entenda o comportamento do sistema, agindo diretamente na causa do problema e não em como ele ocorre.

3.3 Tipos de Testes de Software

Os tipos de testes são as formas de se testar uma aplicação, esses testes se diferem devido as técnicas empregadas para sua execução e não em seu ambiente. Nos atentaremos aqui aos quatro principais tipos de testes empregados atualmente. (FREEMAN, 2002) São esses:

- Testes de Unidade/Testes Unitários: Metodologia exaustivamente utilizada na base da pirâmide citada por COHN (COHN, 2009). Esse tipo de teste visa isolar uma pequena parcela específica do código, onde verifica sua funcionalidade. É a menor e a primeira possibilidade de verificação e teste. Essa etapa é frequentemente executada pelos próprios desenvolvedores do software, não havendo necessidade de contato com os testadores convencionais.

- Teste de Integração: Possui foco em verificar a funcionalidade dos módulos individualmente assim como integrados com outros módulos da aplicação, em grupos. Geralmente dependem da conclusão do desenvolvimento da aplicação para sua execução. É conseguinte do teste unitário porém possui um escopo maior, é antecessor dos testes de sistema.

- Testes de Sistema: Tipo de teste cujo objetivo é uma verificação total das funcionalidades e integrações do sistema baseado em requisitos previamente levantados. Testes de sistema geralmente são executados em ambiente de caixa-preta e dependem da conclusão do teste de integração para sua execução.

  Possui um escopo bastante limitado pois, além de não possui conhecimento da lógica implementada para o desenvolvimento ainda precisa buscar defeitos no sistema como um todo.

- Testes de Aceitação: São testes conduzidos também em ambiente caixa-preta e são geralmente dirigidos através de roteiros que descrevem os requisitos exatos da aplicação. Diversos casos são utilizados para simular e dessa forma garantir que as requisições levantadas em contrato sejam cumpridas.

  Os testes de aceitação não se preocupam com as funcionalidades do software em si, que já devem ser previamente verificadas. Sua principal funcionalidade é a garantia da entrega da aplicação dentro das requisições acordadas com o cliente. Frequentemente são verificações visuais ou de especificações pontuais, como tamanho de arquivos para envio ou recebimento.

3.4 Testes de Software em Aplicações Web

Assim como qualquer aplicação dedicada exclusivamente a dispositivos, os softwares desenvolvidos para ambiente web também precisam ser verificados por meio de testes. Porém em grande parte dessas aplicações, as teorias e métodos de testes tradicionais precisam ser adaptados como testes em diferentes Sistemas Operacionais, navegadores, tamanhos de monitores, pois essas aplicações são bastante peculiares e de maior complexidade(LUCCA; FASOLINO, 2006).

Existem maneiras de prevenir erros durante sua construção através de boas práticas, processos bem definidos e atenção na codificação auxiliam no desenvolvimento porém não impedem o surgimento de problemas, como sobrecarga ou falta de visão externa dos casos.

A descoberta tardia de um defeito na aplicação pode causar prejuízos ao usuário e um alto custo de correção, além de causar um desperdício de tempo (ANDRADE; VIEIRA, 2009), caso os processos utilizados forem manuais, já que essa é uma forma lenta de atuação(WANG; DU, 2012).

Aplicações Web podem ser consideradas sistemas distribuídos, utilizando uma arquitetura que se assemelha a uma arquitetura cliente-servidor ou de múltiplas camadas(LUCCA; FASOLINO, 2006). Deve-se considerar também o ambiente em que se encontra a aplicação online, tendo em vista que o usuário pode utilizar diversos navegadores diferentes, assim como também pode-se utilizar diferentes sistemas operacionais.

Alguns tipos de testes necessitam de um estudo complementar para serem executados para aplicações web, por exemplo, os testes de usabilidade precisam verdadeiramente de um estudo complementar que analise o comportamento do usuário utilizando a aplicação, aumentando em muito o custo de aplicação desse teste por conta do esforço e tempo despendido para a utilização do mesmo. Conforme se aproxima e se entra em um processo de entrega, novas versões surgem rapidamente e se faz necessário um modelo de testes que siga o mesmo padrão. (LUCCA; FASOLINO, 2006)

Os testes web mais comumente aplicados são:

- Testes de Sistema: Funcionalidades básicas são verificadas. Seu objetivo é encontrar defeitos e erros que afetem toda a aplicação.
- Testes de acessibilidade: São bastante utilizados no escopo web, e tem como objetivo garantir que todos os usuários consigam visualizar a página de forma satisfatória.
- Testes de carga: Utilizados para verificar a limitação de tráfego que o sistema consegue suportar, permitindo prever a capacidade máxima de usuários utilizando o sistema simultaneamente.

Outras formas de testes também são utilizadas durante as validações necessárias, porém as supracitadas se tornam as principais pelo seu impacto na qualidade.

4 Métodos

Este projeto segue a abordagem qualitativa conforme Gil (2008), que procura melhorar a compreensão do fenômeno, entendê-lo em profundidade, através de informações de grupo de pessoas acerca do problema estudado, para obter conclusões correspondentes aos dados coletados, após análise qualitativa.

Para classificar uma pesquisa quanto a seus objetivos (GIL, 2008), apresenta que podem ser exploratórias, descritivas e explicativas, sendo que esse estudo é pesquisa exploratória, porque desenvolve, esclarece e modifica conceitos e ideias, para apresentar uma visão geral sobre o tema, considerando pouco explorado, além de observar sua aplicação prática. Pode-se considerar, também, como descritiva, uma vez que busca descrever características do tema, criar uma nova visão do problema, estabelecendo relações entre variáveis, observando a atuação prática.

- Análise Descritiva Qualitativa: Ao final deste trabalho, os dados levantados serão organizados, relacionados e será feita uma análise qualitativa dos resultados obtidos.
- Coleta de Dados: Os dados para comparativo das ferramentas serão obtidos através da execução de testes automatizados por meio das duas ferramentas em um mesmo cenário de teste.
- Procedimento de Análise e Interpretação: Os dados coletados serão comparados, analisados e avaliados de acordo com o usuário que efetuou a coleta.
- Conclusão: Com base nas análises feitas será capaz de concluir qual ferramenta, em âmbito geral, é melhor ou em que cada uma se destaca de acordo com suas funcionalidades.

Para mensurar as características de cada ferramenta a fim de fazer o comparativo, os parâmetros foram definidos e acompanhados durante a utilização dos mesmos para que ambas fossem comparadas em paralelo. As características que serão analisadas em cada um deles foram:

- Usabilidade: Dado baseado na experiência de utilização, através de um usuário inexperiente com qualquer tipo de ferramentas desse tipo.
- Performance: Baseado em tempo de execução de cada um dos testes analisados.
- Customização: Parâmetro levantado através da capacidade de executar diversas tarefas não pré-programadas pelas aplicações.

- Abrangência: Capacidade da ferramenta de abranger mais cenários de teste sem necessidade de utilização de ferramentas externas.

Houve uma participação ativa do autor desse trabalho nos comparativos referentes às ferramentas supracitadas, por serem parâmetros subjetivos podem ser interpretados de formas diferentes com base nas experiências do leitor.

4.1 Ferramentas para automação de Testes Web

Uma ferramenta comumente utilizada na automação de testes de software web é o Selenium IDE[1], que é um complemento anexado ao navegador Mozilla Firefox, que possibilita a automação seguindo um padrão de Capturar e Reproduzir

Temos também o Selenium WebDriver[2], um framework que através de codificação tem como objetivo a automatização de testes. (HOLMES; KELLOGG, 2006)

Ambas são ferramentas bastante confiáveis, oferecem suporte ao usuário através de seus sites, com documentaçaõ acessível a todos. São simples de manusear e são ferramentas *open source*, o que significa que a comunidade desenvolve constantemente novas funcionalidade, levando assim a escoha dessas ferramentas para desenvolvimento do trabalho

[1] Selenium IDE. Disponível em: <http://seleniumhq.org/docs/02$_s$*elenium*$_i$*de.html* >
[2] Selenium WebDriver. Disponível em <https://www.seleniumhq.org/projects/webdriver/>

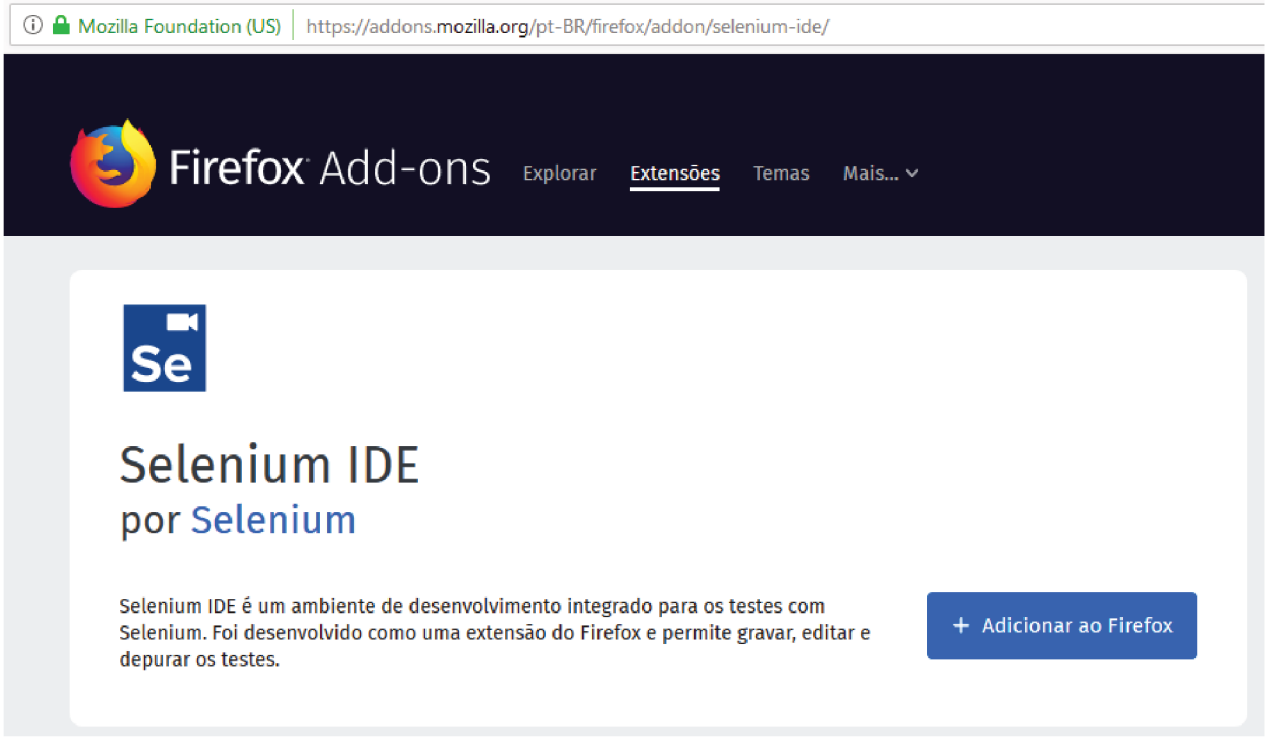
5 Desenvolvimento

Nesta parte do trabalho, como houve a necessidade de desenvolvimento dos testes para a coleta de dados, será disponibilizado nesta seção um "tutorial"de instalação, configuração e manuseio tanto para a ferramenta Selenium IDE quanto para a Selenium WebDriver.

5.1 Utilizando o Selenium IDE

Para desenvolver os testes para a aplicação web através do plugin Selenium IDE é necessário o download do navegador Mozilla Firefox[1]. Após o download do mesmo é necessário acessar a página de complementos para o navegador, que disponibiliza a extensão Selenium IDE[2]. Ao clicar em "Adicionar ao Firefox", uma janela aparecerá solicitando as permissões de utilização da ferramenta.

Figura 2 – Extensão Selenium IDE para Mozilla Firefox. Acessado em 17/09/2018



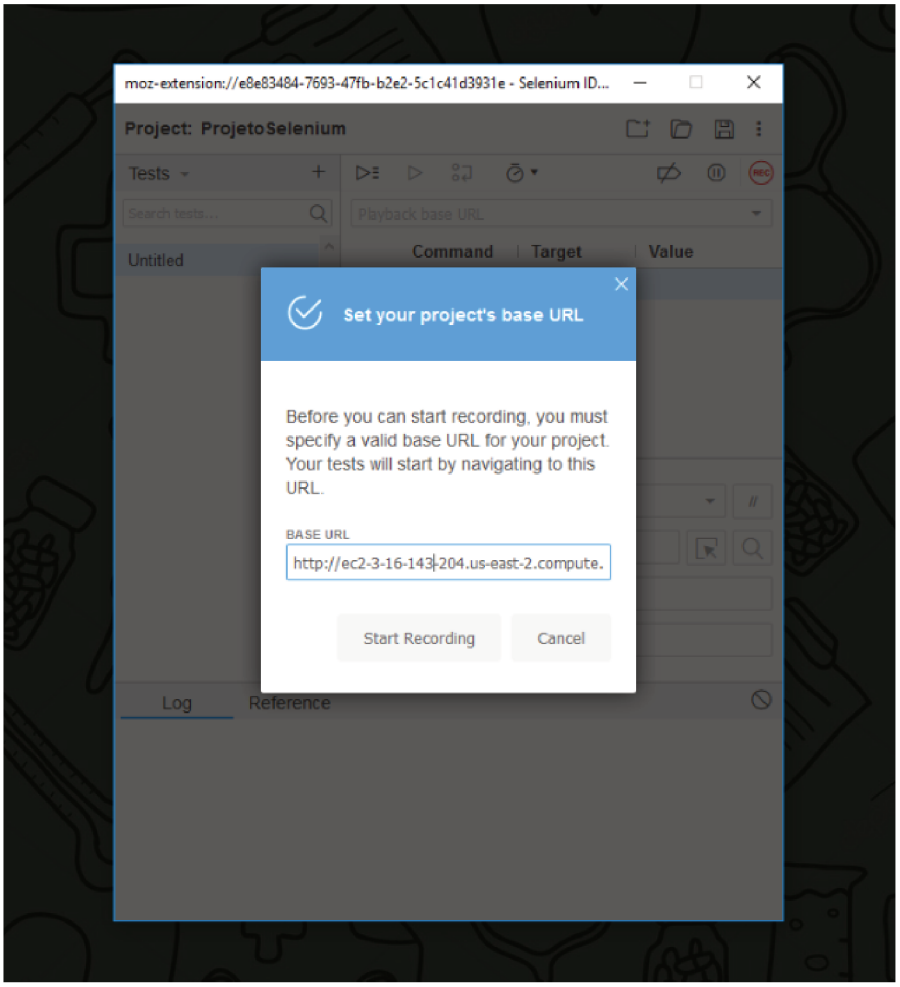
Fonte: Elaborada pelo autor

Ao clicar no ícone da extensão no canto superior direito, aparecerá uma tela com as opções para criar um novo projeto, carregar um projeto existente ou fechar a extensão.

[1] Mozilla Firefox. Disponível em: <https://www.mozilla.org/pt-BR/firefox/new/>
[2] Selenium IDE. Disponível em: <https://addons.mozilla.org/pt-BR/firefox/addon/selenium-ide/>

Selecionando a opção de criar um novo projeto, será solicitado o nome do novo projeto e a URL base da aplicação web que será testada.

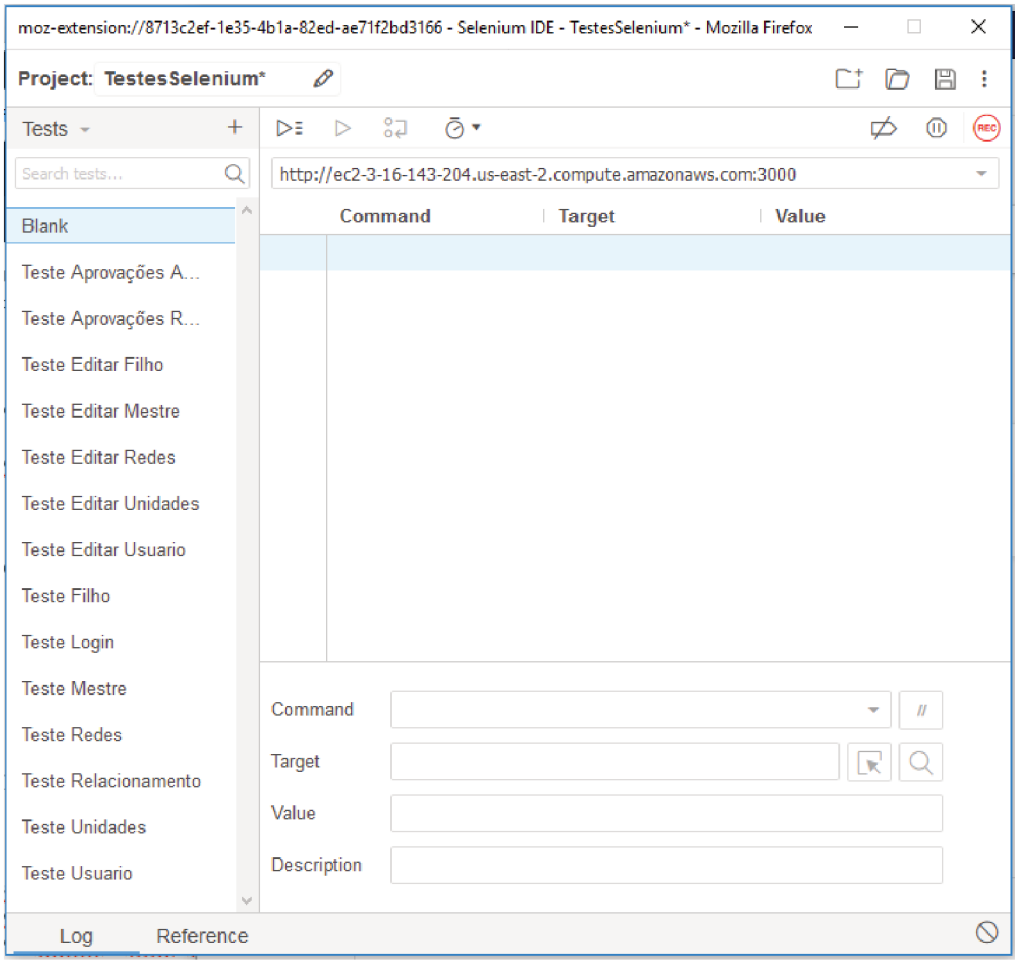
Figura 3 – Informando URL base da aplicação.



Fonte: Elaborada pelo autor

A URL informada será aberta e o plugin começará a gravar os movimentos feitos pelo cursor e as entradas de informações providas via teclado. A gravação pode ser pausada a qualquer momento abrindo a janela do plugin e clicando no botão de "pause"no canto superior direito, assim como uma nova gravação pode ser iniciada ou continuada.

Figura 4 – Menu principal para desenvolvimento de caso de testes.



Fonte: Elaborada pelo autor

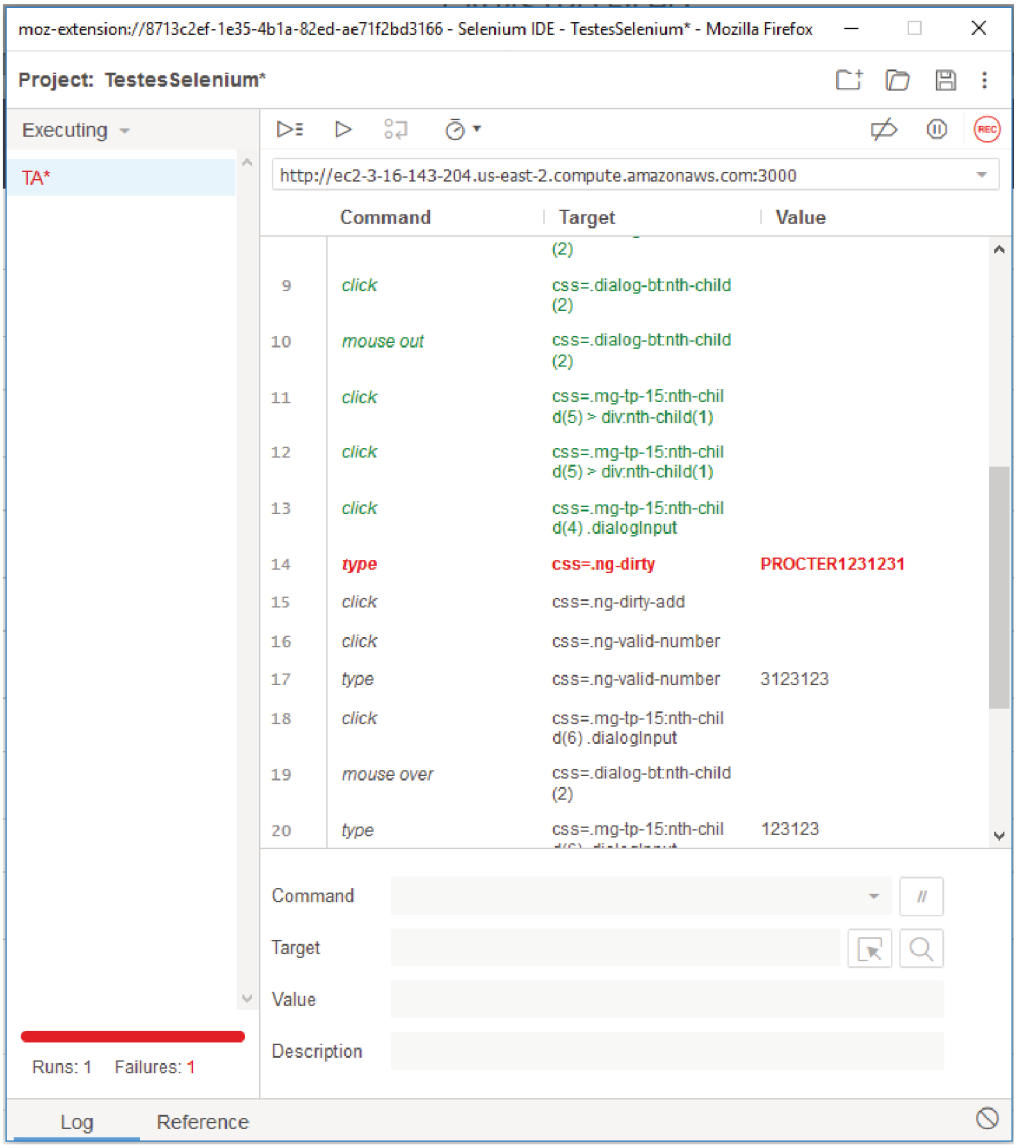
Para aplicações com várias funcionalidades e/ou telas, é possível a criação de diversos testes em um mesmo projeto, clicando no ícone de positivo no canto superior esquerdo.

Em algumas ocasiões pode ser necessário a inserção de alguns comandos manualmente, o que também é permitido pela aplicação através do campo "*Command*", nele é inserido o comando desejado como clicar, submeter, repetir uma ação e até mesmo digitar algo específico. Na opção "*Target*"é informado ao programa em qual elemento da tela será executado o comando supracitado. Essa informação deve ser passada por um identificar único do elemento, como um "id", uma classe ou até mesmo seu "xpath". O campo "*Value*"é utilizado para informar a IDE o valor a ser inserido no elemento mapeado. Esse campo geralmente é utilizado para simular a inserção de caracteres via teclado, como preencher um nome ou um endereço.

Ao gerar um caso de teste, o mesmo é executado clicando no botão "*Run Current Test*"localizado na parte superior da IDE, ou pressionando Ctrl+R. Ao ser executado, o plugin abrirá uma nova janela do navegador e iniciará sua rotina. Cada ação feita

corretamente implicará cor verde no comando executado, caso um erro ocorra, a execução do caso de teste é paralizada e o local onde ocorreu o incidente é destacado na cor vermelha.

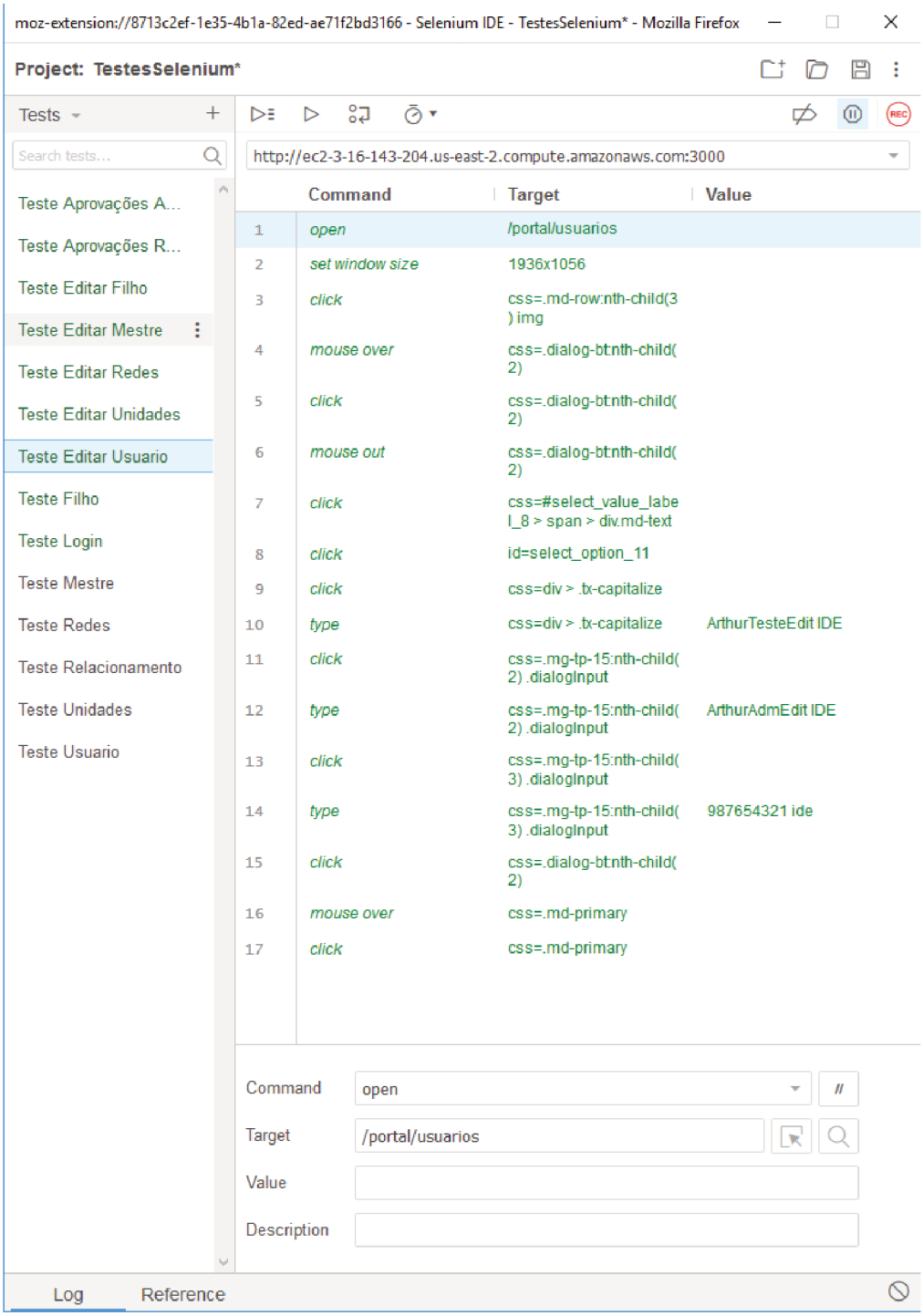
Figura 5 – Execução paralizada após encontrar um erro.



Fonte: Elaborada pelo autor

Caso o teste ocorra com sucesso, todos os comandos executados ficarão verdes e a janela do Selenium IDE será trazida para frente, sinalizando o final da execução do caso de teste.

Figura 6 – Execução finalizada com sucesso.



Fonte: Elaborada pelo autor

Para o caso específico do desenvolvimento deste trabalho foi utilizado uma aplicação de gerenciamento de categorias. Uma aplicação Web que permite cadastro, alteração e manutenção de dados relacionados a redes de farmácia, medicamentos, unidades, relacionamentos, etc. Através do Selenium IDE foram testadas as seguintes funcionalidades: Tela de Login, Cadastro de novos usuários, Criação de novas redes, Cadastro de unidades, Cadastro de medicamento mestre, Cadastro de medicamento filho, Criação de relacio-

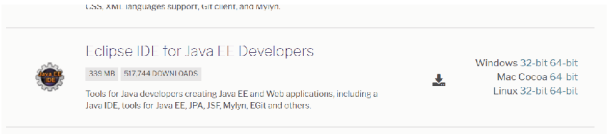
namento, Resolução de incossistência, Aprovação de atividades, Recusa de atividades, Edição de usuários, Edição de redes, Edição de unidades, Edição de medicamento mestre e edição de medicamento filho.

5.2 Utilizando o Selenium WebDriver

Para a utilização do Selenium WebDriver é necessário a utilização de uma linguagem de programação entre: Java, Python, C#, Ruby ou Javascript. Para o desenvolvimento dos testes contidos nesse trabalho, utilizou-se a linguagem Java, portanto a utilização será descrita para o desenvolvimento com a mesma.

Para iniciar o desenvolvimento utilizando a linguagem Java, é necessário o download de uma IDE para a codificação. No caso desse trabalho a IDE utilizada foi o Eclipse. Para descarregar o mesmo é necessário acessar o site do Eclipse e efetuar o download da IDE "Eclipse IDE for Java EE Developers"[3], disponibilizada gratuitamente.

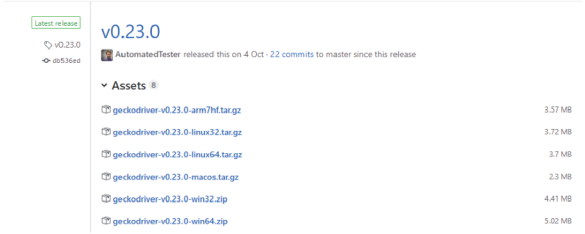
Figura 7 – Tela para download do Eclipse IDE. Acessado em 10/10/2018



Fonte: Elaborada pelo autor

Como os testes utilizando o Selenium IDE foram feitos no navegador Mozilla Firefox, os testes efetuados através do Selenium WebDriver aqui descritos utilizarão o mesmo navegador, evitando assim divergência de desempenho. Para acessar o Mozilla através da IDE é necessário o download de um "driver", que pode ser obtido através do github dos desenvolvedores[4]. O driver deve ser baixado e extraído em uma pasta qualquer da máquina que será feito o desenvolvimento dos testes. O mesmo tem como objetivo abrir o navegador escolhido para a execução dos casos de teste.

Figura 8 – Github para download do Gecko Driver. Acessado em 10/10/2018



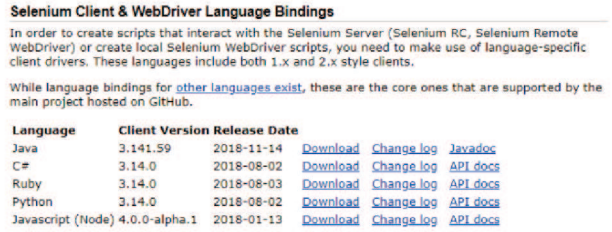
Fonte: Elaborada pelo autor

[3] Eclipse IDE for Java EE Developers. Disponível em: <https://www.eclipse.org/downloads/packages/release/kepler/ide-java-ee-developers>

[4] Gecko WebDriver. Disponível em: <https://github.com/mozilla/geckodriver/releases>

Deve-se acessar o site Selenium HQ[5] e efetuar o download da biblioteca para automatização de testes em Java.

Figura 9 – Tela para download da biblioteca Java. Acessado 15/11/2018



Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

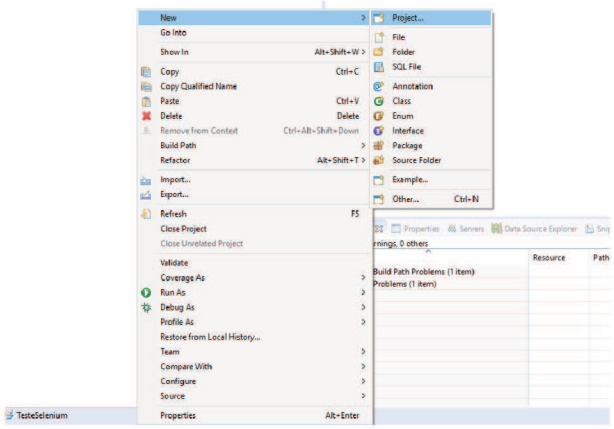
While language bindings for other languages exist, these are the core ones that are supported by the main project hosted on GitHub.

Language	Client Version	Release Date			
Java	3.141.59	2018-11-14	Download	Change log	Javadoc
C#	3.14.0	2018-08-02	Download	Change log	API docs
Ruby	3.14.0	2018-08-03	Download	Change log	API docs
Python	3.14.0	2018-08-02	Download	Change log	API docs
Javascript (Node)	4.0.0-alpha.1	2018-01-13	Download	Change log	API docs

Fonte: Elaborada pelo autor

Com o Eclipse aberto, deve-se criar um novo projeto, clicando com o botão direito, acessando a opção "New"e "Project". O mesmo deve utilizar o "JavaSE 1.8".

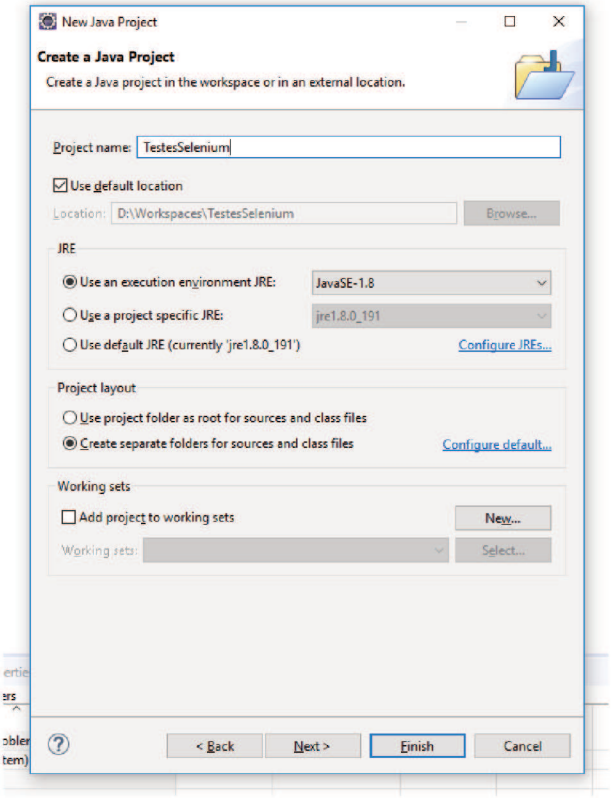
Figura 10 – Caso de teste para tela de login.



Fonte: Elaborada pelo autor

[5] Selenium HQ. Disponível em: <https://www.seleniumhq.org/>

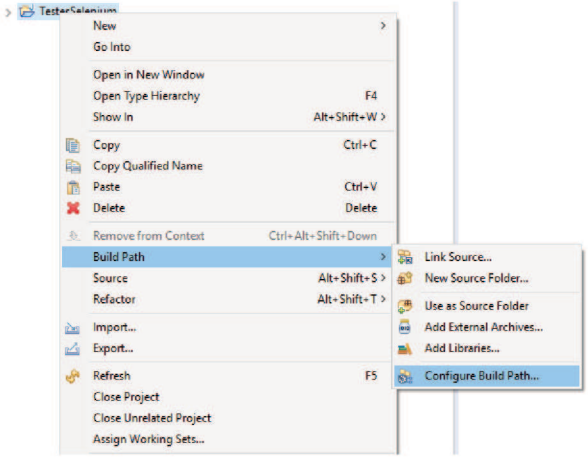
Figura 11 – Caso de teste para tela de login.



Fonte: Elaborada pelo autor

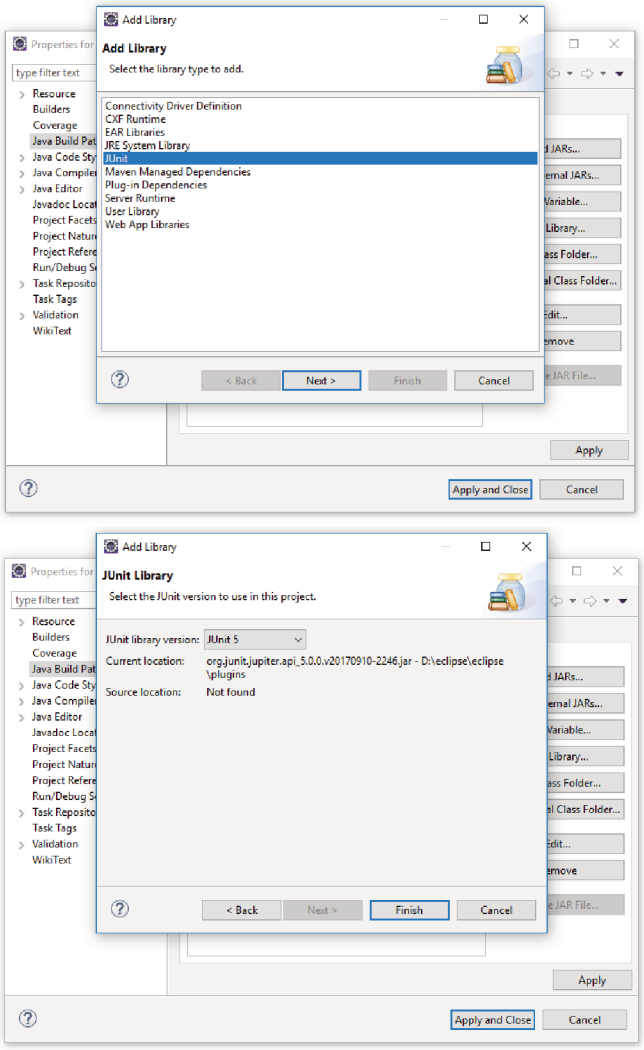
Após a criação do projeto, deve-se configurar o caminho de construção do Java, clicando com botão direito no projeto criado, "*Build Path*"e "Configure Build Path". Nesse local configuraremos as bibliotecas externas e os recursos que a aplicação utilizará. Primeiramente deve-se adicionar uma biblioteca clicando em "*Add Library*"e após isso, "JUnit", ao qual adicionaremos a biblioteca nativa "JUnit 5"em nosso projeto.

Figura 12 – Acessando o caminho de construção Java.



Fonte: Elaborada pelo autor

Figura 13 – Adicionando a biblioteca JUnit 5



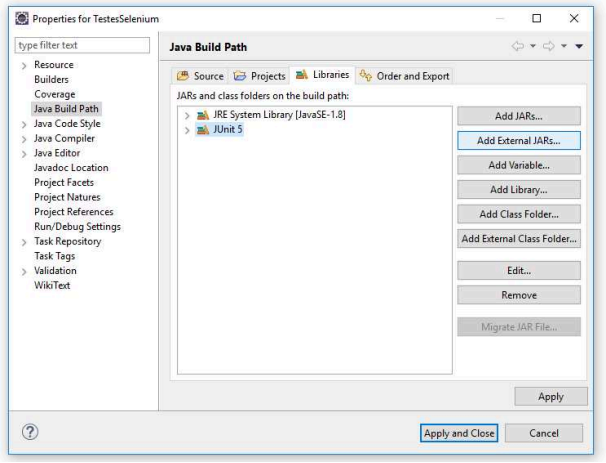
Fonte: Elaborada pelo autor

Com o JUnit adicionado, adiona-se então a biblioteca externa, que foi efetuado o download pelo site do Selenium HQ. A mesma pode ser adicionada através do botão "Add External JARs..."e selecionando a biblioteca a partir do local que ela foi extraída.

Depois disso deve-se criar um pacote e uma classe, para iniciar o desenvolvimento dos testes.

Neste trabalho específico, foram criados três pacotes, os quais, um pacote tem como função armazenar as classes de acesso as páginas, outro pacote tem como objetivo armazenar as classes com a codificação das funcionalidades de cadastro e o terceiro com o objetivo alocar as classes de edição de dados existentes na aplicação. As funcionalidades testadas através do Selenium WebDriver foram as mesmas validadas através do Selenium IDE.

Figura 14 – Adicionando a biblioteca externa.



Fonte: Elaborada pelo autor

As classes de acesso às páginas possuem a URL base da aplicação, o caminho do "driver"que efetuamos o download através do github anteriormente e a "engine"utilizada pelo Mozilla Firefox, disponível também no recurso supracitado, assim como invocar os métodos dos casos de teste localizados nas outras classes.

As classes de cadastro e edição tem como objetivo definir as ações dos testes e mapear os elementos e formulários encontrados na tela. Da mesma forma como no Selenium IDE, os elementos devem ser mapeados por um identificador único, dependendo de como a interface da aplicação foi desenvolvida.

6 Conclusão

Após a coleta de dados efetuada durante a aplicação dos testes tanto com a ferramenta Selenium IDE quanto a Selenium WebDriver os dados foram agrupados, avaliados, tabelados e categorizados entre, do pior para o melhor, "ruim, regular, bom, muito bom", sendo classificados da seguinte forma:

Tabela 1 – Resultado dos testes

Parâmetro	Selenium IDE	Selenium WebDriver
Usabilidade	Boa	Regular
Performance	Regular	Boa
Customização	Ruim	Muito boa
Abrangência	Regular	Regular

Tanto o Selenium IDE quanto o Selenium WebDriver possuem seus pontos fortes e fracos. Durante o desenvolvimento desse trabalho, esbarrou-se em algumas complicações em ambas as aplicações.

Durante os testes efetuados através do Selenium IDE identificou-se uma facilidade e uma grande praticidade em sua utilização, não necessitando um conhecimento prévio de programação, já que a ferramenta utiliza de um sistema de gravação e replicação. Não há a necessidade de instalação de nenhum software externo além do navegador que será utilizado e o mesmo apresenta uma interface simples, limpa e intuitiva.

Os problemas encontrados em sua utilização foi a necessidade de mapear uma série de elementos pertencentes a interface da aplicação. O mesmo utiliza-se de alguns padrões para o mapeamento, sendo um deles, a classe dos elementos na tela. Em algumas aplicações web, que foi o caso na aplicação testada neste trabalho, os elementos nem sempre possuem classes únicas ou valores de identificação, sendo assim necessário um mapeamento manual para captura-lo e identifica-lo unicamente. Outro problema identificado é a falta de eficiência dos métodos utilizados nessa ferramenta. Como há uma gravação dos comandos inseridos pelo usuário, há muita geração de comandos incorretos e/ou inúteis por parte da ferramenta de captura. No geral, essas inserções sem finalidade não impactam na finalidade dos testes, porém prejudica seu desempenho. Houve também algumas complicações com a gravação de “cache” pela aplicação, que produzia um erro ao tentar efetuar o login na aplicação, já que a mesma o efetuava automaticamente.

Já durante os casos desenvolvidos e executados através do Selenium WebDriver pode-se destacar a extrema escalabilidade da codificação. É de grande facilidade replicar o

desenvolvimento previamente feito para telas com características semelhantes. Há também a assertividade da codificação, como o código é desenvolvido por um programador e não gerado automaticamente, a codificação só executa o que de fato foi programado para fazer, sem a prolixidade encontrada no método de gravação e replicação, tornando os testes mais objetivos e assertivos. Uma das dificuldades encontradas durante a utilização do WebDriver foi a necessidade de um conhecimento de programação prévio e, mesmo utilizando a linguagem de programação Java, há uma certa necessidade de adaptação dos comandos e objetos utilizados. Esbarrou-se também em outra dificuldade que é a necessidade de outros softwares, plugins e bibliotecas externas.

Conclui-se por meio desse trabalho que há uma divergência pontos negativos e positivos em cada uma das ferramentas. Em alguns casos crê-se que a utilização do Selenium IDE é mais indicada, em aplicações com uma interface padronizada e com alto índice de utilização de identificadores únicos nos elementos, pelos testes utilizados pode-se sugerir seu uso por pessoas com pouca experiência em codificações e aplicações web com poucas telas, devido sua baixa escalabilidade.

A utilização do Selenium WebDriver pode ser indicada para aplicações com muitas telas e que haja uma grande necessidade de escalabilidade dos testes executados. Sua funcionalidade encaixa-se com maestria em telas com formulários de estrutura semelhante e/ou que possuam uma grande quantidade de campos e que sua performance seja algo de suma importância.

O WebDriver requer um conhecimento prévio de programação em uma linguagem específica, o que restringe o perfil de seus usuários. Sua complexidade não é alta porém sua necessidade de outras aplicações e/ou ferramentas pode ser desestimulante para aplicações web pequenas e de baixa complexidade.

6.1 Trabalhos Futuros

Com a finalização deste trabalho, uma gama de possibilidades é encontrada para ampliação e aperfeiçoamento do mesmo:

- Melhorar a estrutura lógica dos testes, utilizando mais elementos de orientação a objetos, como, herança, para auxiliar na manutenção preditiva do sistema de automação de testes.
- Aperfeiçoamento da codificação e mapeamentos dos elementos encontrados na interface da aplicação para melhora de desempenho dos testes executados.
- Utilizar os testes automatizados de sistema para realizar testes de desempenho, simulando a conexão de diversos usuários no sistema e permitindo prever anteci-

padamente a necessidade de compra de hardware ou ampliação na capacidade dos servidores externos.

- Ampliar exploração dos casos de testes para cobertura de casos de erros de validação, inserções incorretas de dados e busca de exceções.

Referências

ANDRADE, L. V. d. A.; VIEIRA, L. G. S. *Testes em Software Livre*. [S.l.]: Universidade Federal de Santa Catarina, 2009. Citado 2 vezes nas páginas 8 e 11.

BR, M. Mps. br–melhoria de processo do software brasileiro. 2011. Citado na página 8.

COHN, M. *Succeeding with Agile: Software Development Using Scrum*. Pearson Education, 2009. (Addison-Wesley Signature Series (Cohn)). ISBN 9780321660565. Disponível em: <https://books.google.com.br/books?id=8IglA6i_JwAC>. Citado na página 10.

DELAMARO, M.; JINO, M.; MALDONADO, J. *Introdução ao teste de software*. [S.l.]: Elsevier Brasil, 2013. Citado na página 8.

FREEMAN, H. Software testing. *IEEE instrumentation & measurement magazine*, IEEE, v. 5, n. 3, p. 48–50, 2002. Citado na página 10.

GIL, A. C. *Métodos e técnicas de pesquisa social*. [S.l.]: 6. ed. Ediitora Atlas SA, 2008. Citado na página 13.

HOLMES, A.; KELLOGG, M. Automating functional tests using selenium. In: IEEE. *AGILE 2006 (AGILE'06)*. [S.l.], 2006. p. 6–pp. Citado na página 14.

KOSCIANSKI, A.; SOARES, M. dos S. *Qualidade de Software-2ª Edição: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software*. [S.l.]: Novatec Editora, 2007. Citado na página 8.

KUMAR, D.; MISHRA, K. The impacts of test automation on software's cost, quality and time to market. *Procedia Computer Science*, Elsevier, v. 79, p. 8–15, 2016. Citado na página 6.

LUCCA, G. A. D.; FASOLINO, A. R. Testing web-based applications: The state of the art and future trends. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 48, n. 12, p. 1172–1186, dez. 2006. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2006.06.006>. Citado na página 11.

SARAVANAN, K.; PRASAD, E. P. C. Open source software test automation tools: A competitive necessity. *Scholedge International Journal of Management Development*, v. 3, n. 6, p. 103–110, 2016. Citado na página 6.

SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1. Citado na página 8.

TEAM, C. P. Cmmi for development, version 1.2. 2006. Citado na página 8.

WANG, F.; DU, W. A test automation framework based on web. In: *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science*. Washington, DC, USA: IEEE Computer Society, 2012. (ICIS '12), p. 683–687. ISBN 978-0-7695-4694-0. Disponível em: <http://dx.doi.org/10.1109/ICIS.2012.21>. Citado na página 11.

WISSINK, T.; AMARO, C. Successful test automation for software maintenance. In: *2006 22nd IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2006. p. 265–266. ISSN 1063-6773. Citado na página 6.