INTERFACE TECNOLÓGICA

SELENIUM WEB DRIVER NA EVOLUÇÃO DOS TESTES MANUAIS

SELENIUM WEB DRIVER IN THE EVOLUTION OF MANUAL TESTS

Isabela Ziviani Romanini – bella_zr@hotmail.com

Eder Carlos Salazar Sotto – eder.sotto@fatec.sp.gov.br

Faculdade de Tecnologia de Taquaritinga (FATEC) –SP –Brasil

DOI: 10.31510/infa.v16i2.627

RESUMO

Com a evolução dos modelos de gerenciamento de projetos, as práticas de métodos ágeis (Scrum) com seus ciclos curtos de desenvolvimento e seus times enxutos, impactam significativamente no modo como os testes de software hoje são feitos, por exemplo, executar um teste de regressão após uma mudança no código de maneira manual acaba sendo oneroso gastando muito tempo do analista de teste e gerando um custo a mais para a empresa. Com isso surgiram novas maneiras de se fazer testes de software, são eles os testes automatizados. Encontramos inúmeras ferramentas para a aplicação desses testes. Este artigo aborda por meio de vivências e pesquisas uma introdução para aplicação de automação de testes utilizando o framework Selenium WebDriver e o TestNG. No final, os profissionais interessados na área de teste terá uma base sobre como aplicar as ferramentas para aumentar a confiabilidade e a agilidade na execução de casos de teste.

Palavras-chave: Selenium WebDriver. Teste automatizados. Teste funcional.

ABSTRACT

As project management models evolve, agile method practices (Scrum) with their short development cycles and lean teams have a significant impact on how software testing today is done, for example performing a regression test. Following a manual change of code ends up being costly wasting a lot of time from the test analyst and costing the company more. With that came new ways to do software testing, they are automated testing. We found numerous tools for applying these tests. This paper discusses through experiences and research an introduction to test automation application using the Selenium WebDriver framework and TestNG. In the end, test professionals will have a foundation on how to apply tools to increase reliability and agility in test case execution.

Keywords: Selenium WebDriver. Automated testing. Functional test.

1 INTRODUÇÃO

O teste de *software* é visto como uma das fases do processo de desenvolvimento de *software* que visa garantir a qualidade do sistema, ou seja, garantir que todos os requisitos levantados pelo cliente estão funcionando de acordo com o que foi especificado.

Com a evolução da área de desenvolvimento de *softwares*, surgiram várias ferramentas, padrões e modelos de produção que exigem que não só que o profissional desenvolvedor execute códigos para testar o seu código, como também os analistas de testes desenvolvam scripts de testes nas suas execuções para garantir que o *software* desenvolvido esteja dentro das especificações do cliente. E códigos testando códigos aumentam a confiabilidade dos testes.

Com isso, iniciou-se o processo de automatização dos testes, além de garantir uma assertividade maior, ele reduz o tempo que seria gasto com testes manuais.

De acordo com Myers (2004), o emprego de testes automatizados contribui significativamente para a redução de custos e tempo de projeto durante o processo de desenvolvimento.

Sistemas de software devem não só atender as especificações do cliente, mas também ser desenvolvido de forma segura, eficiente, flexível, de fácil manutenção e evolução. Para atender esses padrões de qualidade alguns dos processos podem ser feitos de forma automatizada com o uso de *frameworks* como *Selenium* e *TestNG*.

Selenium é uma ferramenta utilizada para automatização com integração direta ao navegador, permitindo o testador executar os testes no ambiente real da aplicação.

TestNG é um framework inspirado no JUnit que serve para a organizar os testes, escrever a lógica do negócio, acrescentar anotações e comparar o resultado obtido com o resultado esperado dos testes.

O objetivo do presente estudo é ressaltar a importância e o uso de testes automatizados na construção e manutenção de sistemas, para garantir agilidade e assertividade nos testes, fazendo uso dos frameworks mencionados.

2 TESTE DE *SOFTWARE* E A SUA IMPORTÂNCIA

O teste de *software* é um conjunto de atividades dinâmicas que consistem na execução de um programa com algumas entradas específicas, visando a verificar se o comportamento do programa é condizente com sua especificação (BERTOLINO; DELAMARO, 2007).

Não se pode garantir que um *software* esteja isento de erros. O tamanho do projeto e a quantidade de pessoas envolvidas elevam a complexidade do *software*, aumentando assim também a quantidade de possíveis erros a ser encontrados.

Segundo Delamaro, Andrade e Junior (2017), negligenciar as atividades de teste, muitas vezes, pode remeter à produção de software de má qualidade e prejuízos econômicos. Isso também, pode gerar uma não confiança na marca e/ou produto.

Falhas no desenvolvimento do *software* podem ser originadas por vários motivos, a especificação pode estar incompleta, a implementação pode estar errada e também pode haver erros no algoritmo. Portanto, uma falha pode estar relacionada a um ou mais defeitos dentro do sistema.

Hoje, inúmeras empresas têm a informação como um dos maiores capitais, o que garante a segurança desses dados e a sua integridade são os sistemas. Se um sistema não foi testado, não é possível garantir a qualidade do mesmo. É por esse motivo que as fábricas de *software* têm investido em testes para garantir a qualidade de seus produtos.

3 AUTOMATIZAÇÃO DE TESTES

Automatização consiste em construir um *software* externo para comparar os resultados obtidos com os resultados esperados.

Segundo Rafi (2012), aumento da qualidade do produto, alto índice de cobertura, redução do tempo de teste, aumento da confiança, diminuição de esforços humanos e redução de custos são alguns dos benefícios dos testes automatizados.

De acordo com o manifesto ágil, o software deve ser entregue funcionando, nas escalas de semanas a meses, com preferência aos períodos mais curtos. A automatização de testes é imprescindível no desenvolvimento ágil, pois ela ajudará o software a se tornar sustentável.

Segundo Crispin Lisa e Gregory Jane (2009), é necessário poder executar os testes quantas vezes forem necessários e de forma rápida, para obter resultados a respeito da qualidade do código.

3.1 Quando devo automatizar

Utilizando a frase de Kent Beck criador do modelo de desenvolvimento de software *Extreme Programming* “Qualquer funcionalidade que não possui testes automatizados simplesmente não existe.” (apud CHEQUE, 2008, p. 43). Com isso, podemos entender que

testes automatizados são fundamentais em um ambiente ágil e os ciclos curtos exigem que sejam automatizados sempre que possível.

De acordo com Campos (2010) em sua publicação “Quando automatizar” diz que a automatização de teste pode ser utilizada em qualquer fase do processo de teste como por exemplo, para especificar casos de teste, para gerar métricas, para executar testes, para montagem de ambientes, etc. Porém, salientam que a automação deve ser um passo a ser dado quando existir um processo de teste bem estruturado e uma equipe preparada, pois exige alto conhecimento técnico e deve ser apoiado por um processo maduro.

Campos (2010) também menciona que alguns dos cenários para automatização de testes de software são os testes de regressão, testes manuais repetitivos e a preparação de pré-condições, smoke teste, testes de cálculos matemáticos e testes onde exigem um grau de dificuldade em executá-los de forma manual, como teste de performance, testes unitários e testes de integração.

Neste estudo são apresentados três tipos de teste de sistema, que são mais frequentes no dia a dia do analista de teste:

- **Teste de regressão**

Esse teste é utilizado quando o código sofre alguma mudança por implementações novas ou até mesmo correção de bugs. É um dos melhores casos para automatização, devido a sua grande repetitividade de execução. Quando automatizados, a possibilidade do sistema ir para produção com um defeito é baixa.

- **Testes manuais repetitivos**

Humanos estão sujeitos a falha, assim como o desenvolvedor pode gerar defeitos, testadores podem executar um teste errado.

Nesse caso um mesmo teste é executado inúmeras vezes e o analista de teste pode se distrair, perder o foco e executar o teste de maneira errada. Sendo assim, justifica-se automatizar para garantir que os testes estejam livres de falhas humanas.

- **Preparação de pré-condições**

Nesses testes o testador precisa preparar uma “massa de teste”, que seria uma pré-condição para executar a validação do teste propriamente dito (este feito de forma manual).

Uma vez que já tenha o controle da ferramenta de automatização, deve-se utilizá-la para aumentar a produtividade.

É importante pensar em testes e testes automatizados em todas as fases do processo de desenvolvimento mas alguns testes podem ser mais propícios a automatização devido ao seu grande número de repetições ou complexidade, diminuindo o tempo gasto com esses e garantindo uma assertividade maior nos testes.

4 FERRAMENTAS UTILIZADAS

Nesta seção serão apresentadas as ferramentas escolhidas para a execução dos testes automatizados levando em consideração os cenários de testes construídos.

4.1 Selenium WebDriver

Uma ferramenta utilizada para automatização de testes que permite o usuário executá-los de forma rápida na própria aplicação, devido a sua função da chamada diretamente ao navegador utilizando o suporte à automatização nativo em cada um. De maneira simplificada, ele permite automatizar as ações de um usuário.

Com ele pode - se criar scripts de teste de forma simples utilizando as mais diversas linguagens de programação como Java, csharp, python, ruby, php, perl e javascript.

Ele utiliza uma API *JavaScript* extensiva que atrelada ao *JavaScript* do próprio navegador permite acesso ao documento DOM da página HTML, assim ele manipula as suas propriedades e funções (DEVMEDIA, 2013).

4.2 TestNG

O *TestNG* é um *framework* de testes inspirado no *JUnit*, mas com algumas novas funcionalidades.

Com ele é possível fazer anotações, executar testes em grandes *pools* de *threads* com várias políticas disponíveis, testar se o código é *multithread* seguro, configurar testes flexíveis, dar suporte para testes orientado a dados, dar suporte a parâmetros, além de ter um modelo de execução robusto e suportar várias ferramentas e *plug-ins*.

Com ele também é possível projetar testes que abrangem todas as categorias, como teste de unidade, teste funcional, fim-a-fim, integração e etc (TESTNG, 2004).

4.3 Conceito de page objects

Page objects é um padrão de projeto com a proposta de criar objetos para cada página *web*, utilizando orientação a objetos.

Ele permite criar repositórios de objetos com elementos da página *Web*. Assim, para cada página, deve haver uma classe correspondente. Esta classe obtém e classifica os elementos da página (*WebElements*) permitindo ao desenvolvedor manipulá-los.

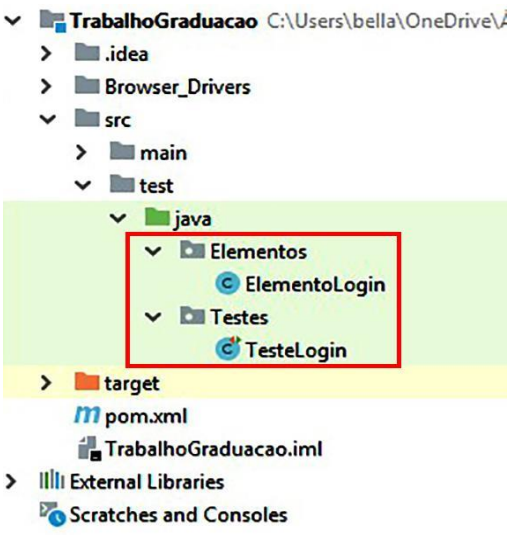
Os métodos, por sua vez, podem receber o nome das ações, por exemplo, um método de clicar em um botão poderá se chamar ClicarBotao (MEDIUM, 2017).

5 APLICAÇÃO

Os cenários de teste aqui construídos testam apenas a interação do sistema com o usuário, são testes funcionais que comparam o resultados esperado e obtidos na funcionalidade de efetuar um login em caso de sucesso (senha e login corretos) e em caso de falha (usuário inválido e/ou senha inválida), o site utilizado foi o http://the-internet.herokuapp.com/login.

A Figura 1 mostra a criação dos pacotes, um para os testes com o nome Testes e outro para o mapeamento dos elementos da página e criação dos métodos chamado Elementos.

Figura 1 – Criação da estrutura do projeto



Fonte: Elaborada pelos autores (2019)

Dentro do pacote de elementos da página é criado uma classe utilizando o conceito de *page objects*, dentro da classe os métodos criados para ações e exibição de mensagem chamam as variáveis com os elementos mapeados da página.

A Figura 2 mostra a criação das variáveis, enquanto a Figura 3 mostra os métodos criados.

Figura 2 – Variáveis criadas com elementos da página

```
package Elementos;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class ElementoLogin {

    private WebDriver driver;

    private By usuarioCampoLogin = By.id("username");
    private By senhaCampoLogin = By.id("password");
    private By botaoLogin = By.xpath("//*[@id=\"login\"]/button/i");
    public  By erroLogin = By.xpath("//*[@id=\"flash\"]");
    public By sucessoLogin = By.xpath("//*[@id=\"flash\"]");
```

Fonte: Elaborada pelos autores (2019)

Figura 3 - Métodos da classe de elementos

```
public  void clickBotaoLogin(){
    driver.findElement(botaoLogin).click();

}
public  void digitaUsuarioCampo(String usuarioCampoDigitado){
    driver.findElement(usuarioCampoLogin).clear();
    driver.findElement(usuarioCampoLogin).sendKeys(usuarioCampoDigitado);
}
public  void digitaSenhaCampo(String senhaCampoDigitado){
    driver.findElement(senhaCampoLogin).clear();
    driver.findElement(senhaCampoLogin).sendKeys(senhaCampoDigitado);
}
public void loginSucesso (String login, String senha){
    driver.findElement(usuarioCampoLogin).clear();
    driver.findElement(usuarioCampoLogin).sendKeys(login);
    driver.findElement(senhaCampoLogin).clear();
    driver.findElement(senhaCampoLogin).sendKeys(senha);
    driver.findElement(botaoLogin).click();
}
public String mensagemErro() {
    String msgErroEsperado;
    msgErroEsperado = driver.findElement(erroLogin).getText();
    System.out.println(msgErroEsperado);
    return msgErroEsperado;
}
public String mensagemSucesso(){
    String msgSucessoEsperado;
    msgSucessoEsperado = driver.findElement(sucessoLogin).getText();
    System.out.println(msgSucessoEsperado);
    return msgSucessoEsperado;
}
```

Fonte: Elaborada pelos autores (2019)

Após a conclusão, é criada uma segunda classe no pacote de teste para iniciar os testes efetivamente com o nome Testes. Com o *TestNG* é criado um método e adicionado uma notificação para ser executado antes dos métodos de teste e após (@BeforeMethod e @AfterMethod) mostrados na Figura 4.

No @BeforeMethod é executado o *driver* do *Chrome* colocado na pasta junto ao projeto, e o navegador é aberto com o endereço da página a ser testada. Ao final é adicionado uma exceção, caso a página não responda dentro de 5 segundos, o teste será falhado.

Em @AfterMethod é criado um método para encerrar o navegador assim que o teste for concluído.

Figura 4 – Anotação After e Before

```
public class TesteLogin {

    WebDriver driver;

    //chamada do driver do chrome, abre o site a ser testado e caso demore mais de 5 segundo o teste falha
    @BeforeMethod(description = "entrar no site")
    public void abreSite() {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\bella\\OneDrive\\Área de Trabalho\\BKP_Isabela\\FACULDADE\\TCC
        driver = new ChromeDriver();
        driver.get("http://the-internet.herokuapp.com/login");
        driver.manage().timeouts().implicitlyWait( l: 5, TimeUnit.SECONDS);
    }

     // fecha o navegador apos cada teste
    @AfterMethod(description = "encerrar navegador")
    public void setupFinal (){
        driver.quit();
    }
```

Fonte: Elaborada pelos autores (2019)

Na Figura 5 tem-se os métodos com os testes chamando os métodos criados com as ações na classe de Elementos:

Figura 5 – Teste de login sucesso

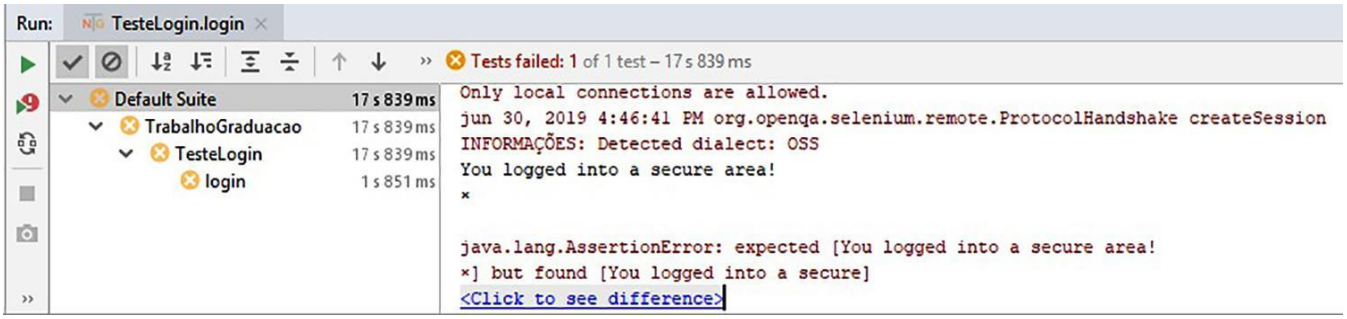
```
//Teste de login sucesso usuario e senha validos
@Test  (description = "fazer login",priority = 1)
public void login() {
    ElementoLogin ElementoLogin = new ElementoLogin(driver);
    ElementoLogin.digitaUsuarioCampo( usuarioCampoDigitado: "tomsmith");
    ElementoLogin.digitaSenhaCampo( senhaCampoDigitado: "SuperSecretPassword!");
    ElementoLogin.clickBotaoLogin();
    Assert.assertEquals( s: "You logged into a secure",ElementoLogin.mensagemSucesso());
}
```

Fonte: Elaborada pelos autores (2019)

A declaração "*AssertEquals*" (Figura 5) recebe dois parâmetros, o primeiro é o resultado esperado e o segundo é o valor retornado. O *TestNG* compara os dois valores: o resultado esperado, mencionado na declaração do teste e o resultado retornado, criado no método na classe de elementos que extrai o texto do elemento, caso o texto inserido na declaração de resultado esperado seja diferente do valor mapeado no elemento (resultado retornado) o teste irá falhar.

Na Figura 6 é exemplificado o teste falhado devido a sua declaração de resultado esperado estar diferente do resultado extraído do elemento.

Figura 6 – Resultado teste falhado



Fonte: Elaborada pelos autores (2019)

No segundo teste mostrado na Figura 7 é replicado o teste de login, porém, com o fluxo alternativo de senha inválida (digitado usuário correto e simulado uma senha inválida).

Figura 7 – Teste login erro

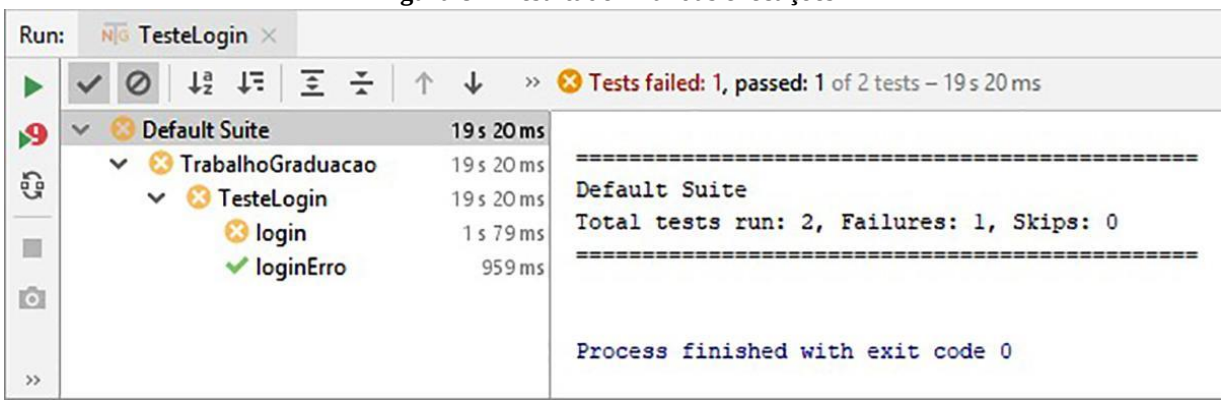
```
//Teste de login erro (senha invalida)
@Test (description = "login erro senha invalida", priority = 2)
public void loginErro(){
    ElementoLogin ElementoLogin2 = new ElementoLogin(driver);
    ElementoLogin2.digitaUsuarioCampo( usuarioCampoDigitado: "tomsmith");
    ElementoLogin2.digitaSenhaCampo( senhaCampoDigitado: "123");
    ElementoLogin2.clickBotaoLogin();
    Assert.assertEquals( s: "Your password is invalid!",ElementoLogin2.mensagemErro());
}
```

Fonte: Elaborada pelos autores (2019)

Da mesma forma ele compara a mensagem de erro esperada com a mensagem de erro atual.

A Figura 8 mostra o resultado da execução dos dois testes juntos.

Figura 8 – Resultado final das execuções



Fonte: Elaborada pelos autores (2019)

6 CONSIDERAÇÕES FINAIS

Construir um *software* requer muita atenção, pois é ele que controla todos os dados de uma organização. Um *software* testado é uma garantia maior na assertividade das informações e quando os testes são automatizados a confiança na qualidade do *software* e no tratamento das informações é ainda maior.

O objetivo do trabalho foi mostrar a importância dos testes automatizados no ciclo de vida do *software*, como a área de teste está evoluindo para uma linha onde códigos testam códigos e como as ferramentas Selenium WebDriver e TestNG podem contribuir para que o profissional aplique esses conceitos ao seu dia a dia. Entretanto, cabe ao profissional identificar e decidir qual o melhor momento para a automatização, analisar a maturidade do projeto e as características do sistema.

As possibilidades de teste que se pode fazer utilizando a ferramenta são infinitas, ambas são completas e robustas e tem apresentado um bom funcionamento no processo de automatização de teste.

REFERÊNCIAS

CAMPOS, Fabrício Ferrari. **Quando Automatizar?**. *In*: Quando Automatizar?. [*S. l.*], 2010. Disponível em: http://www.eliasnogueira.com/o-mundo-de-teste-de-software/capitulo-8-quando-automatizar/. Acesso em: 12 nov. 2019.

CHEQUE, Paulo. **Introdução a Testes Automatizados.** [*S. l.*], 2008. Disponível em: http://ccsl.ime.usp.br/agilcoop/curso_de_verao_2008. Acesso em: 15 nov. 2019.

DESENVOLVIMENTO ÁGIL. **Extreme programming**. Disponível em: <http://www.desenvolvimentoagil.com.br/xp/>. Acesso em: 27 mai. 2019.

DEV MEDIA. **A importância dos testes para a qualidade do software**. Disponível em: <https://www.devmedia.com.br/a-importancia-dos-testes-para-a-qualidade-do-software/28439>. Acesso em: 12 dez. 2018.

DEV MEDIA. **Dominando o Selenium WebDriver na prática**. Disponível em: <https://www.devmedia.com.br/dominando-o-selenium-web-driver-na-pratica/34183>. Acesso em: 28 fev. 2019.

DEV MEDIA. **Introdução aos testes funcionais automatizados com JUnit e Selenium WebDriver**. Disponível em: <https://www.devmedia.com.br/introducao-aos-testes-funcionais-automatizados-com-junit-e-selenium-webdriver/28037>. Acesso em: 25 abr. 2019.

MALDONADO, J. C. et al. **Introdução ao teste de software**. 2004 ed. São Carlos: NOTAS DIDÁTICAS DO ICMC, 2004. 56 p.

MANIFESTO ÁGIL. **Princípios**. Disponível em: <https://www.manifestoagil.com.br/principios.html>. Acesso em: 27 mai. 2019.

MEDIUM. **Page object—design pattern**. Disponível em: <https://medium.com/@nelson.souza/page-object-design-pattern-ed5f6374d32d>. Acesso em: 28 fev. 2019.

JÚNIOR, Misael; ANDRADE, Stevão; DELAMARO, Márcio. **Capítulo 6: Automatização de teste de software com ênfase em teste de unidade**. 2017. Disponível em: https://www.researchgate.net/publication/313360663_Capitulo_6_Automatizacao_de_teste_de_software_com_enfase_em_teste_de_unidade. Acesso em: 13 nov. 2019.

TESTES AUTOMATIZADOS. **Principais vantagens da automatização de testes**. Disponível em: <http://www.testesautomatizados.com.br/principais-vantagens-da-automatização-de-testes/>. Acesso em: 27 fev. 2019.

TESTNG. **Testng**. Disponível em: <https://testng.org/doc/>. Acesso em: 13 mai. 2019.