IV Encontro de Iniciação Científica e Tecnológica
IV EnICT
ISSN: 2526-6772
IFSP – Câmpus Araraquara
24 e 25 de outubro de 2019

QUALIDADE DE SOFTWARE, UTILIZANDO TESTE AUTOMATIZADO

KARINA DE JESUS RODRIGUES[1], RENATA MIRELLA FARINA[2]

[1]Graduanda em Sistemas de Informação, Uniara - Universidade de Araraquara, karinarogue_rodrigues@hotmail.com
[2]Orientadora, Uniara - Universidade de Araraquara, mirellafarina@yahoo.com.br

Área de conhecimento (Tabela CNPq): Engenharia de Software – 1.03.03.02-2

RESUMO: Será apresentado por meio deste artigo os artifícios e principais conceitos dos Testes Funcionais de Software, relacionando-os com automação. Devido a crescente expansão da tecnologia na sociedade nos últimos tempos, as empresas que trabalham com esta demanda, vêm procurando formas de atingir uma boa qualidade de software em suas entregas, tais que as facilite e melhore a produção interna de sua empresa. Por meio desta busca contínua de produtividade com qualidade, vem possibilitado o uso de ferramentas e metodologias novas para se atingir seus objetivos. A automatização de teste de software vem ganhando um grande espaço dentro das atividades pré-requisitadas no dia-a-dia de um testador. A automação de testes funcionais torna mais eficiente e rápido o encontro de inconformidades no software, em relação aos requisitos do sistema. Este artigo apresentará alguns exemplos de testes automatizados com base em ferramentas gratuitas disponíveis para web, visando aumentar a qualidade, agilidade e confiabilidade do processo.

PALAVRAS-CHAVE: automação; junit; script de teste; selenium webdriver.

INTRODUÇÃO

Durante o desenvolvimento de um software, a qualidade ganha um vasto espaço, o controle dessa qualidade é um grande desafio para os profissionais da área, pois envolve o processo de desenvolvimento, técnicas, burocracia, regras de negócio e questões humanas, visto que o software não deve apenas fazer corretamente o que o cliente estabelece, mas também visa fazê-lo de forma eficiente, segura, de fácil administração e evolução.

Levando em conta tais características, as empresas vêm investindo cada vez mais em meios de se certificar quanto à qualidade, através de testes manuais do sistema, durante seu desenvolvimento e após o seu término, seja ele dividido em módulos ou inteiro, com intuito de encontrar possíveis falhas e defeitos, para que assim evite que futuros erros sejam encontrados pelo usuário ou cliente, de forma que garanta que o software final seja de qualidade, seguro e de confiabilidade.

Este artigo abordará o assunto por meio de levantamento bibliográfico baseado em livros e artigos científicos de maneira que leve a discutir sobre os artifícios para se atingir a qualidade de um software, em especial os Testes Automatizados, com base em ferramentas gratuitas disponíveis para aplicações web.

Neste contexto, este artigo será organizado da seguinte forma: Na seção Fundação Teórica é realizado a revisão bibliográfica do tema, com base em livros, artigos científicos e sites específicos relacionados à área de tecnologia. Na seção Metodologia, será apresentado o estudo de caso, onde será simulado o preenchimento de um formulário de forma automatizada pelo site ToolsQA com uso das ferramentas Selenium WebDriver, JUnit e Java. A seção Resultados e discussão será designada a apresentar e interpretar os resultados alcançados do estudo de caso, de forma direta e objetiva, a fim de apontar sua significância e sua relevância na área de qualidade de software além de mostrar a performance dos testes

automatizados em relação aos testes manuais. Na seção Conclusões será sintetizado as possíveis respostas aos problemas envolvendo os testes manuais e automatizados, com seus objetivos e hipóteses finais, relacionado ao estudo de caso.

FUNDAMENTAÇÃO TEÓRICA

A maioria das organizações tem como objetivo alcançar um alto nível de qualidade de produto ou serviço, sendo um software como estes, não é mais tolerável à entrega de algum produto com baixa qualidade ou de reparação de problemas somente após a entrega ao cliente (SOMMERVILLE, 2004). No entanto a qualidade do software não pode ser definida de maneira simples, é um conceito complexo. Classicamente, a noção de qualidade tem sido a de que o produto desenvolvido deve cumprir com sua especificação (CROSBY, 1979).

A construção de um software por mais que bem planejada, pode ocorrer o surgimento de erros (GAEA, 2019a). O teste de software surge justamente para auxiliar neste processo de qualidade, para que seja verificado e analisando cada fluxo antes da entrega do produto, encontrando possíveis erros ou falhas no software em tempo para correção, a fim de garantir a satisfação total dos clientes (QUALITY, 2019).

O teste de um software, contudo, nos permite encontrar erros, falhas ou defeitos, antes que o produto desenvolvido seja entregue ao cliente final (SILVA, 2011a).

Figura 1 -Conceitos Defeito, Erro e Falha
Fonte: testesw.wordpress.com/conceitos-basicos/, 2019

A independência desta plataforma possibilita o programa ser executado em diferentes plataformas e sistemas operacionais, através de um emulador conhecido como a Máquina Virtual Java ou JVM (*Java Virtual Machine*) que ajuda rodar os sistemas baseados em Java (PALMEIRA, 2012).

Assim como no código Java, a partir da criação de classes e métodos que executam casos de teste de pequena, média ou grande complexidade, as classes do JUnit ficam encarregadas de testar o software com suas asserções (SILVA, 2013).

O Selenium é um *framework* para testes de *software web,* que facilita a automação de testes dos sistemas, que permite ao usuário reproduzi-los rapidamente no ambiente real da aplicação, conforme sua integração direta com o navegador. Em geral o *script* é criado em formato *HTML*, e pode ser exportado e editado para testes em diversas outras linguagens inclusive o Java (SOUSA, 2016).

A automação de testes tem como grande objetivo diminuir a prática dos testes manuais repetitivos durante todo o processo, a fim de agilizar a análise e minimizar o índice de erros, quem é familiarizado com os testes manuais sabe o quanto é demorado e repetitivo tal processo, então a automação promete mudar este cenário, envolvendo o uso de ferramentas/softwares específicos que tenham a função de executar testes automáticos das funcionalidades colocadas no software desenvolvido. A automatização como aliada a rastrear falhas, ganha mais força por realizar entregas do produto finalizado mais rápido e com melhor qualidade, pois ao contrário dos testes manuais a automatização exige menos esforço físico e tempo para assegurar que suas funcionalidades implementadas cumpram com o seu papel (GAEA, 2019b).

A automação é considerada um complemento na etapa de testes manuais, substituindo assim aqueles testes repetidos várias vezes, que geram desinteresse profissional e consequentemente produz erros durante o

processo de teste devido a repetição, a automatização permite que os profissionais da área dedique mais tempo a outras atividades a fim de complementar a qualidade do produto final a ser entregue, tornando-os mais criativos a resolução de problemas, com mais tempo a ser gasto de forma estratégica, planejando e executando planos melhores de automação e aplicando mais testes exploratórios no qual a automação não substitui (YACKEL, 2018).

METODOLOGIA

Nesta seção será abordada a estruturação do caso de teste de duas formas distintas, seguindo os mesmos critérios de aceitação.O primeiro teste será realizado manualmente por um profissional da área de testes de software, e seu tempo de execução será cronometrado e apresentado na seção de resultados e discussão. O segundo será desenvolvido um *script* de teste com os mesmos critérios utilizados no teste manual, bem como as ferramentas e metodologias utilizadas para a criação dos cenários de forma automatizada. A validação dos testes se dá sempre se baseando nos resultados esperados de determinada funcionalidade. Ao desenvolver um teste que atenda esses resultados, são definidos critérios de aceitação que consolidam o resultado esperado a partir da execução e confirmação do mesmo. Portanto, o *script* terá uma descrição do que ela fará, e um critério de aceitação para ser avaliado se o mesmo atingiu o resultado esperado.

Para a realização dos testes será utilizada uma aplicação web que disponibiliza diversos recursos para a prática de automação, entre eles o formulário que utilizaremos que se encontra em: https://www.toolsqa.com/automation-practice-form/. A descrição do *script* se dá pelo preenchimento dos campos "*First name*", "*Last name*", "*Sex*", "*Years of Experience*", "*Date*", "*Profession*", "*Automation Tool*" e o clique no botão "*Button*" e o critério de aceitação para que o teste seja válido será que apenas os campos acima serão considerados obrigatórios para preenchimento além do clique no botão. E como comprovação de que o teste passou a frase "*Always click on the ads display at the right side*" será considerada como uma confirmação de atualização da página.

Para criação do *script* será utilizado o Eclipse, Selenium Webdriver, JUnit empregados no Google Chrome através de um driver do Selenium, que executará o Google Chrome de forma automatizada. Ao escrever o *script* no Eclipse usamos algumas anotações sobre os métodos para que o JUnit entenda eles como os testes que devem ser executados. As anotações são um recurso usado para anotar classes, campos e métodos, para que essas marcações possam ser tratadas pelas bibliotecas, compilador e ferramentas de desenvolvimento em tempo de execução ou compilação. No *script* são usadas três anotações que norteiam a utilização do JUnit. A anotação *@Before* mapeia que aquele método será executado toda vez antes da execução da classe de teste, assim podendo definir configurações iniciais que serão necessárias e todos os casos de testes a terão em comum como, por exemplo, a instância de um objeto ou Classe, a configuração da página que o driver irá acessar a maximização da janela do navegador, uma possível conexão com um banco de dados entre outras que possam existir. A anotação neste caso reduz a redundância de código como deixa mais organizado, facilitando o entendimento e a manutenção do código. Já a anotação *@After* fica encarregada de finalizar o que foi aberto ou instanciado na anotação *@Before*, ou seja, tudo o que se inicia, estabelece conexão ou qualquer outra definição feita no *@Before* será finalizada no *@After* a fim de manter o código mais limpo e organizado. Por último falaremos do *@Test* que identifica pro JUnit que aquele método é uma classe de teste e deve passar pelas validações e verificações (*Asserts*) que são as formas que o sistema valida os critérios de aceitação nos testes.

A classe *WebDriver* é uma classe base do Selenium, nela é possível realizar as funcionalidades de navegação, como abrir/fechar o *browser*, definições das configurações da *URL* onde será realizada a execução do *script*, retorno do contexto da página para validar no *Assert* se determinado texto está contido na página conforme era esperado na execução do teste. A classe W*ebElement* é outra classe do Selenium que representa qualquer elemento do HTML. Dentro desta classe temos o método *findElement* que permite localizar os elementos dentro de uma página, ele utiliza o DOM (*Document Object Model*) da página e

localiza todas as ramificações com auxílio da classe *By* que determina, através do *id* se ele foi declarado, do nome da classe do elemento, ou do css ou de alguma *tag*, podendo então localizar os elementos diretamente ou até mesmo com um *Xpath* que seria um caminho dentro do DOM para a definição do elemento. Outra classe muito utilizada é a classe *Assert*, essa classe é responsável por garantir que os critérios serão aceitos podendo assim gerar um erro no *script* caso a validação não seja aceita. Utilizando o *assertTrue* para validação de um dos testes, método que verifica se o valor contido no retorno do método *getPageSource()* está exatamente igual ao definido como: *"Always click on the ads display at the right side"*, desta forma ele consegue retornar que o *script* foi executado com sucesso ou, caso algo não esteja de acordo, com erro.

```
public class FormsTest {

    WebDriver driver;

    @Before
    public void antesCadaTeste() {
        System.setProperty("webdriver.chrome.driver",
                "C:\\Users\\karina.rodrigues\\Desktop\\w_karina\\autom_form\\src\\test\\resources\\chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("https://www.toolsqa.com/automation-practice-form/");
        driver.manage().window().maximize();
    }

    @Test
    public void testeElementos() {

        WebElement campoPrimeiroNome = driver.findElement(By.name("firstname"));
        campoPrimeiroNome.sendKeys("Karina");

        WebElement campoUltimoNome = driver.findElement(By.name("lastname"));
        campoUltimoNome.sendKeys("Rodrigues");

        WebElement selectSexo = driver.findElement(By.id("sex-1"));
        selectSexo.click();

        WebElement selectExperiencia = driver.findElement(By.id("exp-3"));
        selectExperiencia.click();

        WebElement campoData = driver.findElement(By.id("datepicker"));
        campoData.sendKeys("23/01/2019");

        WebElement selectProfissao = driver.findElement(By.id("profession-1"));
        selectProfissao.click();

        WebElement button = driver.findElement(By.id("submit"));
        button.click();
    }

    @Test
    public void atualizarValidarPagina() {
        driver.navigate().refresh();
        Assert.assertTrue(driver.getPageSource().contains("Always click on the ads display at the right side"));
    }

    @After
    public void depoisCadaTeste() {
        driver.quit();
    }
}
```

Figura 2 - Script de teste automatizado.
Fonte: Elaborado pelo autor, 2019.

RESULTADOS E DISCUSSÃO

Após a criação e execução do *script* proposto na sessão anterior, foi obtido o seguinte resultado da própria ferramenta:

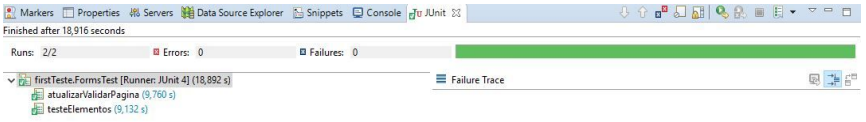

Figura 3 - Resultado do script de teste automatizado.
Fonte: Elaborado pelo autor, 2019.

A ferramenta consegue mensurar o tempo total gasto e em cada teste, como pode-se observar, o tempo total para o *script* ser executado foi de apenas 18,892 segundos. Analisando separadamente, o teste "atualizarValidarPagina" levou em média 9,760 segundos e o "testeElementos" em média 9,132 segundos. Este mesmo *script* foi cronometrado e executado manualmente por uma pessoa conforme as especificações,

levando 43,16 segundos para ser completado.

Com base nestes dados podemos afirmar que a automação de testes faz com que o teste a ser realizado seja executado mais rápido que uma pessoa executando manualmente. No *script* construído como estudo de caso, foi executado um pequeno fluxo de teste para análise, mas levando em conta que ao aplicar a automação de teste em um sistema de grande proporção, seus testes regressivos, ou o encontro de *bugs*, se tornam mais rápidos, o que pode ser observado em longo prazo, e beneficiando a equipe como um todo. Em testes futuros é mais rápido e simples rodar o *script* de automação de teste, do que refazer todos os testes manualmente, e caso haja mudanças na equipe é mais vantajoso ao novo integrante da equipe executar e entender o código desenvolvido do que passar a gastar tempo tentando entender manualmente.

CONCLUSÕES

A automação de testes pode garantir uma maior agilidade na execução de testes em grandes escalas, testes que são repetitivos e testes que devem ser executados com uma certa frequência, seja ela diária, semanal, mensal ou qualquer período onde é possível estipular como os testes deverão ser executados. Um outro ponto a ser observado diz respeito a ampla aplicação desses testes, como em desktop. celular e aplicações web, no qual conseguem simular casos com usuários simultâneos, ou acessos; poupando tempo, esforço e gastos consideráveis com equipamentos e tecnologias. A Figura 4 mostra a diferença no tempo de execução de algumas tarefas entre os testes automatizados e os manuais.

Figura 4 – Mensuração de tempo de execução de testes automatizados e manuais.
Fonte: Peres, 2016, n.p.

Outro ponto a ser ressaltado é que em diferentes etapas do desenvolvimento do projeto, os custos com as falhas e defeitos se tornam mais caros para serem resolvidos, pois envolve uma correção de várias etapas do projeto. Vale lembrar que além dos custos com problemas e defeitos a avaliação para serem implantados e desenvolvidos os testes automatizados é de extrema importância, pois os gastos com a implementação, execução e desenvolvimento são relativamente altos no início da aplicação desses testes. Caso ocorra muitas alterações no projeto, seja por problemas com defeitos ou falhas, seja por ainda possuir muitas etapas a serem desenvolvidas e que irão impactar no sistema, a automação se torna inviável ou com um valor extremamente alto para ser realizada. Se o projeto não ter uma vida útil longa, ou sofrer muitas alterações, ou ainda caso ele será apenas um módulo de um sistema maior, a automação se tornará inviável pois a quantidade de execução dos testes ou o ciclo de vida do projeto se encerrará muito antes do que irá começar a suprir os gastos com mão–de–obra e conhecimento de quem for desenvolver, planejar e escrever

os *scripts* dos testes automatizados.

Os teste automatizados então devem ser implantados apenas em projetos onde seu ciclo de vida seja extenso ou que possua execuções simultâneas ou muito repetitivas para que valha a pena o esforço e para criação dos *scripts* e os custos iniciais para que os testes automatizados sejam realizados no sistema. Em contrapartida, os testes automatizados aumentam o desempenho do time, melhora a qualidade do projeto, garante uma qualidade mais precisa em execuções de testes de grande escala por não possuir um desgaste ou diminuição de tempo de execução comparado a uma pessoa executando.

REFERÊNCIAS

CROSBY, Philip Bayard. ***Quality is free**: The art of making quality certain. New York: New American Library*, 1979.

GAEA. **Não cometa esses 7 erros ao desenvolver um software**. Disponível em: <https://gaea.com.br/nao-cometa-esses-7-erros-ao-desenvolver-um-software/>. Acesso em: 30 abr. 2019a.

GAEA. **Os Desafios da Automatização de testes**. Disponível em: <https://conteudo.gaea.com.br/desafios-da-automatizacao-de-testes>. Acesso em: 04 jun. 2019b.

PALMEIRA, Thiago Vinícius Varallo. **Java: História e principais conceitos**. Devmedia, 2012. Disponível em: <https://www.devmedia.com.br/java-historia-e-principais-conceitos/25178>. Acesso em: 31 mai. 2019.

PERES, Hugo. **Automatizando Testes de Software Com Selenium**. 2012. Disponível em: <https://books.google.com.br/books/about/Automatizando_Testes_de_Software_Com_Sel.html?id=ok6_DQAAQBAJ&printsec=frontcover&source=kp_read_button&redir_esc=y#v=onepage&q&f=false>. Acesso em: 17 ago.2019.

QUALITY. **Fábrica de testes e qualidade de software: o que é, o que faz?**. Disponível em: <http://blog.quality.com.br/fabrica-de-testes-e-qualidade-de-software-o-que-e-o-que-faz/>. Acesso em: 30 abr. 2019.

SILVA, Bruno Firmino. **Introdução aos Testes Funcionais Automatizados com JUnit e Selenium WebDriver**. Devmedia, 2013. Disponível em: <https://www.devmedia.com.br/introducao-aos-testes-funcionais-automatizados-com-junit-e-selenium-webdriver/28037>. Acesso em: 03 jun. 2019.

SILVA, Fernando Rodrigues. Testes de software - entendendo defeitos, erros e falhas. Devmedia, 2011a. Disponível em: <https://www.devmedia.com.br/testes-de-software-entendendo-defeitos-erros-e-falhas/22280>. Acesso em: 27 mai. 2019.

SOMMERVILLE, Ian. **Engenharia de software**: Gerenciamento de Qualidade. 6 ed. São Paulo: Person Education, Inc, 2004. 458 p.

SOUSA, Sueila. **Dominando o Selenium WebDriver na Prática**. Devmedia, 2016. Disponível em: <https://www.devmedia.com.br/dominando-o-selenium-web-driver-na-pratica/34183>. Acesso em: 04 jun. 2019.

YACKEL, Ryan. **A automação de testes de software atinge a maturidade**. Administradores, 16 agosto 2018. Disponível em: <https://administradores.com.br/noticias/a-automacao-de-testes-de-software-atinge-a-maturidade>. Acesso em: 04 jun. 2019.