Um Comparativo na Execução de Testes Manuais e Testes de Aceitação Automatizados em uma Aplicação Web

Vanilton de Souza Freire Pinheiro[1], Natasha M. Costa Valentim[2], Auri Marcelo Rizzo Vincenzi[34]

[1]Centro Universitário do Norte – Laureate International Universities (UNINORTE) Manaus – AM – Brasil

[2]Universidade Federal do Amazonas (UFAM), Manaus – Amazonas – Brasil

[3]Universidade Federal de Goiás (UFG), Instituto de Informática - Goiânia, GO – Brasil

[4]Universidade de São Paulo (USP), Instituto de Ciências Matemáticas e de Computação – São Carlos, SP – Brasil

vanilton18@gmail.com, natasha_costa16@yahoo.com.br, auri@inf.ufg.br

Resumo. *Garantia da qualidade por meio de testes é essencial no processo de desenvolvimento de software. Entretanto, o alto custo da execução de testes manuais e o número cada vez maior de ambientes de testes web solicitados pelos clientes dificultam manter a qualidade do produto e cumprir o prazo estabelecido. Para solucionar este problema é proposto o uso da técnica Behavior-Driven Development em conjunto com Selenium WebDriver e JUnit. Este artigo apresenta um estudo de caso da técnica, relatando os ganhos obtidos como redução no esforço de execução de testes em uma aplicação web. Também são apresentadas as lições aprendidas do estudo de caso.*

Abstract. *Quality assurance through testing is essential in the software development process. However, the high cost of performing manual tests and the growing number of web testing environments requested by customers, difficult to maintain product quality and to meet the deadline stablished. To solve this problem we propose the use of Behavior-Driven Development in conjunction with JUnit and Selenium WebDriver. This paper presents a case study on testing a web application, reporting the gains as a reduction in test execution effort. We also present the lessons learned from the case study.*

1. Introdução

Tendo em vista a necessidade de resultados cada vez mais rápidos no desenvolvimento de software web, também decorrente da evolução de tecnologias e aumento das possibilidades de uso como diversos navegadores, dispositivos e sistemas operacionais, a busca por qualidade também é crescente. Uma forma de garantir a qualidade do software é por meio de atividades de Verificação e Validação (V&V), cujo objetivo é mostrar que um sistema está em conformidade com sua especificação e que atende as expectativas do cliente [Sommerville, 2011]. Com a complexidade da tecnologia e consequentemente das aplicações e com o advento do ambiente cliente/servidor e

posteriormente da web, partes dos sistemas (ou ele todo) se tornaram mais complexos de serem testados [Rios e Moreira 2013].

Sem a utilização de ferramentas que automatizem a aplicação das técnicas e critérios associados, a atividade de teste torna-se onerosa, propensa a enganos e limitada a programas simples [Delamaro *et al.*, 2007]. Para solução destes problemas, técnicas como *Test Driven Development* (TDD), *Acceptance Test Driven Development* (ATDD) e *Behavior-Driven Development* (BDD) [Wynne e Hellesoy, 2012] têm sido propostas. Neste artigo utilizou-se os testes automatizados de aceitação utilizando a técnica BDD em conjunto com os arcabouços de teste Selenium WebDriver [Selenium, 2008] e JUnit [Tahchiev *et al*., 2011]. Escolheu-se a técnica BDD pois esta propõe o alinhamento das expectativas dos comportamentos do software através do uso da linguagem *Gherkin* [Wynne e Hellesoy, 2012], além de estimular a colaboração de todos envolvidos no projeto, aumentando o conhecimento entre eles.

A intenção com esta abordagem é automatizar atividades repetitivas e com grande intervenção humana no processo de teste. O objetivo é obter uma redução no esforço durante a execução do teste, permitir a dedicação de tempo para elaboração de cenários mais complexos, reduzir a incidência de falhas em produção e facilitar a compreensão dos cenários descritos em BDD por meio de sua linguagem *Gherkin*, que se aproxima da linguagem natural.

Desse modo, este artigo apresenta um relato de experiência resultante da aplicação da abordagem proposta em um sistema web, caracterizando os desafios e ganhos obtidos, além de citar as lições aprendidas que podem ser levadas em consideração em projetos futuros considerando contextos similares.

As próximas seções deste artigo estão organizadas da seguinte forma: a Seção 2 apresenta conceitos básicos e terminologias para melhor entendimento do texto. A Seção 3 descreve o estudo, a aplicação utilizada com os cenários de aceite, as ferramentas mencionadas na Seção 2 e as lições aprendidas do estudo de caso. Por fim, a Seção 4 descreve as considerações finais e trabalhos futuros.

2. Terminologia e Conceitos Básicos

A atividade de teste, se realizada manualmente, geralmente é propensa a enganos e se restringe a aplicações de pequeno porte. Nesse contexto, ferramentas de teste podem auxiliar na automatização dessa tarefa, permitindo a utilização de critérios de teste em programas maiores e mais complexos, o apoio a estudos experimentais e a transferência das tecnologias de teste para a indústria [Domingues, 2002].

Segundo Delamaro *et al*. (2007), a técnica de automação é voltada principalmente para melhoria da qualidade, baseando-se fortemente na teoria de teste de software, para aplicar as recomendações dos testes manuais nos testes automatizados.

A automatização de testes se intensificou com as práticas ágeis e se mostra eficaz para melhorar a qualidade do software [Dustin *et al.*, 2009]. Porém, inicialmente, a adoção da técnica é cara, pois demanda um tempo maior para elaboração dos *scripts* e requer mão de obra qualificada [Chiavegatto *et al.*, 2014].

Um ganho essencial que pode ser identificado na automatização de testes segundo Chiavegatto *et al*. (2014), refere-se ao custo benefício quando é necessário

executar testes repetitivos e de regressão. Também podem ser identificados ganhos com o reaproveitamento dos cenários já implementados, proporcionando confiança em possíveis alterações no software. Além de gerar pouco esforço de testes manuais nos fluxos principais da aplicação, atuando de forma preventiva na incidência de defeitos críticos em produção, decorrentes da manutenção realizada. No desenvolvimento deste trabalho a automatização é utilizada juntamente com a técnica BDD que, segundo Chiavegatto *et al.* (2014), facilita a manutenção e possibilita uma documentação mais próxima dos *scripts* elaborados. Existem muitas ferramentas de apoio ao uso da técnica BDD. Neste relato de experiência será utilizado o Cucumber-JVM [Wynne e Hellesoy, 2012] para especificação dos cenários de teste conforme ilustrado na Figura 1.

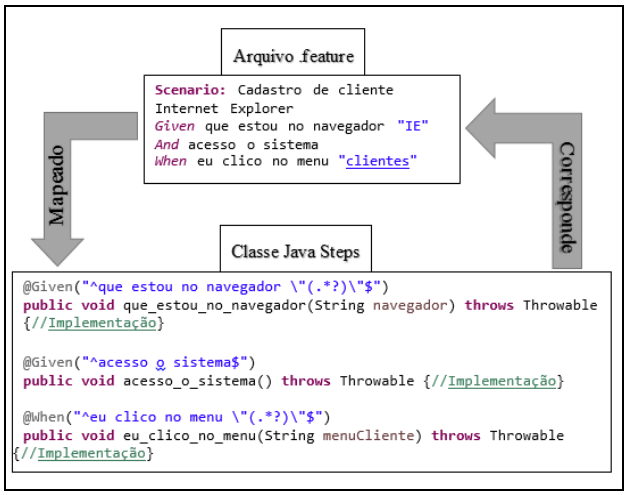

Figura 1. Exemplo de Mapeamento do teste de aceitação no Cucumber-JVM

Na Figura 1 pode-se observar a especificação em *Gherkin* do comportamento do sistema utilizando a técnica BDD [North, 2006]. Além disso, na Classe Java Steps pode-se notar seu mapeamento para linguagem de programação. Para a execução dos testes (cenários especificados), utilizam-se os arcabouços de teste Selenium WebDriver [Selenium, 2008] e JUnit [Tahchiev *et al*., 2011]. Tais ferramentas e arcabouços foram escolhidos considerando que são *Open Source,* podendo ser adaptados conforme a necessidade, também são passíveis de integração e podem ser utilizados em várias linguagens de programação, proporcionando a integração em vários projetos das mais variadas tecnologias. Além disso, apresentam boa documentação e uma comunidade de desenvolvimento bastante ativa. Para informações detalhadas sobre as tecnologias empregadas as referências oferecidas podem ser consultadas.

- ***Behavior-Driven Development* (BDD)**

A técnica ágil *Behavior-Driven Development* (BDD) busca melhorar o entendimento do negócio descrevendo cenários por meio da linguagem *Gherkin,* ou seja, tornando-a ubíqua e compreensível a todos envolvidos no projeto [North, 2006]. Outro benefício que a técnica BDD oferece é a possibilidade de testes automatizados de aceitação, atendendo os cenários de aceite ao quais o sistema se propõe, sem a necessidade de uma intervenção humana para verificar que tais critérios são ou não atendidos.

O foco principal da técnica BDD é o comportamento do software, entretanto ela também oferece benefícios como colaboração e entendimento do que o software deve atender, diminuição de incidência de falhas em teste e redução de retrabalho decorrentes de incompreensão das regras de negócio do sistema. Além de gerar uma documentação rica em detalhes, que pode ser facilmente alterada e atualizada.

3. Estudo de Caso

Esta seção detalha como foi desenvolvido o estudo de caso cujo o objetivo é apresentar uma comparação entre as abordagens de testes manual e automatizada, variando a quantidade de ambientes nos quais os testes devem ocorrer e a quantidade de repetições de teste. Além disso, busca demostrar a diferença de tempo entre as duas abordagens. Um analista de teste com formação em Ciência da Computação e 3 anos de experiência na construção de casos de teste estava envolvido no projeto realizando a especificação, implementação e execução de todos os testes deste estudo de caso. Os casos de teste elaborados exploraram as entradas do sistema por meio dos seguintes critérios de teste: Análise de Valor Limite e Partição de Equivalência [Delamaro *et al*., 2007].

3.1 Aplicação Utilizada

Neste estudo de caso foi utilizada uma aplicação web chamada Empresa que realiza cadastros de Clientes, Recibos e Serviços. A aplicação está disponível em: https://github.com/Vanilton18/empresa/. Informações gerais sobre estrutura do projeto, foram calculadas pela ferramenta de gerenciamento de qualidade de código SonarQube [SonarSource, 2008] e são apresentados na Tabela 1.

Tabela 1. Informações estruturais da aplicação coletadas pelo SonarQube

Linhas de código	Complexidade das Classes	Complexidade das funções	Qtde de Classes	Qtde de funções
1402	6,4	2	26	82

Para o cadastro de Clientes deverão existir os campos Nome, E-mail, Endereço e Histórico. Para o cadastro de Serviços devem haver os campos Descrição e Valor. No cadastro de Recibos são requeridos a Data, Serviço utilizado e Identificador do Cliente. Os testes de aceitação automatizados devem validar os seguintes cenários de aceite:

- Para um Cliente ser válido, o mesmo deve ter todos os campos preenchidos, exceto Histórico.
- Para um Serviço ser válido deve possuir Descrição e Valor.
- Para um Recibo ser válido deve possuir Data, Nome do serviço e Nome do Cliente.
- A aplicação deve ser executada nos navegadores Chrome, Firefox, Internet Explorer, Opera e Safari nas suas versões mais atuais.

3.2 Condução dos Testes

O processo utilizado para realização dos testes automatizados de aceitação é composto por seis atividades organizadas conforme ilustrado na Figura 2. O processo é iniciado na (Atividade 1) com a especificação de um arquivo *.feature.* Este arquivo irá conter todos os passos de interação com a aplicação utilizando a linguagem *Gherkin* por meio da ferramenta Cucumber-JVM. A Figura 3 apresenta uma *feature* com o cenário de cadastro de cliente no navegador Internet Explorer.

Na segunda atividade do processo (Atividade 2) é criado um projeto Java em um *Integrated Development Environment* (IDE), utilizando-se a ferramenta Maven [Apache, 2002] para gerenciar as dependências necessárias para o projeto. Após isto, na (Atividade 3) é criada a classe de configuração e também uma classe de Step que dará suporte ao Cucumber-JVM para identificar quais métodos foram implementados.

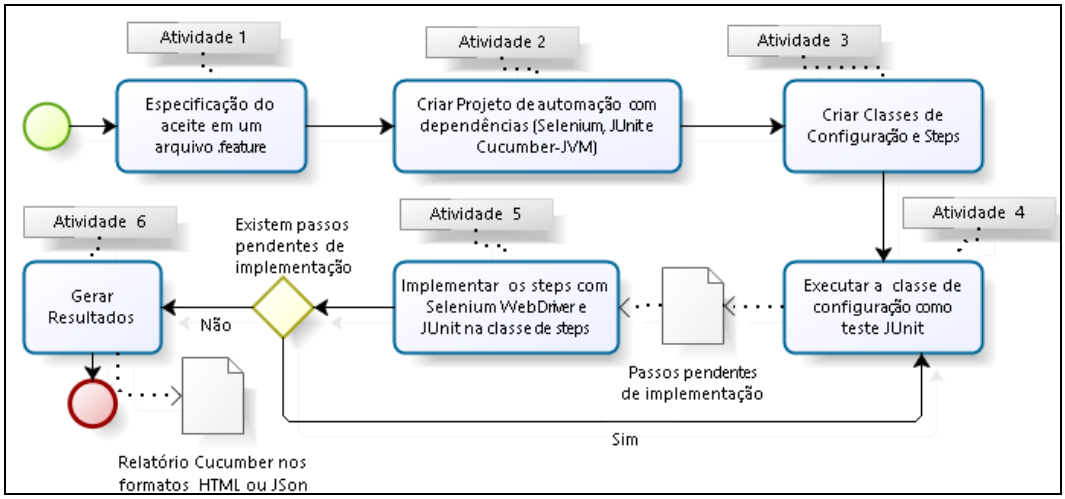

Figura 2. Processo de teste de aceitação automatizados

```
Feature: Cadastro de Cliente

Scenario: Cadastro de cliente Internet Explorer
Given que estou no navegador "IE"
      And acesso o sistema
When eu clico no menu "clientes"
Then eu estou na tela "Gerenciar Clientes"
When eu clico no botão "Adicionar"
Then eu estou no modal "Manter Cliente"
When eu preencho os campos do modal Manter Cliente
|Nome         |Email                |Endereço         |Histórico   |
|José Firmino|josefirmino@gmail.com |Av. itabúba N 40|Sem Histórico|
      And eu clico no botão "Salvar"

Then eu verifico a mensagem "Cliente cadastrado com sucesso!"
```

Figura 3. Feature de Cadastro de Cliente

Para gerar os métodos a serem implementados é preciso executar a classe de configuração como teste JUnit conforme requisitado pela (Atividade 4). Após isso o Cucumber-JVM realizará o mapeamento dos cenários do arquivo *.feature* e exibirá no console da IDE os cenários a serem implementados.

Considerando a *feature* apresentada na Figura 3, a Figura 4 ilustra uma parte do código vazio gerado pelo Cucumber-JVM agora implementado com Selenium WebDriver e JUnit, estes responsáveis pela automatização das funcionalidades das *features*, conforme requisitado pela (Atividade 5). Por fim na (Atividade 6), é gerado o relatório automático por meio do Cucumber-JVM, que contemplará os cenários que passaram e falharam na execução.

3.3 Resultados Obtidos

Para análise dos resultados quanto ao ganho na execução dos testes entre o uso automatizado dos cenários de aceitação e o uso de técnica manual e especificação por script, foram levados em consideração dois fatores na comparação sendo eles: a quantidade de ambientes (navegadores, dispositivos ou sistemas operacionais) de execução e a quantidade de repetições de testes.

```
public void eu_clico_no_menu(String menuCliente) throws Throwable {
	driver.findElement(By.id(menuCliente)).click();
	assertNotNull(driver);
}

@Then("^eu estou na tela \"(.*?)\"$")
public void eu_estou_na_tela(String telaCliente) throws Throwable {

	assertEquals(telaCliente, driver.findElement(By.cssSelector("h1")).getText());

}

@When("^eu clico no botão \"(.*?)\"$")
public void eu_clico_no_bot_o(String btAdicionar) throws Throwable {
	driver.findElement(By.xpath("//button[contains(.,'" + btAdicionar + "')]")).click();
}
```

Figura 4. Classe Java com métodos gerados pelo Cucumber-JVM e implementados com Selenium WebDriver

Considerando a Tabela 2, no teste manual foram especificados 20 casos de teste no tempo de 30 minutos, sendo o total de casos de teste executados em um único ambiente em aproximadamente 5 minutos. Já no teste automatizado foram especificados e implementados os mesmos 20 casos de teste no tempo de 100 minutos, sendo executados em um único ambiente no tempo de 2 minutos. Para o cálculo total gasto na execução foram considerados o tempo de especificação e implementação apenas uma única vez, sendo aplicados os fatores apenas no tempo de execução.

Tabela 2. Custos das atividades

Atividade	Automatizado	Manual	Qtde de casos de teste	Qtde - A (Ambiente), R (Reteste), T (Total de casos de teste executados)					
				1º Amostra			2° Amostra		
				A	R	T	A	R	T
	Tempo Médio gasto nas atividades (min)		20	3	2	120	5	10	1000
Especificação / Implementação	100	30							
Execução (um ambiente)	2	5							
% ganho em tempo para automatização de testes de aceitação frente o teste manual				-87%			29%		
% ganho em tempo de teste manual e especificação por script frente o teste automatizado				46%			-40%		

Conforme a 1ª Amostra foram definidos 3 Ambientes e 2 repetições, totalizando 120 casos de teste executados, o uso de teste manual e especificação por script demostrou ser 46% mais eficiente em termos de tempo de execução em relação aos testes automatizados, conforme exibido na Tabela 2.

Entretanto, na 2ª Amostra com a definição de 5 Ambientes e 10 repetições de testes, totalizando 1000 casos de teste executados, é observado um ganho de 29% no custo da execução utilizando a automatização de testes de aceitação frente aos testes

manuais. Na Figura 5 pode-se observar a tendência de gasto em horas para mais cenários neste contexto.

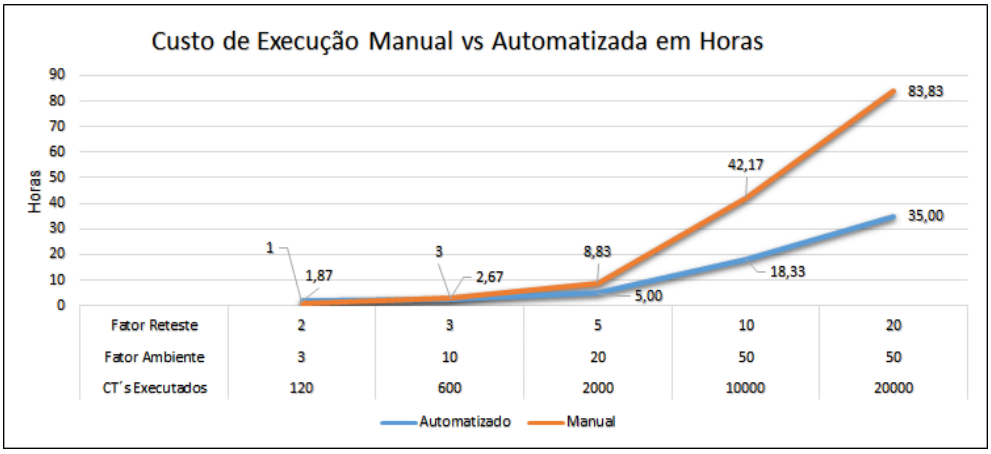

Figura 5. Custo de Execução Manual vs. Automatizada com fatores e quantidade de CT – Casos de teste

Além da redução do tempo, à medida que as atividades repetitivas se tornam mais frequentes, a automatização dos testes de aceitação provê um retorno mais rápido e permite que seja investido mais tempo com a execução de testes exploratórios em cenários mais complexos. Isto contribui para o aumento da cobertura dos cenários de testes e, consequentemente, para a qualidade final do produto. Observa-se que no último cenário da Figura 5, considerando 50 variações de ambiente e 20 repetições de testes, totalizando 20000 mil casos de teste executados, o custo de execução dos testes em termos de tempo chega a ser 48,8% menor se comparado ao custo do teste manual.

3.4. Lições Aprendidas

Pode-se notar que durante o estudo de caso a configuração/instalação do ambiente é de baixo custo tendo em vista que o conhecimento técnico do analista de teste é suficiente. Também houveram ganhos como o reaproveitamento por meio da definição de elementos genéricos e manutenção através da fácil identificação do cenário falho na execução automatizada. Além disso, houve a rastreabilidade dos *scripts* de teste obtidos, pois com o uso da técnica BDD, o *script* de teste se aproxima dos requisitos do software, não havendo a necessidade de um outro artefato com tais especificações. Também foram observados problemas com a paralelização da execução dos testes, devido os testes trabalharem com o uso de apenas um banco de dados. Outro problema identificado diz respeito às atualizações dos navegadores que impactam na funcionalidade correta das tecnologias adotadas. Para isso, é necessário manter um ambiente controlado com versionamento dos navegadores a fim de evitar problemas na execução dos testes.

4. Conclusões e Trabalhos Futuros

Este artigo teve como objetivo relatar um estudo de caso no teste de uma aplicação web utilizando a técnica *Behavior-Driven Development* (BDD) em conjunto com os arcabouços Cucumber-JVM, Selenium WebDriver e JUnit.

Pode-se notar uma vantagem a médio e longo prazo na execução dos testes automatizados, dado que inicialmente existe um esforço para realizar as tarefas de implementação dos *scripts*. Com a análise dos fatores ambientes de teste e repetições de testes pode-se observar um impacto na decisão em utilizar ou não a automatização dos testes de aceitação. Entretanto, frente à necessidade de ter aplicações cada vez mais multiplataformas, a segurança provida pelo teste automatizado de aceitação se mostra efetiva para a empresa de software que propõe fornecer um produto de excelência, cumprir prazos estabelecidos com o cliente, reduzir esforço de teste manual, tornar o desenvolvimento do software mais participativo entre todos os envolvidos e, consequentemente, aumentar a satisfação com seus clientes e usuários finais.

Como trabalhos futuros pretende-se realizar um estudo de caso abordando a paralelização na execução dos testes por meio dos testes automatizados de aceitação utilizando a técnica BDD. Também é esperado realizar um estudo com a automatização dos testes de aceitação no contexto de Integração Contínua.

5. Referências

Apache (2002). Apache Maven Project. Página Web. Disponível em: http://maven.apache.org/. Acesso em: 03/04/2015.

Chiavegatto, R.; Silva, L.; Pinheiro, M.; Vincenzi, A. M. R. (2014) Automatização de testes funcionais em dispositivos móveis utilizando a técnica BDD -- Relato de Experiência *VIII Brazilian Workshop on Systematic and Automated Software Testing -- SAST'2014, SBC,* pp. 107-112

Domingues, A. L. S. (2002). Avaliação de critérios e ferramentas de teste para programas OO. Master's thesis, ICMC/USP, São Carlos/SP - Brasil.

Dustin, E.; Garrett, T.; Gauf, B. (2009). Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley Professional, 368 páginas.

Delamaro, M. E.; Maldonado, J. C.; Jino, M. (2007). Introdução ao Teste de Software. Elsevier, Campus, 394 páginas.

North, D. (2006). Introducing BDD. Disponível em: <http:// http://dannorth.net/introducing-bdd/>. Acessado em 28.03.2015.

Rios, E.; Moreira, T. (2013). Teste de Software, Alta Books, 3° edição, 304 páginas.

Selenium (2008). SeleniumHQ Browser Automation. Página Web. Disponível em: <http:// http://www.seleniumhq.org/ >. Acessado em 28.03.2015.

Sommerville, I. (2011). Engenharia de Software, Pearson Education, Inc., 9° edição, 773 páginas.

SonarSource (2008). SonarSource S.A. Página Web. Disponível em: <http://www.sonarqube.org/. Acesso em: 05/04/2015.

Tahchiev, P.; Leme, F.; Massol, V.; Gregory, G. (2011). JUnit in Action. Manning, 2° Edição, 467 páginas.

Wynne, M.; Hellesoy, A. (2012). The Cucumber Book: Behaviour-Driven Development for Testers and Developers. The Pragmatic Programmers, 313 páginas.