Desenvolvimento Orientado a Comportamento com Testes Automatizados utilizando JBehave e Selenium

Rafael B. Chiavegatto[1], Lidiane V. da Silva[2], Andréia Vieira[2], William R. Malvezzi[1]

[1]Centro de Pós-Graduação e Extensão (CPGE) – Faculdade FUCAPI – Manaus – AM – Brasil

[2]Fundação Desembargador Paulo dos Anjos Feitoza (FPF) – Manaus – AM – Brasil

{rafael.chiavegatto, lidiane.silva, aviera}@fpf.br,
william.malvezzi@fucapi.br

Abstract. *With a view to the continuous improvement of the software development process, this paper presents the application of the technique of test automation using frameworks to optimize the execution of the process activities, implementing the methodology Behavior Driven Development (BDD), aiming at the common language, participatory collaboration between those involved in the process of developing quality software.*

Resumo. *Tendo em vista a melhoria contínua do processo de desenvolvimento de software, este artigo apresenta a aplicação da técnica de automatização de testes utilizando frameworks para otimizar as atividades de execução do processo, implementando a metodologia Behavior Driven Development* (*BDD*), *visando à colaboração participativa em linguagem comum, entre os envolvidos no processo de desenvolvimento de software qualitativo.*

1. Introdução

A partir do final dos anos 90 do século passado, as empresas que desenvolvem *software* vêm buscando a garantia da qualidade dos sistemas desenvolvidos, por meio de estudos sobre técnicas e metodologias a fim de aplicá-las para melhoria do processo de desenvolvimento, objetivando a qualidade dos seus produtos.

> *[...] teste de Software é uma área que tem crescido significativamente nos últimos tempos, em especial a técnica de automatização de testes. Esta abordagem está cada vez mais em evidência devido à agilidade, eficácia, baixo custo de implementação e manutenção, funcionando como um bom mecanismo para controlar a qualidade dos sistemas [...]. (BERNARDO, 2011).*

Todavia, é necessário o mínimo de conhecimento sobre a área para evitar falhas comuns no processo de desenvolvimento das atividades, como erros na escrita dos *scripts* dificultando sua manutenibilidade. Quando os testes automatizados são de baixa qualidade, tendem a não contribuir para o controle da qualidade de um sistema e geram uma demanda maior de esforço na fase de testes.

Com intuito de suprir essas necessidades, este artigo apresenta a adoção da técnica de automatização no processo de teste de *software* (criação, manutenção e

gerenciamento dos casos de testes automatizados), utilizando a metodologia *Behavior Driven Development* (BDD), fazendo com que cada um dos envolvidoss contribua para melhoria contínua da qualidade, com a finalidade de tornar o processo de teste mais eficiente e produtivo.

2. Referencial teórico

2.1. Teste e Qualidade de Software

No processo de desenvolvimento de *software*, testar deve tornar-se um hábito, de forma a garantir a qualidade dos produtos desenvolvidos. A atividade está ligada diretamente à qualidade, pois, para obtenção de um *software* de que atenda as expectativas do cliente, é necessário que seja realizado um conjunto de testes específicos.

Schwaber e Beedle afirmam que

> *"metodologias de desenvolvimento ágeis como Scrum, recomendam que todas as pessoas envolvidas em um projeto trabalhem controlando a qualidade do produto todos os dias e a qualquer momento, pois baseiam-se na ideia de que prevenir defeitos é mais fácil e barato que identificá-los e corrigi-los a posteriori." (apud BERNARDO, 2001).*

Inthurn (2001) ainda complementa, "teste de *software* tem como objetivo aprimorar a produtividade e fornecer evidências de confiabilidade, em complemento a outras atividades de garantia de qualidade ao longo do processo de desenvolvimento do *software.*"

Dessa forma, testar é um processo de repetição contínua de passos para analisar se os requisitos e as condições funcionais e não funcionais especificadas, foram implementadas de fato, bem como detectar erros e identificar falhas. Sommerville (2003) afirma que, "[...] os testes constituem uma fase dispendiosa e trabalhosa do processo [...]". A execução manual de um caso de teste é rápida e efetiva, mas a execução e repetição de um vasto conjunto de testes é uma tarefa árdua e cansativa.

Dado esse cenário, utilizar meios automáticos para execução desta etapa de desenvolvimento pode agregar ganho de tempo para a Organização. Devido a tal fato, a automatização de testes vem sendo utilizada como uma forma de evitar esse problema, com a substituição parcial dos testes manuais, diminuindo os custos de produção do *software* através da agilidade que os testes automatizados proporcionam.

2.2. Automatização de testes

A introdução de testes automatizados no processo de desenvolvimento remete a iniciativa de melhoria da qualidade dos *software*. A automatização de testes possui o objetivo de apoiar o processo, reduzindo ou eliminando gargalos no tempo de execução, entretanto esse processo vai além da escolha de uma ferramenta.

Segundo Maldonado *et al* (2007), "A técnica de automação é voltada principalmente para melhoria da qualidade, baseia-se fortemente na teoria de Teste de *software*, para aplicar as recomendações dos testes manuais na automação dos testes."

Os testes automatizados tendem a ter uma maior produtividade, onde é necessário um esforço inicial que é compensando na execução a longo prazo. "O teste

manual não pode ser eliminado, deve sim, ser reduzido ao máximo possível e focado naquilo que é muito caro automatizar.” (MOLINARI, 2010).

A automatização de testes é uma prática ágil e eficaz para melhorar a qualidade dos *software*, porém inicialmente a adoção da técnica é mais custosa, pois, demanda um esforço maior de tempo e de mão de obra qualificada. É necessário conhecimento, organização e experiência para evitar que a aplicação dessa técnica não seja utilizada de forma incorreta, e para que não haja redução no custo-benefício dessa prática no desenvolvimento de *software*.

2.3. Behaviour Driven Development

Behaviour Driven Development (BDD) é uma das técnicas ágeis para desenvolvimento de *software* que estimula a colaboração entre os participantes de um projeto, trabalhando a comunicação de forma ubíqua, a fim de que os envolvidos falem a mesma linguagem e consigam compreender e contribuir para que os requisitos constantemente estejam atualizados, especificando estórias referentes às funcionalidades que futuramente poderão ser executadas.

É considerado que BDD é um aprimoramento de *Test Driven Development* (TDD). A principal diferença é a mudança de foco: de teste para comportamento. “Ao invés de escrever testes para verificar se um método faz o esperado, o desenvolvedor escreve especificações descrevendo o comportamento que a funcionalidade deve possuir.” (ASTELS, 2006).

O BDD visa minimizar a falha de comunicação entre os *stakeholders* envolvidos no projeto (cliente e equipe de desenvolvimento), uma vez que utilizando uma linguagem livre de termos típicos de teste. Os termos são comuns e unificados, propondo a descrição e atualização dos requisitos.

Há uma gama de ferramentas para apoiar o processo de BDD, para diversas linguagens de programação, todas seguindo a mesma abordagem. Dentre elas estão RSpec, Cucumber, JDave, BDoc e JBehave, que será utilizada nesse artigo.

2.4. JBehave

O JBehave é um *framework* de desenvolvimento orientado a comportamento, que permite criar especificações em linguagem natural ou seja, é possível escrever cenários exemplificando cada funcionalidade de modo que os envolvidos compreendam os critérios de aceitação dos cenários de teste, focando no comportamento do *software*.

Baseia-se em um arquivo texto projetado para conter os cenários de uma estória de um determinado usuário. Essas estórias são descritas em formato de passos detalhados, de forma que um usuário que não detenha conhecimento e perfil técnico sobre *software* possa compreender, conforme Figura 1.

```
Narrativa: Como um usuário
           Desejo acessar o site Gmail no site www.gmail.com
           E então acessar a minha caixa de e-mail.

Cenário: Acessar o site e efetuar login com o usuário e senha corretos

Dado que estou no site www.gmail.com
Quando informo o login rafael e senha 123456
Então a caixa de e-mail é exibida
```

Figura 1. Exemplo de Especificação de cenário no JBehave

Os cenários são compostos pelas seguintes notações:

- Narrativa: Descrição da estória
- Cenário: Descrição do cenário para estória
- Dado que (*Given*) – São as pré-condições para executar o cenário
- Quando (*When*) – São os testes (passos) para execução do cenário
- Então (*Then*) – É o resultado esperado da execução dos passos.

"O JBehave em si é um *plugin* do JUnit (*framework* de testes de unidade) e as especificações criadas para os passos devem ser mapeadas para testes de unidade reais." (SANTOS, 2010). Em nível de especificação de cenários, o mapeamento serve para parametrização de passos, o que facilita na reutilização para a criação de novos arquivos de texto ou outros cenários de teste.

A especificação é salva em um arquivo com extensão *.story*, que posteriormente será interpretado pelo JBehave seguindo o mapeamento realizado. O mapeamento instrui como o JBehave deve ler o arquivo de especificação e como criar um teste a partir dele.

Para cada arquivo de cenários (*.story*), deve existir um arquivo .java contendo o respectivo mapeamento. Para ser elaborado é necessário à utilização das anotações: *@Given*, *@When* e *@Then*. As palavras chaves parametrizadas iniciadas com o símbolo: $, significam que será passado por parâmetro o valor (destacados em vermelho no arquivo *.story* da Figura 1) para o método correspondente no arquivo .java, conforme mostra Figura 2:

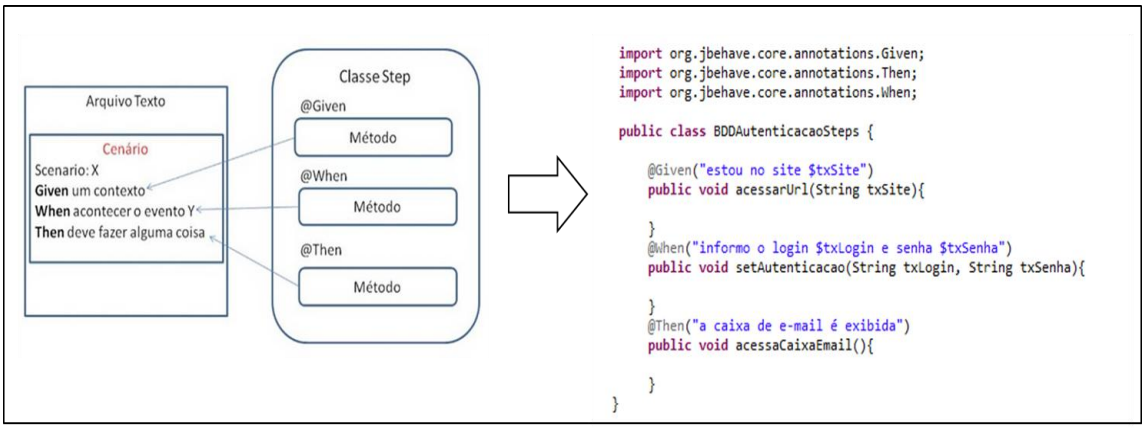


Figura 2. Classe .java contendo o mapeamento dos cenários no JBehave

Depois de realizado o mapeamento é possível utilizar um *framework* de testes para a execução dos cenários, nesse caso será utilizado o JUnit. O JBehave possui uma

classe própria para essa associação, bastando criar uma classe e estendê-la à classe *JUnitStory*. Ao executá-la como *JUnit Test* serão executados os testes dos cenários elaborados.

A classe de configuração possui: o idioma que deve ser interpretado os cenários, a localização de onde estão os cenários especificados e o formato que será gerado os relatórios, conforme exibido na Figura 3.

```
import java.util.List;

public class ClasseTesteConfiguracao extends JUnitStory{

    @Override
        public List<CandidateSteps> candidateSteps() {
            try {
                List<CandidateSteps> candidateSteps = new InstanceStepsFactory( configuration(),
                        new ClasseTesteSteps()).createCandidateSteps();
                return candidateSteps;
            } catch ( Exception e ) {
                throw new RuntimeException( e );
            }
        }
        @Override
        public Configuration configuration() {
            //A linguagem que será utilizada no .story
            Keywords keywords = new LocalizedKeywords( new Locale("pt", "BR") );
            return new MostUsefulConfiguration()
            .useKeywords( keywords )
            .useStoryParser(new RegexStoryParser(keywords))
            // Onde procurar pelas estórias
            .useStoryLoader( new LoadFromClasspath( this.getClass() ) )
            // Para onde fazer os reports
            .useStoryReporterBuilder(
                    new StoryReporterBuilder()
                    .withFormats( Format.HTML, Format.CONSOLE )
                    .withDefaultFormats()
                    .withFailureTrace( true )
                    );
        }
}
```

Figura 3. Exemplo de arquivo de configuração do JBehave

O JBehave é um *framework* muito utilizado pela comunidade Java, sua documentação é bem vasta e de fácil entendimento. Muitos *blogs* e fóruns de discussões fornecem informações sobre sua utilização e configuração.

2.5. Selenium

Conforme visto, as funcionalidades do sistema são representadas por estórias e dessas são extraídos os testes de aceitação, que são automatizados por um *framework* de automatização de testes.

O Selenium é um *framework open source*, utilizado para automatização de testes funcionais em aplicações *web*. "[...] os testes podem ser executados em praticamente todos os navegadores existentes e uma vantagem é que eles podem ser escritos em diversas linguagens de programação, tais como: Java, C#, Python, PHP, Perl, HTML e Ruby." (Gonçalves, 2011).

2.6. JUnit

O JUnit é um *framework open source* para utilização de testes unitários em Java, sendo possível criar classes de testes que podem ter um ou mais métodos para execução. Os testes podem ser executados sequencialmente ou de forma modularizada, dessa forma, os sistemas podem ser testados em partes ou de uma única vez.

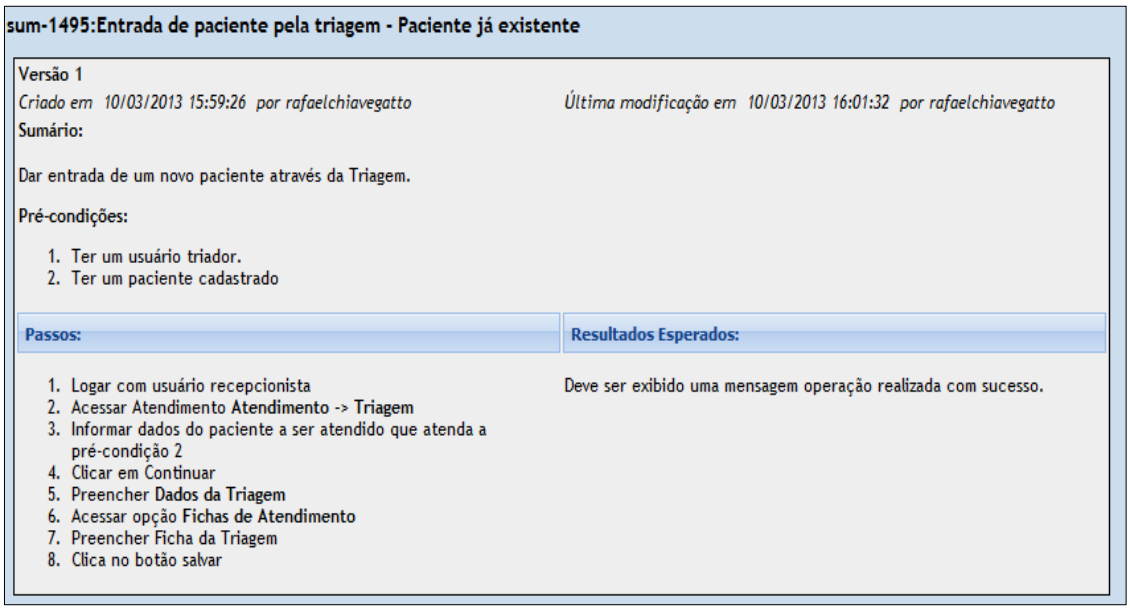
Conforme Hunt e Thomas (2003), “Em 1998, Kent Beck e Erich Gamma desenvolveram o arcabouço JUnit para a linguagem Java, inspirado no SUnit.” Desde então a prática de testes automatizados vem sendo disseminada dentre as equipes de desenvolvimento de *software* adeptas a automatização.

De forma geral seu funcionamento se dá através da verificação dos métodos de uma classe, constatando se os mesmos funcionam da maneira esperada, exibindo de forma visual o resultado da execução positiva ou negativa dos testes (*Passed or Fail*).

3. Estudo de Caso: Utilização de BDD com frameworks para automatização de testes

Nesse estudo de caso será utilizado um sistema para soluções hospitalares composto de nove módulos. O mesmo encontra-se em produção, porém continua sofrendo alterações em suas funcionalidades e constantemente é necessário passar por testes de regressão. Entretanto, por se tratar de um sistema grande e complexo, o esforço com esses testes tem sido custoso demandando esforço e tempo da equipe.

O cenário atual dos testes do sistema ocorre através de execução manual. São elaborados casos de teste funcionais, baseados nas especificações dos requisitos, conforme Figura 4.



sum-1495:Entrada de paciente pela triagem - Paciente já existente

Versão 1

Criado em 10/03/2013 15:59:26 por rafaelchiavegatto — *Última modificação em 10/03/2013 16:01:32 por rafaelchiavegatto*

Sumário:

Dar entrada de um novo paciente através da Triagem.

Pré-condições:

1. Ter um usuário triador.
2. Ter um paciente cadastrado

Passos:	Resultados Esperados:
1. Logar com usuário recepcionista 2. Acessar Atendimento Atendimento -> Triagem 3. Informar dados do paciente a ser atendido que atenda a pré-condição 2 4. Clicar em Continuar 5. Preencher Dados da Triagem 6. Acessar opção Fichas de Atendimento 7. Preencher Ficha da Triagem 8. Clica no botão salvar	Deve ser exibido uma mensagem operação realizada com sucesso.

Figura 4. Especificação de Caso de teste

A especificação dos casos de testes, não fornece todas as informações detalhadas para execução, pois o responsável pela elaboração já possui conhecimento prévio sobre a aplicação. Isso poderia dificultar a execução se, o executor dos testes não for o mesmo que elaborou.

Essa problemática pode ser resolvida com a utilização da metodologia BDD, pois sua especificação é voltada a comportamento, sendo exemplificados detalhadamente todos os passos para execução do cenário, como mostra a Figura 5.