ESTUDO DE CASOS: UTILIZANDO O CUCUMBER PARA AUXILIAR O BDD

CASE STUDY: USING CUCUMBER TO HELP BDD

BRUNO BARBOSA

Graduando em Ciência da Computação

Bruno.fernandes845@hotmail.com

LORENA SALAZAR

Graduando em Ciência da Computação

lorena.evangelista.25@gmail.com

LUCAS GIACOMIN

Graduando em Ciência da Computação

lucascgiacomin16@gmail.com

MATHEUS BARBOSA

Graduando em Sistemas de Informação

mdpb.matheus@gmail.com

MATHEUS MARMO

Graduando em Ciência da Computação

marmo1001@hotmail.com

RENAN SANTANA

Graduando em Ciência da Computação

renanblog15@gmail.com

RESUMO

Testes de software possuem a função primordial de assegurar que os produtos de softwares desenvolvidos estão de acordo com os padrões de negócio levantados e possuem um padrão de qualidade em todos os seus processos. Para um melhor uso do tempo de desenvolvimento e de esforços, utilizar uma ferramenta automatizada se faz extremamente necessário, pois permite a execução de múltiplos testes de forma simultânea o que diminui o tempo de busca por erros, padronização dos documentos, e aumento expressivo de produtividade. Tendo isso em vista isso, o Cucumber é uma ferramenta que busca automatizar a execução de testes baseada primariamente no método BDD (*Behavior Driven Development* ou Desenvolvimento Orientado a Comportamento) utilizando-se da abordagem Gherkin que permite uma maior facilidade no entendimento dos cenários de teste, é *open-source* e possui versões para múltiplas plataformas e linguagens, sendo apresentado aqui um estudo de caso com a linguagem Java.

Palavras chaves: Testes de software, Automação de testes, BDD, Gherkin.

ABSTRACT

Software testing has the primary function of ensuring that the software products developed are in accordance with the business standards and have a standard of quality in all its processes. For a better use of the development time and efforts, using an automated tool is extremely necessary, since it allows the execution of multiple tests simultaneously, which reduces the time of search for errors, standardization of documents, and impressive increase of productivity. With this in mind, Cucumber is a tool that seeks to automate the execution of tests based primarily on the BDD method (*Behavior Driven Development*) using the Gherkin approach that allows an easier understanding of the test scenarios, is open-source and has versions for multiple platforms and languages, being presented here a case study with the Java language.

Key words: Software testing, Test automation, BDD, Gherkin.

INTRODUÇÃO

O interesse pela aplicação *Cucumber* surgiu com a busca por uma ferramenta que possua um foco colaborativo e interativo entre as diferentes partes do projeto; alia-se a isso o foco na metodologia de desenvolvimento *BDD - Behavior Driven Development;* (Desenvolvimento Orientado a Comportamento) que permite, desde o início, que se desenvolva todo um projeto calcado em testes e na busca incessante por um padrão de qualidade em todo o projeto, da documentação ao código.

Considerando que o mercado de ferramentas de teste comumente oferece somente versões pagas e de código privado, o *Cucumber* se diferencia pois permite a avaliação do código por ser *open-source*, e possuir uma versão gratuita da aplicação permitindo que todos os desenvolvedores possam experimentar e desenvolver seus projetos com o auxílio de uma ferramenta de auxílio a testes; caso o projeto necessite, pode-se utilizar uma versão paga da aplicação. Analisaremos aqui a versão gratuita da aplicação *Cucumber*.

Este artigo busca apresentar um estudo de caso abordando o processo de testes na aplicação *Cucumber*, descobrindo as possibilidades de uso da aplicação e em como elas podem aumentar a produtividade dos desenvolvedores, a confiabilidade do projeto e a qualidade geral dos produtos de software e documentações entregues ao final do processo de desenvolvimento.

Considerando que a maior parte dos produtos e soluções baseados em software são realizados por múltiplos profissionais, o uso de uma ferramenta que permita que todos os profissionais envolvidos mantenham um mínimo padrão e conversa sobre o desenvolvimento sobre o que está sendo desenvolvido se faz essencial, pois permite que a automatização e a referência a um padrão estabelecido durante a fase inicial de concepção do projeto, permita que os indivíduos desenvolvam os produtos e documentações com base nele, e que mantenham a coerência durante todo o processo, visando sempre e de forma permanente a máxima qualidade.

REFERENCIAL TEÓRICO

Teste de software: Garantir que um software está funcionando exatamente como o especificado nos requisitos e detectar erros durante o desenvolvimento de uma aplicação antes dela estar em ambiente de produção, é uma finalidade dos testes de software. Esse processo está ligado a dois termos conhecidos como Verificação e Validação, onde para Morlinari (2008, p. 96) "Verificação é o processo de confirmação de que algo (o software) vai ao encontro das especificações. Validação é o processo de confirmação de que o software vai ao encontro dos requerimentos do usuário."

Qualidade de software: Durante o desenvolvimento de um software, todas as decisões tomadas pela equipe podem comprometer a qualidade final do produto. Desta forma, todas as ações tomadas no ciclo de desenvolvimento irão afetar o produto. Para garantir as reais especificações do produto e necessário atribuir um esforço em qualidade em todo o processo de desenvolvimento.

Uma definição abrangente sobre os conceitos de qualidade de software foi definida por BARTIÉ (2002, p. 16) "Qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos".

Automação: Testes automatizados são utilizados para evitar o trabalho manual excessivo em etapas que necessitam a execução de testes de regressão, sendo executado rapidamente sempre que necessário, contribuindo para a qualidade do software.

Teste de regressão: Durante o desenvolvimento do software, é comum termos situações em que ou à inclusão de uma nova funcionalidade pelo cliente ou encontrar um erro na lógica do código fonte. Independentemente do cenário, o desenvolvedor terá de fazer a alteração na programação. Em alguns casos, uma simples mudança pode comprometer toda a lógica já escrita, invalidando quaisquer testes básicos feitos no processo de desenvolvimento.

"Alterações na versão do software podem influenciar nos formatos das entradas e saídas e os casos de testes podem não ser executáveis sem as alterações correspondentes “YOUNG e PEZZÈ (2008, p. 454), sendo assim é necessário reexecutar estes testes com intuito de garantir que as demais funcionalidades ou partes do software já desenvolvidas estejam funcionando corretamente. Segundo Roger Pressman (2016, p. 478) "o teste de regressão é a reexecução do mesmo subconjunto de testes que já foram executados, para assegurar que as alterações não tenham propagado efeitos colaterais indesejados."

Gherkin: O Gherkin é um elemento essencial quando se tem BDD (Behavior-Driven Development) na automatização de testes, com a função de padronizar a forma de descrever os senários de teste, baseado nas regras de negócio, ele permite deixar os testes automatizados muitos mais fáceis de se ler, mesmo para uma pessoa leiga.

BDD: O BDD (Desenvolvimento Orientado por Comportamento) é uma técnica semelhante ao TDD em vários aspectos. O BDD muda o foco dos testes de implementação para os comportamentos que o sistema expõe, ele é focado na colaboração entre desenvolvedores, analistas de negócios ou mesmo pessoas que não fazem parte da área técnica, visando integrar regras de negócios com linguagem de programação, focando no comportamento do software.

Teste de Segurança: O Teste de Segurança tem como objetivo garantir que o funcionamento da aplicação esteja exatamente como especificado. Verifica também se o software se comporta adequadamente mediante as mais diversas tentativas ilegais de acesso, visando possíveis vulnerabilidades. Para isso, testa se todos os mecanismos de proteção embutidos na aplicação de fato a protegerão de acessos indevidos.

Teste de Caixa-preta: O teste de caixa-preta é baseado na entrada e saída de dados de acordo com uso do cliente final, com o objetivo de verificar se a aplicação está se comportando exatamente como a especificação. Geralmente são criados e executados por analistas de teste ou por clientes. São casos de teste que requerem um conhecimento do funcionamento interno do sistema.

Debug: Processo de encontrar erros que podem impedir que os códigos funcionem adequadamente. É possível determinar o que está ocorrendo dentro do código-fonte e obter sugestões de ações de melhorias. Através das ferramentas de depuração de código é possível inspecionar internamente o código-fonte durante a execução da aplicação. Economizando tempo localizando os Bugs da aplicação com mais rapidez e evitando refeitos em grandes projetos.

Sistemas de Controle de Versão (GIT): O gerenciamento de versão corresponde a atividades e processos para o controle da evolução de diferentes de componentes ou itens de configuração utilizados durante a manutenção de software (SOMMERVILLE et al, 2011).

Dentre as ferramentas mais conhecidas para o gerenciamento de versão está o GIT. Este tem propriedades bastante interessantes que o levam a ser popular. O GIT considera que os dados são como um conjunto de snapshots (captura de algo em um determinado instante, como em uma foto) de um minissistema de arquivos (CHACON E STRAUB,2014), dessa forma, o *git* armazena referências sobre os snapshots. Assim, o sistema de controle de versão não armazena arquivos redundantes, apenas as referências de alterações sobre um arquivo principal.

MATERIAL E MÉTODO (ou METODOLOGIA)

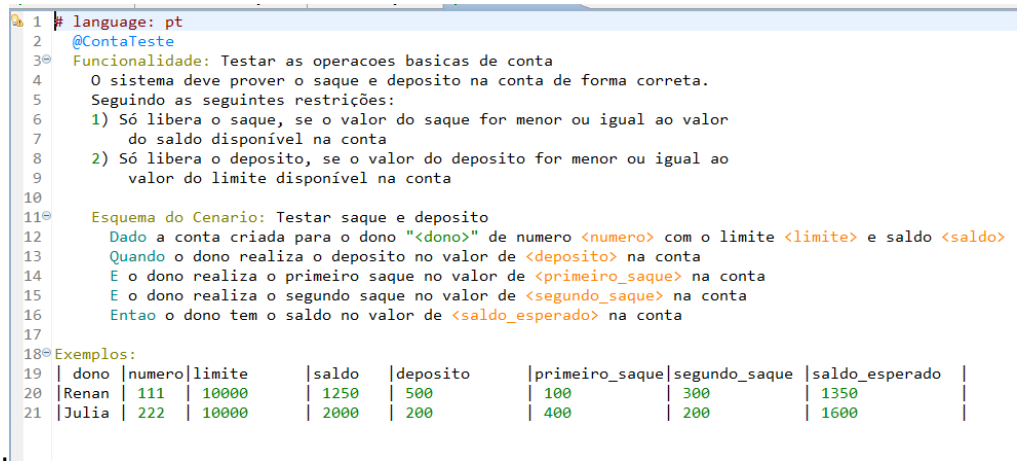
O Cucumber é um software que usa da técnica BDD para gerar os testes. O BDD consiste na identificação do objetivo de negócio. Para a nossa explicação usaremos "Negociação bancária" como objetivo, que contém um banco e conta bancária. Faremos o desenvolvimento de testes de aceitação de duas funcionalidades utilizando o Cucumber em Java.

A primeira funcionalidade vai possibilitar que o usuário realize as operações de fazer saque e deposito utilizando sua conta. Essas operações deveram seguir algumas restrições, que serão.

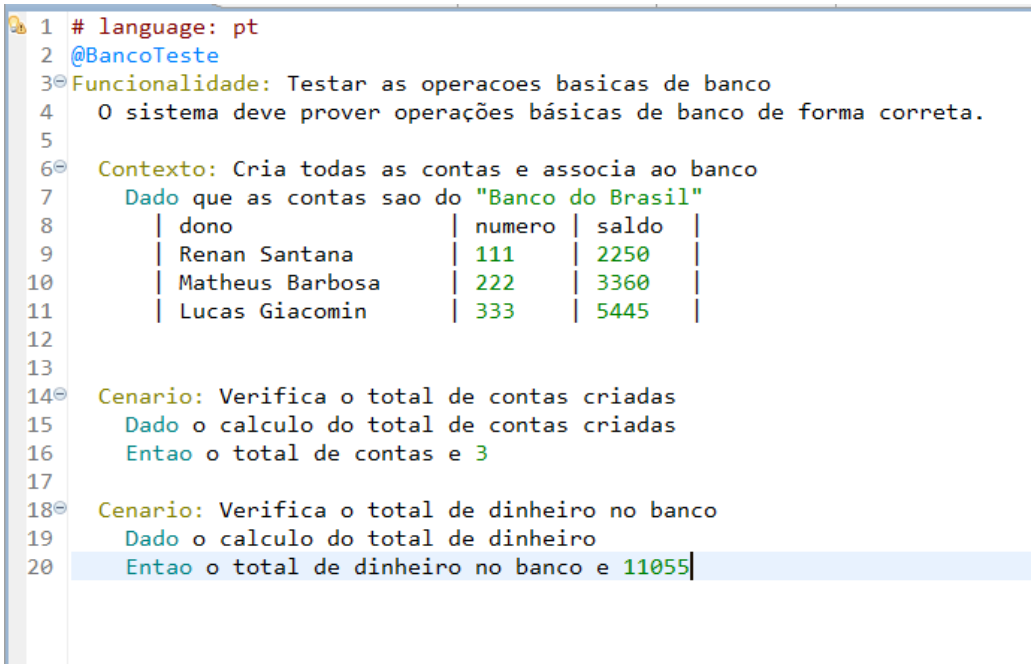
O sistema só libera o saque só o valor deste for menor ou igual ao valor do saldo disponível na conta, e o sistema só libera o deposito se o valor deste for menor ou igual ao valor do limite disponível na conta.

A segunda funcionalidade irá possibilitar o usuário realizar operações básicas no banco, como, obter o dinheiro total disponível no banco e obter o total de contas criadas.

Primeiro foram criados os arquivos *features*, onde é uma usado uma descrição de alto nível para relatar como nossos testes deve se comportar, usamos uma linguagem padrão para especificação de testes de aceitação, a famosa linguagem “Gherkin”, do Cucumber.



```
# language: pt
@ContaTeste
Funcionalidade: Testar as operacoes basicas de conta
  O sistema deve prover o saque e deposito na conta de forma correta.
  Seguindo as seguintes restrições:
  1) Só libera o saque, se o valor do saque for menor ou igual ao valor
      do saldo disponível na conta
  2) Só libera o deposito, se o valor do deposito for menor ou igual ao
      valor do limite disponível na conta

  Esquema do Cenario: Testar saque e deposito
    Dado a conta criada para o dono "<dono>" de numero <numero> com o limite <limite> e saldo <saldo>
    Quando o dono realiza o deposito no valor de <deposito> na conta
    E o dono realiza o primeiro saque no valor de <primeiro_saque> na conta
    E o dono realiza o segundo saque no valor de <segundo_saque> na conta
    Entao o dono tem o saldo no valor de <saldo_esperado> na conta

Exemplos:
| dono |numero|limite      |saldo   |deposito      |primeiro_saque|segundo_saque |saldo_esperado  |
|Renan | 111  | 10000      | 1250   | 500          | 100          | 300          | 1350           |
|Julia | 222  | 10000      | 2000   | 200          | 400          | 200          | 1600           |
```



```
# language: pt
@BancoTeste
Funcionalidade: Testar as operacoes basicas de banco
  O sistema deve prover operações básicas de banco de forma correta.

  Contexto: Cria todas as contas e associa ao banco
    Dado que as contas sao do "Banco do Brasil"
      | dono                | numero | saldo  |
      | Renan Santana       | 111    | 2250   |
      | Matheus Barbosa     | 222    | 3360   |
      | Lucas Giacomin      | 333    | 5445   |


  Cenario: Verifica o total de contas criadas
    Dado o calculo do total de contas criadas
    Entao o total de contas e 3

  Cenario: Verifica o total de dinheiro no banco
    Dado o calculo do total de dinheiro
    Entao o total de dinheiro no banco e 11055
```

Existe uma anotação chamada @*RunWith*(Cucumber.class), isso diz ao JUnit que o Cucumber irá assumir o controle da execução dos testes nesta classe. Outra anotação definida na classe é a @CucumberOptions, onde podemos definir parâmetros customizáveis utilizados pelo Cucumber na execução dos testes.

testes-cucumber ▸ src/test/java ▸ cucumber.teste ▸ BancoTeste

```
1  package cucumber.teste;
2
3  import org.junit.runner.RunWith;
4  import cucumber.api.CucumberOptions;
5  import cucumber.api.junit.Cucumber;
6
7  @RunWith(Cucumber.class)
8  @CucumberOptions(features = "classpath:caracteristicas", tags = "@BancoTeste",
9                   glue = "cucumber.teste.passos", monochrome = true, dryRun = false)
10
11 public class BancoTeste {
12
13 }
14
```

testes-cucumber ▸ src/test/java ▸ cucumber.teste ▸ ContaTeste

```
1  package cucumber.teste;
2
3  import org.junit.runner.RunWith;
6
7  @RunWith(Cucumber.class)
8  @CucumberOptions(features = "classpath:caracteristicas", tags = "@ContaTeste",
9                   glue = "cucumber.teste.passos", monochrome = true, dryRun = false)
10
11 public class ContaTeste {
12
13 }
14
```

Observe que na classe ContaTestePassos estamos utilizadas as anotações @Dado, @Quando, @E e @Entao, que correspondem ao mesmo conteúdo e as palavras-chave do Gherkin definidas nos arquivos *feature*.

testes-cucumber ▸ src/test/java ▸ cucumber.teste.passos ▸ ContaTestePassos ▸

```java
package cucumber.teste.passos;

import static org.junit.Assert.assertEquals;

public class ContaTestePassos {

    private Conta conta;

    @Dado("^a conta criada para o dono \"(.*?)\" de numero (\\d+) com o limite (\\d+) e saldo (\\d+)$")
    public void a_conta_criada_para_o_dono_de_numero_com_o_limite_e_saldo(String dono, int numero, Double limite,
            Double saldo) throws Throwable {
        conta = new Conta(dono, numero, limite, saldo);
    }

    @Quando("^o dono realiza o deposito no valor de (\\d+) na conta$")
    public void o_dono_realiza_o_deposito_no_valor_de_na_conta(Double valorDeposito) throws Throwable {
        assertTrue("O dono " + conta.getDono() + " não tem limite disponível na conta para este valor de deposito",
                conta.depositar(valorDeposito));
    }

    @E("^o dono realiza o primeiro saque no valor de (\\d+) na conta$")
    public void o_dono_realiza_o_primeiro_saque_no_valor_de_na_conta(Double valorSaque) throws Throwable {
        assertTrue("O dono " + conta.getDono() + " não tem saldo disponível na conta para este valor de saque",
                conta.sacar(valorSaque));
    }

    @E("^o dono realiza o segundo saque no valor de (\\d+) na conta$")
    public void o_dono_realiza_o_segundo_saque_no_valor_de_na_conta(Double valorSaque) throws Throwable {
        assertTrue("O dono " + conta.getDono() + " não tem saldo disponível na conta para este valor de saque",
                conta.sacar(valorSaque));
    }

    @Entao("^o dono tem o saldo no valor de (\\d+) na conta$")
    public void o_dono_tem_o_saldo_na_conta_no_valor_de(Double saldoEsperado) throws Throwable {
        assertEquals("O dono " + conta.getDono() + " está com o saldo incorreto na conta", saldoEsperado,
                conta.getSaldo());
    }
```

Writable | Smart Insert | 1 : 1 : 0

Na classe BancoTestePassos também utilizamos algumas anotações @Dado e @Entao.

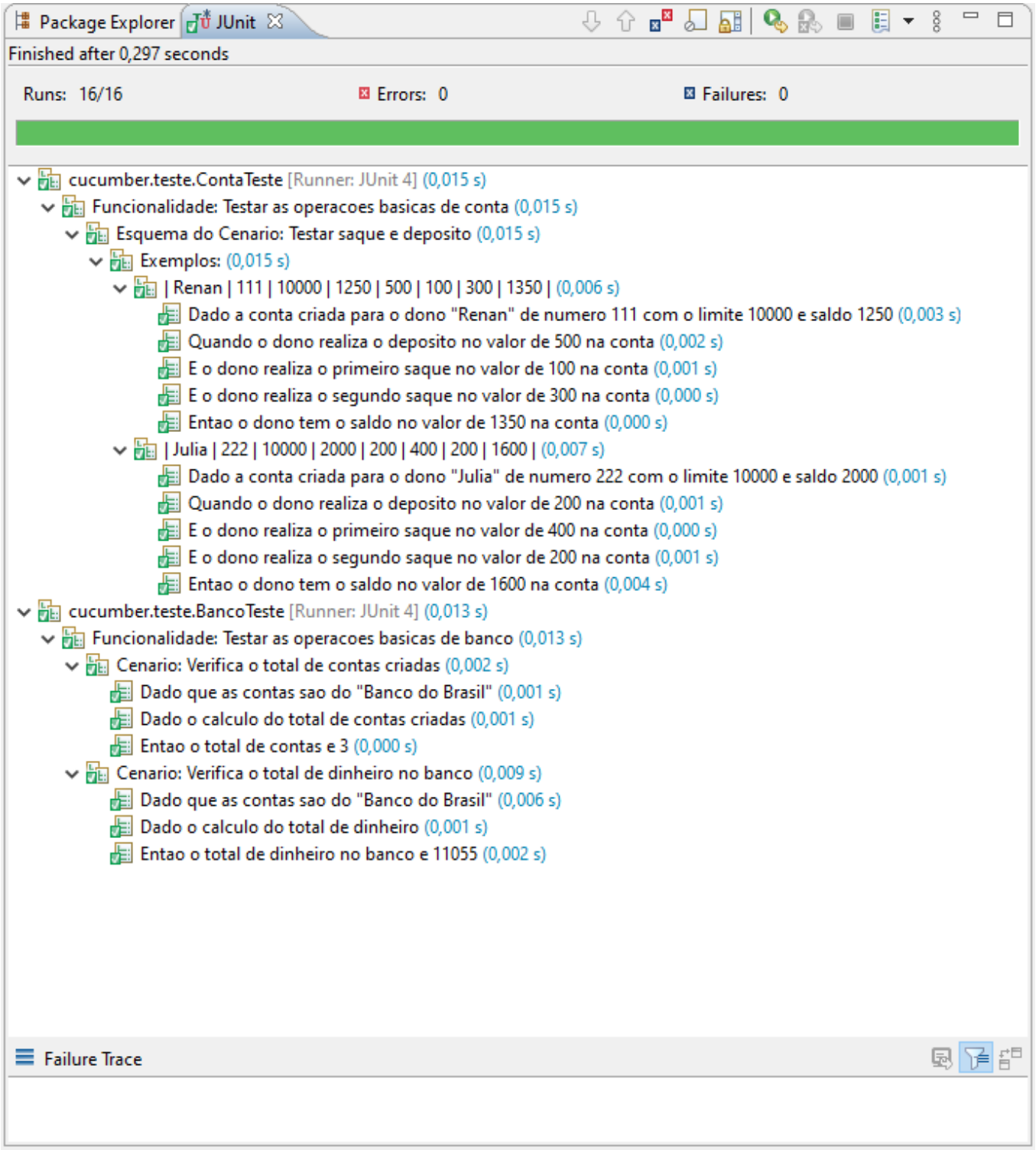
testes-cucumber ▸ src/test/java ▸ cucumber.teste.passos ▸ BancoTestePassos ▸

```java
package cucumber.teste.passos;

import cucumber.api.java.pt.Dado;

public class BancoTestePassos {

    private Banco banco;
    private int totalContas;
    private Double totalDinheiro;

    @Dado("^que as contas sao do \"(.*?)\"$")
    public void que_as_contas_sao_do(String nome, List<Conta> listaDeContas) throws Throwable {
        banco = new Banco(nome, listaDeContas);

    }

    @Dado("^o calculo do total de contas criadas$")
    public void o_calculo_do_total_de_contas_criadas() throws Throwable {
        totalContas = banco.getListaDeContas().size();
    }

    @Entao("^o total de contas e (\\d+)$")
    public void o_total_de_contas_e(int totalContasEsperado) throws Throwable {
        assertEquals("O cálculo do total de contas está incorreto", totalContasEsperado, totalContas);
    }

    @Dado("^o calculo do total de dinheiro$")
    public void o_calculo_do_total_de_dinheiro() throws Throwable {
        totalDinheiro = banco.getListaDeContas().stream().mapToDouble(c -> c.getSaldo()).sum();
    }

    @Entao("^o total de dinheiro no banco e (\\d+)$")
    public void o_total_de_dinheiro_no_banco_e(Double totalDinheiroEsperado) throws Throwable {

        assertEquals("O cálculo do total de dinheiro no banco " + banco.getNome() + " está incorreto",
                totalDinheiroEsperado, totalDinheiro);
    }
```

Writable | Smart Insert | 1 : 1 : 0

RESULTADOS E DISCUSSÃO

Os resultados saíram como esperado, cada passo executou conforme definido nos arquivos *feature*.



CONCLUSÃO

Conforme visto acima, o Cucumber possui uso simplificado, baseado primariamente em texto puro, o que permite uma clara visualização dos problemas e do passo-a-passo de todo o processo e os caminhos lógicos para a obtenção dos resultados esperados.

Por ser básico e seguir o padrão *Gherkin*, a implementação destes processos será mais fácil e inteligível para os desenvolvedores, além de posteriormente facilitar a documentação para futuros desenvolvedores, agilizando a metodologia de testes pois permite que a lógica de código seja testada de forma mais eficaz dentro dos limites estabelecidos e em concordância com as regras de negócio.

Caso seja feita no processo de idealização de certa aplicação, poderá permitir que se analise os caminhos lógicos possíveis para cada aplicação e a sua implementação antes mesmo de qualquer parte de código ser escrita, o que possibilita a agilidade do projeto e ganhos exponenciais de produtividade.

REFERÊNCIAS

ONEDAYTESTING, Gherkin: Introduzindo seus conceitos e benefícios. s.d. Disponível em https://blog.onedaytesting.com.br/gherkin/. Acesso em 26 de mai de 2022

ROQUETTE, José Henrique, Uma abordagem utilizando *Behavior Driven Development* para geração de casos de teste: um estudo de caso na área automotiva, nov. 2018, disponível em https://repositorio.utfpr.edu.br/jspui/bitstream/1/15978/1/PG_COCIC_2018_2_06.pdf. Acesso em 22 de mai. de 2022.

ANDERLE, Angelita, Introdução de BDD (*Behavior Driven Development*) como Melhoria de Processo no Desenvolvimento Ágil de Software, 2015, disponível em http://www.repositorio.jesuita.org.br/bitstream/handle/UNISINOS/5314/Angelita%20Anderle-Monografia_.pdf?sequence=1&isAllowed=y . Acesso em 22 de mai. de 2022.

Ismael, Desenvolvimento orientado por comportamento (BDD), 2011, disponível em https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127 , Acesso em 23 de mai. de 2022.

DE PAULA, Wilker Hudson, qualidade de software e desenvolvimento dirigido por comportamento - bdd: um estudo de caso, dez. 2019, disponível em https://www.monografias.ufop.br/bitstream/35400000/2433/1/MONOGRAFIA_QualidadeDeSoftware.pdf, Acesso em 27 de mai. de 2022.

HOSTGATOR, O que é debug e qual a sua importância no desenvolvimento web? mai. 2020, disponível em https://www.hostgator.com.br/blog/debug-desenvolvimento-web/, Acesso 1 de jun. de 2022.

ATLASSIAN, O que é controle de versão, s.d disponível em https://www.atlassian.com/br/git/tutorials/what-is-version-control#:~:text=Os%20sistemas%20de%20controle%20de,forma%20mais%20r%C3%A1pida%20e%20inteligente, acesso em 14 de jun. 2022.

DIAS, André Felipe, O que é gerencia de configuração de software, mai. 2016, disponível em https://blog.pronus.io/posts/controle-de-versao/o-que-eh-gerencia-de-configuracao-de-software/, acesso em jun.2022.