TÍTULO EM PORTUGUES

TÍTULO EM INGLES

Bruno Barbosa

Lorena Salazar

Lucas Giacomin

Matheus Barbosa

Matheus Marmo

Renan Santana

Graduandos em Ciência da Computação e Sistemas de Informação

RESUMO

O resumo deve ter a seguinte estrutura: parágrafo único, em português, com um número de 200 palavras. Apresentar os elementos essenciais do artigo: tipo de estudo, objetivo, resultados, conclusão ou considerações finais. Evitar primeira pessoa do singular ou do plural. Evite palavras supérfluas tais como: este artigo, este estudo, portanto, nessa perspectiva, entre outros etc. Veja exemplo abaixo como os autores apresentam os elementos essenciais do estudo.

Palavras-chave: Resumo. Artigo. Estudo.

ABSTRACT

(O *ABSTRACT* não significa que o estudante deve submeter o resumo em um tradutor online apenas. Em geral estes tradutores contêm erros, portanto, procure um profissional para fazer a revisão do abstract).

Keywords: *Abstract*. *Article*. *Study*.

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.

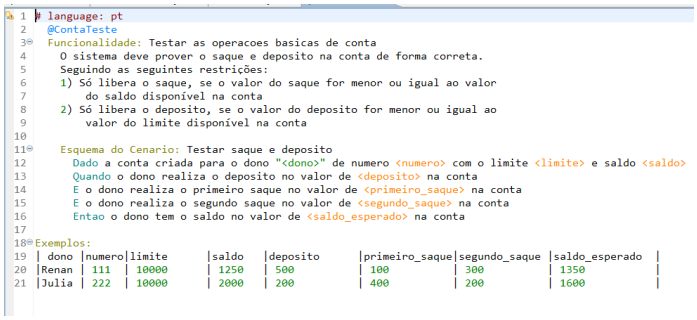
MATERIAL E MÉTODO (ou METODOLOGIA)

Para começarmos, a técnica BDD se inicia na identificação do objetivo de negócio e como exemplo tomamos como objetivo de negócio a “Negociação bancária” que contém um Banco e Conta bancária. Faremos o desenvolvimento de testes de aceitação de duas funcionalidades utilizando o framework Cucumber em Java e a técnica BDD.

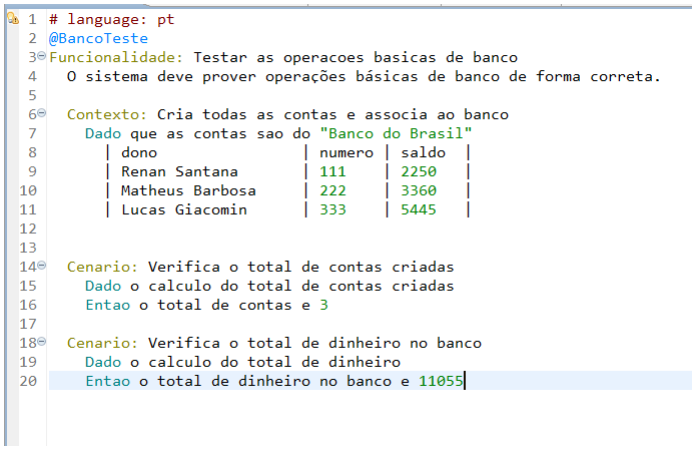
- Primeira funcionalidade: Possibilitar que o usuário realize as operações bancárias utilizando sua conta, que são:
    - Fazer saque e depósito, com as seguintes restrições:
        - Só liberar o saque se o valor deste for menor ou igual ao valor do saldo disponível na conta;
        - Só liberar o depósito se o valor deste for menor ou igual ao valor do limite disponível na conta.
- Segunda funcionalidade: Possibilitar o usuário realizar operações básicas no banco, que são:
    - Obter o total de dinheiro no banco;
    - Obter o total de contas criadas no banco.

1. 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 “Grerkin”, do Cucumber.

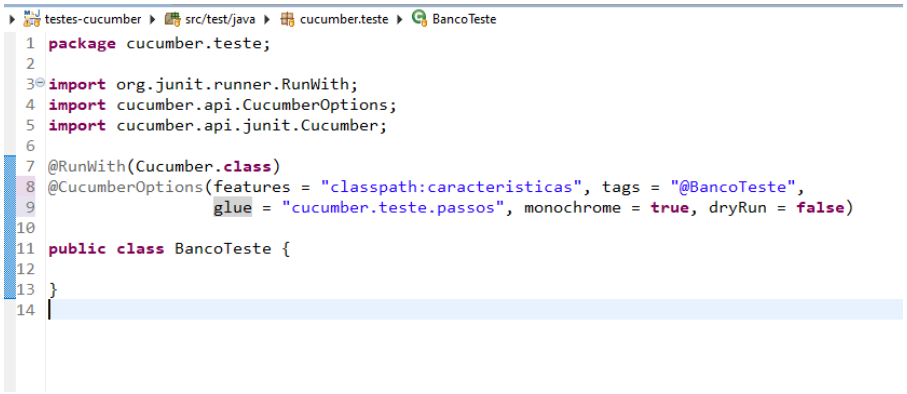


```
# language: pt
@ContaTeste
Funcionalidade: Testar as operacoes basicas de conta
  O sistema deve prover o saque e deposito na conta de forma correta.
  Seguindo as seguintes restrições:
  1) Só libera o saque, se o valor do saque for menor ou igual ao valor
      do saldo disponivel na conta
  2)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

2. 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

```
package cucumber.teste;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(features = "classpath:caracteristicas", tags = "@BancoTeste",
                 glue = "cucumber.teste.passos", monochrome = true, dryRun = false)

public class BancoTeste {

}
```

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

```java
package cucumber.teste;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(features = "classpath:caracteristicas", tags = "@ContaTeste",
                 glue = "cucumber.teste.passos", monochrome = true, dryRun = false)

public class ContaTeste {

}
```

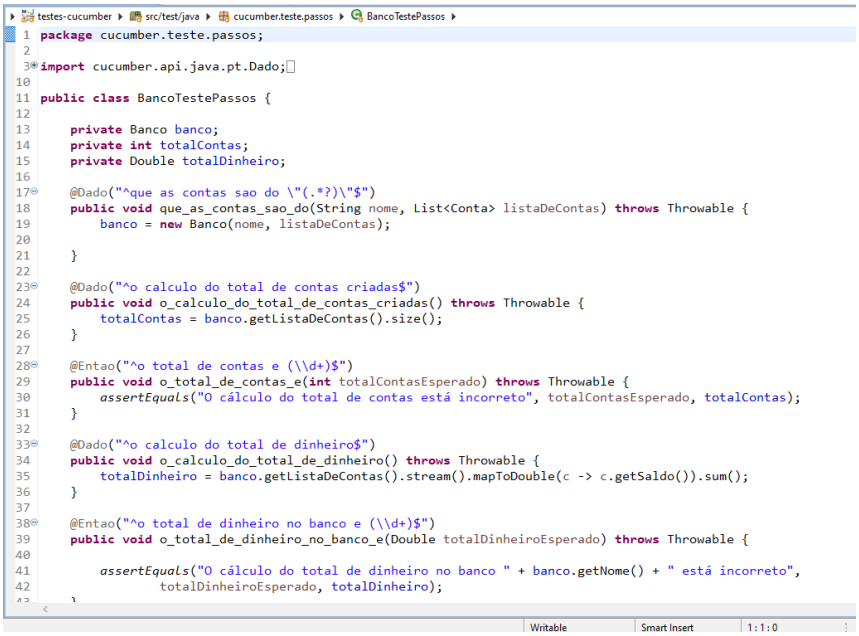
3. Observe que na classe ContaTestePassos estamos utilizadas as anotações @Dado, @Quando, @E e @Entao, que correspondem ao mesmo conteúdo e as palavras-chave do Gherkin definidas nos arquivos .feature.

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

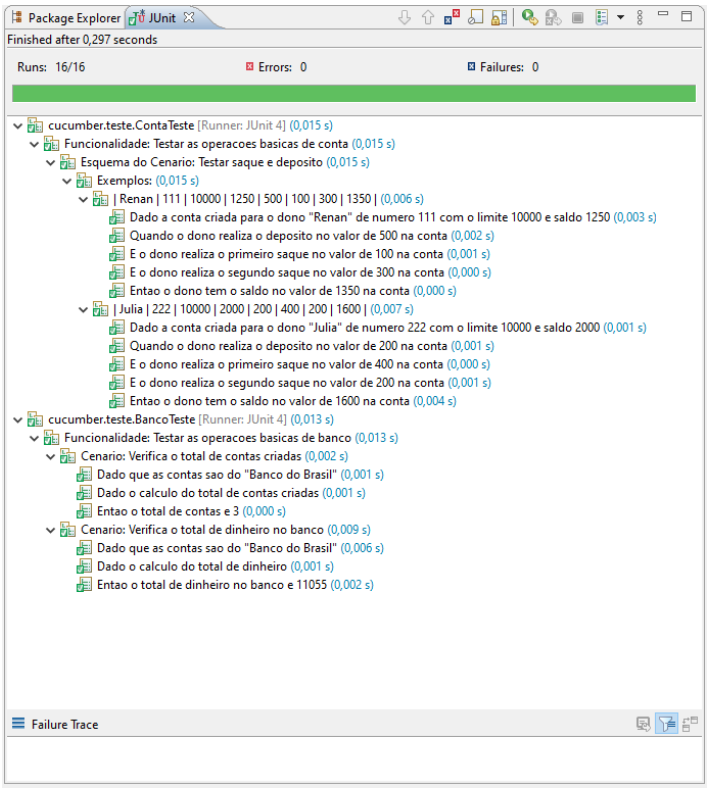
Na classe BancoTestePassos também utilizamos algumas anotações @Dado e @Entao



```
testes-cucumber ▸ src/test/java ▸ cucumber.teste.passos ▸ BancoTestePassos ▸
1  package cucumber.teste.passos;
2
3  import cucumber.api.java.pt.Dado;
10
11 public class BancoTestePassos {
12
13     private Banco banco;
14     private int totalContas;
15     private Double totalDinheiro;
16
17     @Dado("^que as contas sao do \"(.*?)\"$")
18     public void que_as_contas_sao_do(String nome, List<Conta> listaDeContas) throws Throwable {
19         banco = new Banco(nome, listaDeContas);
20
21     }
22
23     @Dado("^o calculo do total de contas criadas$")
24     public void o_calculo_do_total_de_contas_criadas() throws Throwable {
25         totalContas = banco.getListaDeContas().size();
26     }
27
28     @Entao("^o total de contas e (\\d+)$")
29     public void o_total_de_contas_e(int totalContasEsperado) throws Throwable {
30         assertEquals("O cálculo do total de contas está incorreto", totalContasEsperado, totalContas);
31     }
32
33     @Dado("^o calculo do total de dinheiro$")
34     public void o_calculo_do_total_de_dinheiro() throws Throwable {
35         totalDinheiro = banco.getListaDeContas().stream().mapToDouble(c -> c.getSaldo()).sum();
36     }
37
38     @Entao("^o total de dinheiro no banco e (\\d+)$")
39     public void o_total_de_dinheiro_no_banco_e(Double totalDinheiroEsperado) throws Throwable {
40
41         assertEquals("O cálculo do total de dinheiro no banco " + banco.getNome() + " está incorreto",
42                 totalDinheiroEsperado, totalDinheiro);
43     }
Writable | Smart Insert | 1:1:0
```

RESULTADOS E DISCUSSÃO

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



CONCLUSÃO

Nas considerações finais, você deve destacar os principais aspectos encontrados no seu estudo. Você também poderá fazer recomendações Veja exemplo abaixo.

O estudo permitiu um mapeamento da produção científica brasileira sobre o tema Expatriação no período de 2004- 2009.

A média obtida da distribuição dos 7 artigos no período analisado é de 1,75 publicações/ ano. Os anos de 2007 e 2008 não houve nenhum artigo publicado, enquanto o ano de 2009 se destaca com 3 publicações.

Quanto ao tipo de pesquisa, 71,43% foram realizados estudo de campo enquanto 28,57% têm como base estudo teóricos.

Nos últimos anos, o tema ganha destaque em Recursos Humanos, pois a tendência em um mundo de empresas globalizadas é que estas também tenham profissionais cada vez mais globalmente móveis.

O foco encontrado das pesquisas foi do impacto da expatriação para os expatriados e organizações, assim como a análise de correntes teóricas que permitem compreender sobre as expatriações.

Observa-se com os artigos analisados que ainda há muito a ser explorado nesta temática, tais como: análise de expatriação nos níveis operacional e técnico, o papel de RH nos processos de expatriação para o expatriado e organização, o multiculturalismo em empresas multinacionais, o processo de repatriação, assim como a diversidade cultural no ambiente de trabalho.

REFERÊNCIAS

https://blog.onedaytesting.com.br/gherkin/

https://repositorio.utfpr.edu.br/jspui/bitstream/1/15978/1/PG_COCIC_2018_2_06.pdf

http://www.repositorio.jesuita.org.br/bitstream/handle/UNISINOS/5314/Angelita%20Anderle-Monografia_.pdf?sequence=1&isAllowed=y

https://www.devmedia.com.br/desenvolvimento-orientado-por-comportamento-bdd/21127

https://www.monografias.ufop.br/bitstream/35400000/2433/1/MONOGRAFIA_QualidadeDeSoftware.pdf

https://www.hostgator.com.br/blog/debug-desenvolvimento-web/

O estudante deve seguir as normas do manual da FAESA disponível no AVA.

OBS: *Em todo o artigo o texto será escrito em fonte Arial ou Times New Roman.
*****O artigo deverá conter no Mínimo 15 e no Máximo 25 páginas até referências.***
******Casos omissos consultar Manual de Normas da Faesa.***