INF321 – Verificação e Validação de Software
2º Laboratório Prático (LAB02)

01/10/2016

Objetivo

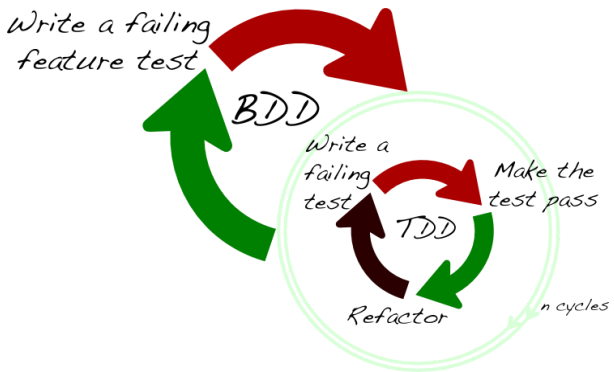
Aplicar BDD (*Behaviour Driven Development*) para desenvolver as funcionalidades previstas em casos de uso do aplicativo Bookstore utilizando mocks e stubs (dublês) para eliminar a dependência de sistemas externos nos testes e garantindo cobertura de arestas ("branches") de pelo menos 75% com os testes de caixa preta.

TAREFA

1. Aplique BDD para o desenvolvimento dos casos de uso **UC14, UC15** e **UC16** do sistema Bookstore localizados no moodle no arquivo **Requisitos_LAB02_INF321_Modelo.pdf**.

   No BDD, primeiro você cria especificações executáveis (feature) descrevendo os cenários (contendo a descrição dos passos) baseadas no comportamento da aplicação em uma linguagem natural chamada Gherkin. Depois disso, você implementa o código do seu teste automatizado (step definitions), e apenas depois destes dois passos, implementa-se o código do sistema. Cada cenário da especificação executável representa um teste.

   Enquanto não há código ou este está incompleto, seus testes automatizados falham (**vermelho**); na medida em que você constrói o código que implementa o comportamento previsto, parte dos testes começam a passar (**verde**); esse processo se repete até que todos os testes passem.



2. Os casos de uso **UC14, UC15 e UC16** especificam a integração do Bookstore com o sistema dos correios. Implemente o código necessário para fazer estas integrações de acordo com o especificado.
3. Seus testes de caixa preta devem garantir que o código criado:

    a) implementa os comportamentos que estão previstos na especificação.
    b) funciona corretamente com o serviço de preço e prazo, busca de cep e rastreio de objetos dos Correios, tanto nas situações em que este retorna sucesso quanto em todos os cenários com retorno de falha.
    c) invoca de forma adequada o serviço provido pelos Correios.
    d) tolera falhas de rede quando estas ocorrem.

    O manual de integração dos Correios referente ao serviço de preço e prazos (citado no **UC14**), para que você configure de forma adequada o seu dublê, pode ser encontrado no link: http://www.correios.com.br/para-voce/correios-de-a-a-z/pdf/calculador-remoto-de-precos-e-prazos/manual-de-implementacao-do-calculo-remoto-de-precos-e-prazos.

    O manual de integração dos Correios referente ao serviço de rastreio de objetos (citado no **UC15**), para que você configure de forma adequada o seu dublê, pode ser encontrado no link: https://www.correios.com.br/para-voce/correios-de-a-a-z/pdf/rastreamento-de-objetos/Manual_SROXML_28fev14.pdf

    Para consulta do endereço através do CEP (citado no **UC16**), utilize a documentação da API https://viacep.com.br/ para configurar o seu dublê.

ATENÇÃO! Seu objetivo não é implementar o serviço do correio ou componentes extras do Bookstore, muito menos utilizar serviços de homologação, mas sim utilizar dublês para substituí-los enquanto você implementa e testa sua(s) classe(s).

Instruções de apoio

- Baixe o projeto base de exemplo da url: http://www.students.ic.unicamp.br/~otoniel/downloads/inf321_lab02/bookstore-correios.zip ou baixe o pacote já com o eclipse para o ambiente linux na url http://www.students.ic.unicamp.br/~otoniel/downloads/inf321_lab02/inf321-lab02-eclipse-bdd.zip**,** que já possui o projeto e todos os plugins necessários instalados.
- No projeto base há um exemplo de código criado usando BDD, a classe Calculadora.java localizada em src/main/java/br/unicamp/exemplo/. Na pasta do projeto

src/test/resources/features/ está localizada a especificação executável *Calculadora.feature* descrevendo os comportamentos esperados para a classe Calculadora.java, assim como a implementação dos passos dos cenários (step definitions) localizado em: src/test/java/br/unicamp/exemplo/steps/. Para executar o exemplo abra a classe RunCalculadoraTest.java localizada em src/test/java/br/unicamp/exemplo/runner/ e execute a classe com o junit.

- Levando em consideração que não há o aplicativo pronto ainda do bookstore, ao implementar o código relacionado aos casos de uso **UC14**, **UC15** e **UC16**, deve-se construir métodos que receberão por parâmetro os dados que são pré-requisitos para a execução dos casos de uso. Isso facilitará a integração do código criado com o aplicativo posteriormente. Assim sendo, os testes implementados deverão exercitar diretamente o código, invocando os métodos criados passando por parâmetro valores pertinentes para validar o código e cobrir o máximo de casos possíveis.

- Para implementar os requisitos descritos no caso de uso **UC14** relacionados a salvar as informações na base de dados, utilize o método saveDadosDeEntrega da interface DadosDeEntregaDAO.java localizada no projeto base na pasta src/main/java/br/unicamp/comprefacil/dao/. Durante a implementação do teste, utilize o Mockito para mockar o DAO e verificar se o método saveDadosDeEntrega foi invocado corretamente.

- Utilize o framework de mock WireMock para criar os dublês para o serviço dos Correios, de forma que você fique concentrado no que interessa neste momento: o comportamento esperado do seu sistema. A dependência para o WireMock já foi incluída no projeto modelo disponibilizado.

- Utilize o Mockito caso necessite de outros dublês de componentes de terceiros. O projeto modelo disponibilizado também já inclui a dependência para o Mockito.

- Lembre-se de que seu código também deverá tolerar falhas em que há lentidão nos serviços dos correios, assim como respostas que indicam erros no tráfego de dados (http://wiremock.org/simulating-faults.html).

- Utilize o Cucumber-JVM integrado com o Junit para implementar os testes de caixa preta e escrever os cenários na linguagem Gherkin conforme citado anteriormente. O projeto modelo disponibilizado no Moodle já inclui as bibliotecas no arquivo pom.xml.

- Para facilitar a edição do arquivo feature contendo a especificação executável, instale o plugin do eclipse BecauseQA-Cucumber (Siga as instruções de instalação no link:

http://students.ic.unicamp.br/~otoniel/downloads/inf321_lab02/becauseQA-cucumber/install.html). O eclipse disponiblizado já possui o plugin instalado.

- Utilize o plugin EclEmma para análise de cobertura de código para ter certeza de que seus testes garantem o critério de cobertura requisitado. Para instalar, siga as instruções em http://www.eclemma.org/installation.html. O eclipse disponiblizado no zip já possui o plugin instalado.

- Evite lógica de programação (laços, condicionais, dados aleatórios) na implementação dos passos (step definitions) dos cenários da especificação executável. Isso torna o teste mais complexo e difícil de manter. Sempre que possível utilize métodos/classes helpers para facilitar o entendimento da implementação dos passos dos cenários.

- Outros links de apoio:
  https://thomassundberg.wordpress.com/2014/05/29/cucumber-jvm-hello-world/
  https://skillsmatter.com/skillscasts/3060-getting-started-with-cucumber-jvm
  https://dzone.com/refcardz/mockito
  http://wiremock.org/getting-started.html
  http://testdetective.com/rest-api-mocking-with-wiremock/

Entrega

- O projeto Java que contenha as especificações executáveis que descrevem os comportamentos esperados do seu sistema, as classes que implementam os passos de cada cenário, as classes de teste que integram o Cucumber ao Junit para executar os testes e o código que implementa os casos de uso solicitados.

- Cada membro do grupo deverá completar a avaliação de sua equipe, utilizando o formulário online: http://goo.gl/forms/avE7CDPlUl. O não preenchimento deste formulário até o prazo de entrega do trabalho também conta como atraso na entrega.

Prazo

Até 14/Out, 23h55.

Critérios de avaliação

- 10% - Todos os itens requisitados foram entregues?
- 10% - Os cenários criados na especificação executável foram escritos de forma clara?
- 10% - O código dos testes implementados não utilizam lógica de programação (loops, condicionais, dados aleatórios)? Foram criados classes/métodos Helpers para deixar o código do teste mais limpo?
- 30% - Foram criados cenários para todos os requisitos informados?
- 20% - O código implementado condiz com o que foi especificado?
- 20% - O critério de cobertura requisitado foi satisfeito?