ACH 2006
Engenharia de Sistemas de Informação I

Aula 19 - Teste de Software: TDD e BDD

Prof. Marcelo Medeiros Eler
marceloeler@usp.br

Escola de Artes, Ciências e Humanidades - Universidade de São Paulo

Test Driven Development (TDD)

Alguns conceitos iniciais:

- Em português: desenvolvimento dirigido (orientado) por testes
- É uma prática de desenvolvimento de software criada por Kent Beck (criador do XP - Extreme Programming)
- Consiste em escrever os testes (unitários) antes de escrever o código que implementa o comportamento que vai ser testado (Test first)

Test Driven Development (TDD)

Ciclo ou processo básico:

1 - Escolha uma funcionalidade

2 - Escreva um teste

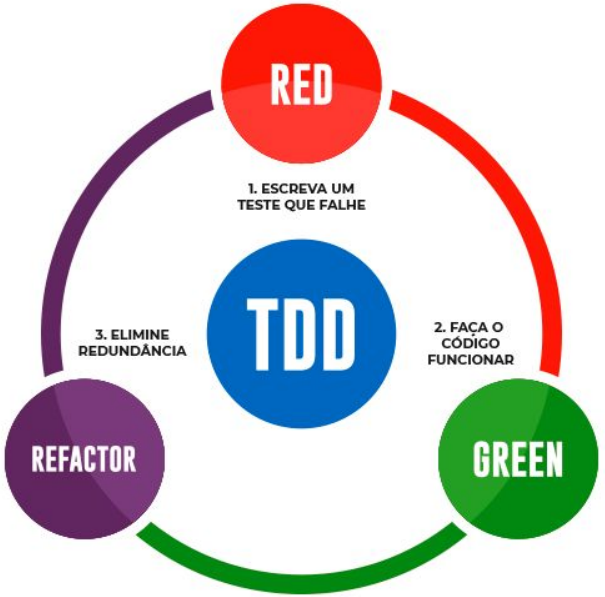
3 - Execute o teste (deve falhar)

4 - Desenvolva para o teste passar

5 - Execute o teste (deve passar)

6 - Refatore

7 - Volte para o passo 2



Fonte: https://www.treinaweb.com.br/blog/afinal-o-que-e-tdd

Test Driven Development (TDD)

Regras:

1 - Não escreva código antes de escrever uma especificação de teste que falhe

2 - Não escreva mais do que um teste para falhar (um de cada vez)

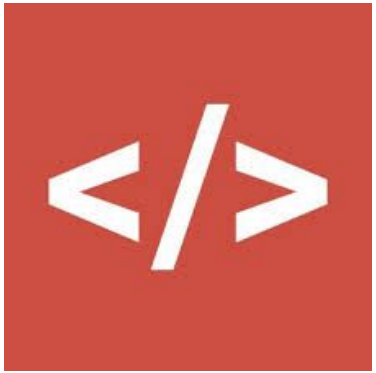
3 - Escreva somente código suficiente para que o teste passe (o mais simples possível)

Demonstração

Função: classificação de triângulo

- A função recebe como entrada três números inteiros positivos que representam os tamanhos dos três lados de um triângulo, e retorna o tipo de triângulo formado (Equilátero, isósceles ou escaleno).
- Se algum dos lados for negativo, o algoritmo deve informar que houve um erro porque um dos lados não tem valor válido
- Se um dos lados tiver tamanho maior ou igual a soma dos outros dois lados, então o algoritmo deve informar que esses valores não são suficientes para formar um triângulo

Demonstração



Coderpad

Python

Pytest

Demonstração

testtriangulo.py

Considerações finais sobre o TDD

TDD é uma prática de desenvolvimento

Os testes literalmente dirigem/guiam o desenvolvimento do código

Todo código escrito tem um teste associado

Os testes são uma especificação do comportamento esperado do software e verificam o comportamento implementado

Mudança de perspectiva: testar o que está desenvolvido VS testar o que está especificado

Teste desacoplado da implementação

Considerações finais sobre o TDD

Vantagens do TDD:

- Feedback rápido sobre a nova funcionalidade e as outras funcionalidades
- Código mais limpo, já que escrevemos códigos simples para o teste passar
- Segurança na refatoração e na correção de bugs
- Maior produtividade já que o desenvolvedor encontra menos bugs e não desperdiça tempo com depuradores
- Código da aplicação mais flexível, já que para escrever testes temos que separar em pequenos "pedaços" o nosso código, para que sejam testáveis, ou seja, nosso código estará menos acoplado.

Fonte: https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533

Considerações finais sobre o TDD

Curiosidade:
1. “A software system can best be designed if the testing is interlaced with the designing instead of being used after the design.”

2. “A simulation which matches the requirements contains the control which organizes the design of the system.”

3. “Through successive repetitions of this process of interlaced testing and design the model ultimately becomes the software system itself.”

4. “I think that it is the key of the approach that has been suggested, that there is no such question as testing things after the fact with simulation models, but that in effect the testing and the replacement of simulations with modules that are deeper and more detailed goes on with the simulation model controlling, as it were, the place and order in which these things are done."

Alan Perlis, NATO conference on Software Engineering, 1968

Atividade prática

Aplique TDD para o desenvolvimento da seguinte função: **calcula_taxa_desconto(tipo_cliente, valor_compra)**

Esta função retorna o valor de desconto de uma compra com base em dois critérios: tipo de cliente e valor da compra. As regras são as seguintes:

- Cliente bronze tem 5% de desconto
- Cliente prata tem 10% de desconto
- Cliente ouro tem 15% de desconto
- Compras entre 500 e 1000 (inclusive) tem desconto de 10%
- Compras acima de 1000 tem desconto de 15%
- Os descontos não acumulam. Na incidência de dois descontos, deve-se aplicar a regra que oferece o maior desconto. Por exemplo, a taxa de desconto para cliente bronze com compra acima de 500 reais é de 15%.
- Qualquer configuração diferente das regras anteriores tem taxa de 0% de desconto

Tempo para a atividade prática: 10 minutos

Behavior Driven Development (BDD)

Em português, Desenvolvimento Dirigido por Comportamento

O BDD foi criado por Dan North em 2006 para suprir algumas questões conceituais do TDD:

- No TDD, o nome do jogo não é necessariamente teste, mas especificação
- No TDD, o objetivo é confirmar um comportamento especificado, e não testar código escrito

Behavior Driven Development (BDD)

Segundo Dan North, ele criou o BDD para apresentar o TDD de uma maneira direta e objetiva, e com foco nas questões de comportamento e não com foco em teste

Algumas reflexões o ajudaram a definir sua metodologia:

- Os nomes de métodos de teste deveriam ser sentenças
- Um modelo de sentença simples deixa os testes mais focados
- Um nome de teste expressivo/significativo é útil quando o teste falha
- Comportamento é uma palavra mais útil do que teste
- Requisitos também são comportamentos

Os nomes de métodos de teste deveriam ser sentenças

```
public class CustomerLookupTest extends TestCase {
  testFindsCustomerById() {
    ...
  }

  testFailsForDuplicateCustomers() {
    ...
  }
  ...
}
```


```
CustomerLookup
- finds customer by id
- fails for duplicate customers
- ...
```

Os nomes de métodos de teste deveriam ser sentenças

Desenvolvedores descobriram que **agiledox** poderia fazer uma documentação para eles, então começaram a escrever os nomes dos métodos como se fossem sentenças com significado real

Além disso, descobriram que quando eles escreviam métodos na linguagem de negócio, considerando os requisitos e o domínio de informação, os documentos gerados faziam sentido até para os analistas de negócio

Um modelo de sentença simples deixa os testes mais focados

Convenção: The class should do something (A classe deveria fazer algo)

Por exemplo, uma classe de teste que valida dados de entrada com os seguintes métodos:

- testShouldFailForMissingSurname
- testShouldFailForMissingTitle

Comportamento é uma palavra mais útil do que teste

É claro que o teste está relacionado com o TDD, pois o resultado é um conjunto de métodos que “garante” que o código funciona em um certo contexto.

Entretanto, se os métodos não descrevem de forma compreensiva o comportamento do sistema, então os testes vão somente dar uma falsa sensação de que o sistema se comporta como deveria

Então Dan North passou a usar o termo comportamento (behavior) ao invés de teste.

Com isso, passou a entender que o teste deve ser escrito como uma sentença do próximo comportamento em que você está interessado em ver funcionar.

A linguagem do BDD

Com base em histórias de usuário

- As a [X] (Como um [X])

  I want [Y] (Eu quero [Y])

  So that [Z] (Para que [Z])

Foi criada um template para representar os critérios de aceitação de histórias:

- Given some initial context (the givens), (Dado algum contexto)

  When an event occurs, (Quando algum evento acontece)

  Then ensure some outcomes. (Então as saídas são garantidas)

A linguagem do BDD

Exemplo:

```
Title: Customer withdraws cash**

As a customer,
I want to withdraw cash from an ATM,
so that I don't have to wait in line at the bank.
```

```
Scenario: Account is in credit

Given the account is in credit
And the card is valid
And the dispenser contains cash
When the customer requests cash
Then ensure the account is debited
And ensure cash is dispensed
And ensure the card is returned
```

```
Scenario: Account is overdrawn past the overdraft limit

Given the account is overdrawn
And the card is valid
When the customer requests cash
Then ensure a rejection message is displayed
And ensure cash is not dispensed
And ensure the card is returned
```

Critérios de aceitação devem ser executáveis

Os cenários podem ser implementados para que sejam executáveis, utilizando a mesma linguagem

```
public class AccountIsInCredit implements Given {
  public void setup(World world) {
    ...
  }
}

public class CardIsValid implements Given {
  public void setup(World world) {
    ...
  }
}
```

```
public class CustomerRequestsCash implements Event {
  public void occurIn(World world) {
    ...
  }
}
```

Linguagens, frameworks e ferramentas

Para apoiar o uso de BDD na prática, com a especificação e implementação direta dos cenários, diversos recursos, frameworks e ferramentas foram criados

- Gherkin: linguagem para descrição de comportamento
- Cucumber: ferramenta que suporta o BDD
- RobotFramework: framework baseado em keywords que apoia o BDD

Linguagens, frameworks e ferramentas

Para apoiar o uso de BDD na prática, com a especificação e implementação direta dos cenários, diversos recursos, frameworks e ferramentas foram criados

- **Gherkin: linguagem para descrição de comportamento**
- Cucumber: ferramenta que suporta o BDD
- **RobotFramework: framework baseado em keywords que apoia o BDD**

Gherkin

Foi criada por Aslak Hellesøy, pois ele estava incomodado como os requisitos eram escritos, de forma ambígua e mal compreendida.

Em 2003, Aslak soube da existência de um grupo de pessoas que buscavam maneiras de como o TDD poderia ser melhor aplicado

A ideia era conseguir combinar testes de aceitação automatizados , requisitos funcionais e outros artefatos de software em um formato que fosse compreendido por qualquer pessoa

Após muitas iterações, a ferramenta Cucumber e sua linguagem Gherkin foram criadas em 2008.

Gherkin

https://medium.com/revista-tspi/gherkin-o-dia-em-que-entendi-que-estava-escrevendo-errado-220a84520819

Gherkin

https://medium.com/revista-tspi/gherkin-o-dia-em-que-entendi-que-estava-escrevendo-errado-220a84520819

Gherkin

Cenário (Scenario):

- Um cenário representa um comportamento específico do sistema que se deseja testar. Ele consiste em uma sequência de passos que descrevem a interação entre o usuário e o sistema.

Cenário: Realizar login no sistema com sucesso
Dado que o usuário está na página de login
Quando os dados de autenticação são fornecidos
Então é possível ver a página inicial do sistema

Gherkin

Cenário (Scenario):

- Um cenário representa um comportamento específico do sistema que se deseja testar. Ele consiste em uma sequência de passos que descrevem a interação entre o usuário e o sistema.

Cenário: Realizar pagamento com cartão de crédito e envio gratuito
Dado que o usuário está na página de pagamento
Quando ele opta pelo envio gratuito
E informa os dados do seu cartão de crédito
E confirma a operação
Então é possível ver a informação de que a compra foi bem sucedida

Gherkin (https://cucumber.io/docs/gherkin/reference/)

Gherkin usa um conjunto especial de palavras-chave para dar estrutura e significado a especificações executáveis

A maioria das linhas em um documento Gherkin começa com uma das palavras-chave

Comentários são permitidos apenas no início de uma nova linha. Não são permitidos blocos de comentários.

A indentação é feita com espaços (dois) ou com tab.

Gherkin

As principais palavras-chave são (em inglês):

- Feature
- Rule
- Example (or Scenario)
- Given, When, Then, And, But for steps (or *)
- Background
- Scenario Outline (or Scenario Template)
- Examples (or Scenarios)

Gherkin

As principais palavras-chave são (em português):

- Funcionalidade ou Característica
- Regra
- Cenário, Exemplo
- Dado, Quando, Então, E, MAS
- Contexto, Fundo, Cenário de Fundo
- Esquema do Cenário, Delineação do Cenário

Gherkin

Feature: o propósito da palavra feature é fornecer uma descrição em alto nível de uma funcionalidade do software ou de um grupo de cenários relacionados. É a primeira palavra de um documento Gherkin.

```
Feature: Guess the word

  The word guess game is a turn-based game for two players.
  The Maker makes a word for the Breaker to guess. The game
  is over when the Breaker guesses the Maker's word.

  Example: Maker starts a game
```

Gherkin

Rule: o propósito da palavra Rule é representar uma regra de negócio que deveria ser implementada.

- Ela fornece informação adicional para uma funcionalidade.
- Ela agrupa vários cenários que pertencem a uma regra de negócio.
- Ela deve ter um ou mais cenários que ilustram aquela regra em particular.

Gherkin

Rule

```
# -- FILE: features/gherkin.rule_example.feature
Feature: Highlander

  Rule: There can be only One

    Example: Only One -- More than one alive
      Given there are 3 ninjas
      And there are more than one ninja alive
      When 2 ninjas meet, they will fight
      Then one ninja dies (but not me)
      And there is one ninja less alive

    Example: Only One -- One alive
      Given there is only 1 ninja alive
      Then he (or she) will live forever ;-)

  Rule: There can be Two (in some cases)

    Example: Two -- Dead and Reborn as Phoenix
      ...
```

Gherkin

Given: são passos usados para descrever o **contexto inicial** do sistema - define o início de um cenário. É tipicamente algo que aconteceu no passado.

Exemplos:

- Mickey and Minnie have started a game
- I am logged in
- Joe has a balance of £42

Gherkin

When: usados para descrever um **evento** ou uma **ação**. Pode ser a interação de uma pessoa com o sistema, ou pode ser um evento acionado por outro sistema.

É recomendado que se use apenas uma palavra When para cada cenário. Se realmente for necessário utilizar mais de um, talvez seja o caso de dividir este cenário em outros cenários.

Exemplos:

- Guess a word
- Invite a friend
- Withdraw money

Gherkin

Then: são usados para descrever os **resultados esperados**. Na prática, corresponde a uma asserção para comparar o resultado atual de uma ação com o esperado.

Uma saída deve ser algo observável, não um comportamento interno do sistema ou alguma alteração em uma base de dados.

Exemplos:

- See that the guessed word was wrong
- Receive an invitation
- Card should be swallowed

Gherkin

And, But: para conectar sucessivos Given, When, Then

```
Example: Multiple Givens
  Given one thing
  Given another thing
  Given yet another thing
  When I open my eyes
  Then I should see something
  Then I shouldn't see something else
```

```
Example: Multiple Givens
  Given one thing
  And another thing
  And yet another thing
  When I open my eyes
  Then I should see something
  But I shouldn't see something else
```

Gherkin

* - gherkin também prevê o uso de asterisco (*) no lugar de um passo normal. Isso pode ser útil quando alguns passos são efetivamente uma lista de coisas, então elas podem ser expressas como itens de uma lista para não ter que usar muitos “and”

```
Scenario: All done
  Given I am out shopping
  And I have eggs
  And I have milk
  And I have butter
  When I check my list
  Then I don't need anything
```

```
Scenario: All done
  Given I am out shopping
  * I have eggs
  * I have milk
  * I have butter
  When I check my list
  Then I don't need anything
```

Gherkin

Background: é comum que você precise repetir as mesmas coisas no passo Given para cenários dentro de uma mesma Feature. A palavra Background serve para definir um contexto comum para todos eles.

O que é definido pela palavra Background é executada antes de cada cenário

Gherkin

Background:

```
Feature: Multiple site support
  Only blog owners can post to a blog, except administrators,
  who can post to all blogs.

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Dr. Bill"
    And a blog named "Expensive Therapy" owned by "Dr. Bill"

  Scenario: Dr. Bill posts to his own blog
    Given I am logged in as Dr. Bill
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."

  Scenario: Dr. Bill tries to post to somebody else's blog, and fails
    Given I am logged in as Dr. Bill
    When I try to post to "Greg's anti-tax rants"
    Then I should see "Hey! That's not your blog!"

  Scenario: Greg posts to a client's blog
    Given I am logged in as Greg
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."
```

Gherkin

Scenario Outline: pode ser utilizado para executar o mesmo cenário diversas vezes, mas com diferentes combinações de valores.

```
Scenario: eat 5 out of 12
  Given there are 12 cucumbers
  When I eat 5 cucumbers
  Then I should have 7 cucumbers

Scenario: eat 5 out of 20
  Given there are 20 cucumbers
  When I eat 5 cucumbers
  Then I should have 15 cucumbers
```

```
Scenario Outline: eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers

  Examples:
    | start | eat | left |
    |    12 |   5 |    7 |
    |    20 |   5 |   15 |
```

RobotFramework (https://robotframework.org/)

É um framework de código aberto para automação tanto de teste quanto de processos (RPA)

Ele tem uma sintaxe simples e é baseado em palavras-chave compreensíveis por qualquer pessoa

Ele possui várias bibliotecas e pode ser estendido conforme a necessidade de seus usuários

Ele pode ser integrado a outras ferramentas e frameworks (Selenium, por exemplo)

RobotFramework

Algumas bibliotecas externas:

- **SeleniumLibrary** - Web testing library that uses popular Selenium tool internally.
- **RPA framework** - Collection of open-source libraries and tools for Robotic Process Automation (RPA), designed to be used both with Robot Framework and Python.
- **HTTP RequestsLibrary (Python)** - HTTP level testing using Python Requests internally.
- **Browser Library** - A modern web testing library powered by Playwright. Aiming for speed, reliability and visibility.
- **AppiumLibrary** - Android and iOS testing. Uses Appium internally.
- **RESTinstance** - Test library for HTTP JSON APIs.
- **SSHLibrary** - Enables executing commands on remote machines over an SSH connection. Also supports transfering files using SFTP

RobotFramework

Algumas bibliotecas internas:

- **Builtin** - Provides a set of often needed generic keywords. Always automatically available without imports.
- **Collections** - Provides a set of keywords for handling Python lists and dictionaries.
- **DateTime** - Library for date and time conversions.
- **Dialogs** - Provides means for pausing the execution and getting input from users.
- **Libdoc** - Generate keyword documentation for test libraries and resource files.
- **OperatingSystem** - Enables various operating system related tasks to be performed in the system where Robot Framework is running.
- **Process** - Library for running processes in the system.

Teste end-to-end (Robot Framework)

```
*** Test Case ***
Scenario: Buy product
 Wait Until Page Contains Element  edit_text_username
 Input Text   edit_text_username  "User12345"
 Input Text   edit_text_passwd    "Mypassword12345"
 Wait Until Page Contains Element  btn_login
 Click Element  btn_login
 Wait Until Page Contains Element  edit_text_search
 Input Text   edit_text_search "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_search
 Click Element  btn_search
 Page Should Contain Text  "Micro USB Charger Cable"
 Click Text  "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_buy
 Click Element  btn_buy
 Wait Until Page Contains Element  btn_checkout
 Click Element  btn_checkout
 ...
```

Teste end-to-end (Robot Framework)

```
*** Test Case ***
Scenario: Buy product
 Log User
 Search for product
 Buy product

*** Keywords ***
Log User
 Wait Until Page Contains Element  edit text username
 Input Text   edit text username  "User12345"
 Input Text   edit_text_passwd    "Mypassword12345"
 Wait Until Page Contains Element  btn_login
 Click Element  btn_login

Search for product
 Wait Until Page Contains Element  edit_text_search
 Input Text   edit_text_search "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_search
 Click Element  btn search
 Page Should Contain Text  "Micro USB Charger Cable"

By product
 Click Text  "Micro USB Charger Cable"
 Wait Until Page Contains Element  btn_buy
 Click Element  btn buy
 Wait Until Page Contains Element  btn_checkout
 Click Element  btn_checkout
```

Teste end-to-end (Robot Framework)

```
*** Test Case ***
Scenario: Buy product
 Log User  "User12345" "Mypassword12345"
 Search for product  "Micro USB Charger Cable"
 Buy product  "Micro USB Charger Cable"

*** Keywords ***
Log User  {$username}  {$password}
 Wait Until Page Contains Element  edit text username
 Input Text   edit text username  {$username}
 Input Text   edit_text_passwd    {$password}}
 Wait Until Page Contains Element  btn_login
 Click Element  btn_login

Search for product  {$product}
 Wait Until Page Contains Element  edit_text_search
 Input Text   edit_text_search {$product}
 Wait Until Page Contains Element  btn_search
 Click Element  btn search
 Page Should Contain Text  {$product}

By product  {$product}
 Click Text  {$product}
 Wait Until Page Contains Element  btn_buy
 Click Element  btn buy
 Wait Until Page Contains Element  btn_checkout
 Click Element  btn_checkout
```

Exemplo de BDD (Robot Framework)

```
*** Test Cases ***
Login With Admin
    Given I am on the login page
    When I login with username "admin" and password "admin"
    Then I should see the welcome page

Login With Invalid User
    Given I am on the login page
    When I login with username "invalid" and password "invalid"
    Then I should see the error message
    And I should be able to login again
```

RobotFramework

Demonstração: teste remover item de uma lista (app todo list)

VS Code

RobotFramework

RobotFramework

Robot Framework online: https://robotframework.org/#getting-started

RobotFramework

Para quem quiser ver mais exemplos e explicações sobre BDD e BDD+RobotFramework:

- https://www.youtube.com/watch?v=xwUFQpjHVy0&t=807s

Considerações finais

A atividade de teste de software não pode estar desvinculada da atividade de desenvolvimento

Abordar o teste como parte da especificação do software ajuda a diminuir o acoplamento entre os testes e a implementação - o foco passa a ser o comportamento do software com base em regras de negócio e requisitos declarados explicitamente

A cultura de teste contínuo é fundamental para manter a qualidade global de qualquer produto de software ao longo do tempo

Referências

Kent, B.; Andres, C. Extreme Programming: Explained. 2nd Edition. Addison-Wesley, 2004.

Sommerville, Ian. Engenharia de Software. São Paulo Pearson Prentice Hall, 2011. 9a edição.

Wazlawick, Raul Sidnei. Engenharia de Software –Conceitos e Práticas. Editora Campus, 2013. 1a edição.

Engineering Software as a Service: An Agile Approach Using Cloud Computing Second Edition. 2021. Armando Fox and David Patterson.

http://manifestoagil.com.br/

Referências

The Practical Test Pyramid. Autor: Ham Vocke.
https://martinfowler.com/articles/practical-test-pyramid.html

Certified Tester Syllabus. Foundation Level. CTFL 3.1. BTSQB.
https://bstqb.org.br/b9/doc/syllabus_ctfl_3.1br.pdf

Delamaro, M.E., Maldonado J.C., Jino, M. Introdução ao Teste de Software, Rio de Janeiro, Elsevier, 2007

Freeman, S. e Pryce, N., “Growing Object-Oriented Software Guided by Tests”, Addison-Wesley Professional, 2009

Aniche, M., “Test-Driven Development - Teste e Design no Mundo Real”, Casa do Código, 2012.

https://dannorth.net/introducing-bdd/

ACH 2006
Engenharia de Sistemas de Informação I

Aula 19 - Teste de Software: TDD e BDD

Prof. Marcelo Medeiros Eler
marceloeler@usp.br

Escola de Artes, Ciências e Humanidades - Universidade de São Paulo