UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Test Driven Development (TDD)

Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Introdução

- O que é Teste de Unidade?
  - Trecho de código que valida uma hipótese
  - O objetivo é isolar pequenas partes e verificar que individualmente estão corretas
- Quais os benefícios?
  - Detectar erros no início do desenvolvimento
  - Melhor entendimento do código
  - Utilização de ferramentas automatizadas

Introdução

- O quê é cobertura de código?
  - Medida usada para descrever o grau em que o código-fonte de um programa é executado quando um conjunto de testes é rodado
- Uma alta cobertura é garantia de sucesso?
  - NÃO!
  - Sugere que tem uma chance menor de conter erros de software não detectados
  - Dependerá da qualidade dos testes!

Introdução

- Quando devemos iniciar a fase de testes?
  - Apenas depois de terminada a codificação?
  - Não é uma boa ideia! Por que?
- Desenvolvimento Orientado por Testes
  - Foco deve ser no requisito, não no código!
  - Interface → Comportamento

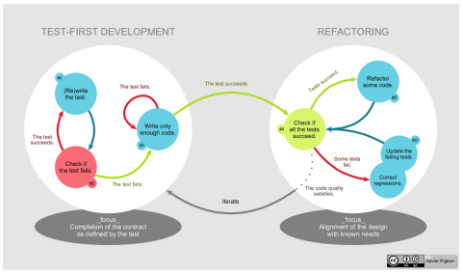
Metodologia de desenvolvimento

Test Driven Development (TDD)

- Escrever o teste antes do código
- Prática extremamente recomendada
  - Código se adapta ao teste, não o contrário!
- Abordagem incremental
  - Pequenos passos (testes) ajudam a alcançar um resultado final de qualidade (projeto)
- Maneira de se desenvolver, não de testar!

Metodologia de desenvolvimento

Test Driven Development (TDD)



Metodologia de desenvolvimento
Test Driven Development (TDD)

- Estratégias de implementação
  - Fake it
    - Retorne uma constante e gradualmente substitua com variáveis até que você tenha o código real
  - Triangulation
    - Pense no exemplo mais simples possível da solução
    - Escreva um teste para cobrir esse exemplo, implemente a solução e refatore.
    - Repita até não conseguir pensar em mais exemplos
  - Obvious implementation
    - Faça a implementação esperada do método

Exemplo 1

- Fizz Buzz
  - Jogo para ensinar crianças sobre a divisão
  - Sequência de números é falada, substituindo:
    - Números divisíveis por 3: "Fizz"
    - Números divisíveis por 5: "Buzz"
    - Números divisíveis por ambos: "FizzBuzz"
- Recebe uma lista e converte para o resultado
  - $[1, 2, 3, 5, 6, 10, 15, 30]$
  - "1, 2, Fizz, Buzz, Fizz, Buzz, FizzBuzz, FizzBuzz"
- Por onde começar?

https://en.wikipedia.org/wiki/Fizz_buzz

Exemplo 1

```
#ifndef FIZZBUZZ_H
#define FIZZBUZZ_H

#include <string>
#include <vector>

class FizzBuzz {

  public:
    std::string convert(std::vector<int> v);

};

#endif
```

```
#include "FizzBuzz.hpp"

std::string FizzBuzz::convert(std::vector<int> v) {

  return "";

}
```


Exemplo 1

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"

#include "FizzBuzz.hpp"

TEST_CASE("Teste FizzBuzz - Numeros Simples") {

  FizzBuzz fb;

  std::vector<int> v1 = {1};
  CHECK(fb.convert(v1) == "1");

  std::vector<int> v2 = {2};
  CHECK(fb.convert(v2) == "2");
}

TEST_CASE("Teste FizzBuzz - Multiplos de 3") {

  FizzBuzz fb;

  std::vector<int> v1 = {3};
  CHECK(fb.convert(v1) == "Fizz");

  std::vector<int> v2 = {6};
  CHECK(fb.convert(v1) == "Fizz");
}

TEST_CASE("Teste FizzBuzz - Multiplos de 5") {

  FizzBuzz fb;

  std::vector<int> v1 = {5};
  CHECK(fb.convert(v1) == "Buzz");

  std::vector<int> v2 = {10};
  CHECK(fb.convert(v1) == "Buzz");
}

TEST_CASE("Teste FizzBuzz - Multiplos de 3 e 5") {

  FizzBuzz fb;

  std::vector<int> v1 = {15};
  CHECK(fb.convert(v1) == "FizzBuzz");

  std::vector<int> v2 = {30};
  CHECK(fb.convert(v1) == "FizzBuzz");
}
```

Complete com outros testes!

Exemplo 1

```
#include "FizzBuzz.hpp"

std::string FizzBuzz::convert(std::vector<int> v) {

  if (v[0] % 3 == 0)
    return "Fizz";
  else if (v[0] % 5 == 0)
    return "Buzz";
  else if ((v[0] % 3 == 0) && (v[0] % 5 == 0))
    return "FizzBuzz";
  else
    return std::to_string(v[0]);

  return "";

}
```

https://wandbox.org/permlink/oBR3Ziv31INAAE5I

Corrija os erros, adicione novos testes, complete o código e então refatore!

Exemplo 2

- Fibonacci
  - Sequência de números definida por
    - $F_n = F_{n-1} + F_{n-2}$
  - onde
    - $F_0 = 0$ e $F_1 = 1$
- Parâmetros
  - Entrada: ordem do elemento (6)
  - Saída: valor do elemento $(0, 1, 1, 2, 3, 5, 8)$
- Por onde começar?
  - Casos base
  - Demais casos
  - Casos excepcionais

https://en.wikipedia.org/wiki/Fibonacci_number

Exemplo 2

```
#ifndef FIBONACCI_H
#define FIBONACCI_H

class Fibonacci {

  public:
    int getElement(int n);

};

#endif
```

```
#include "Fibonacci.hpp"

int Fibonacci::getElement(int n) {

  return 0;

}
```


Exemplo 2

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"

#include "Fibonacci.hpp"

TEST_CASE("Teste Fibonacci - Base") {
  Fibonacci fb;
  CHECK(fb.getElement(0) == 0);
  CHECK(fb.getElement(1) == 1);
}

TEST_CASE("Teste Fibonacci - Demais") {
  Fibonacci fb;
  CHECK(fb.getElement(2) == 1);
  CHECK(fb.getElement(6) == 8);
  CHECK(fb.getElement(30) == 832040);
}

TEST_CASE("Teste Fibonacci - Excecao") {
  Fibonacci fb;
  CHECK_THROWS_AS(fb.getElement(-2), std::invalid_argument);
}
```


Exemplo 2

```
#include "Fibonacci.hpp"

int Fibonacci::getElement(int n) {

  if (n == 0) return 0;
  if (n == 1) return 1;
  return getElement(n - 1) + getElement(n - 2);

}
```

https://wandbox.org/permlink/5tBLVq1agRxI5xNn

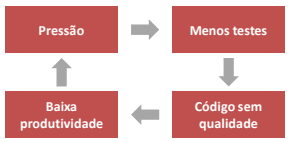
Complete o código para passar em todos os testes e então refatore!

Metodologia de desenvolvimento
Test Driven Development (TDD)

- Tempo de detecção e quantidade diminuem
- Arquitetura do código melhora
  - Mais modular, flexível e extensível
  - Pequenas unidades independentes
- Confiança no código aumenta
  - Assim como a qualidade como um todo
- Custos reduzidos e prazos cumpridos!

Metodologia de desenvolvimento
Test Driven Development (TDD)

- Programadores sabem que devem escrever o teste antes, mas são poucos o que fazem
  - Gera um ciclo vicioso!



Metodologia de desenvolvimento
Test Driven Development (TDD)

- Exige comprometimento da equipe!
- Não garante o sucesso do projeto
  - Os testes foram bem feitos?
  - Robustos, fácil manutenção, realmente testa, ...
- Refatoração ajuda no aumento da qualidade

Exercício

- Entidade Aluno
  - determinarAprovacao
  - Frequencia, NotaFinal, NotaEspecial
- Quais o requisitos?
  - Frequencia < 75% → Reprovado
  - NotaFinal >= 60 → Aprovado
  - (NotaFinal+NotaEspecial)/2 >= 60 → Aprovado