



Test Driven Development

Metodologias e Desenvolvimento de Software

Pedro Salgueiro

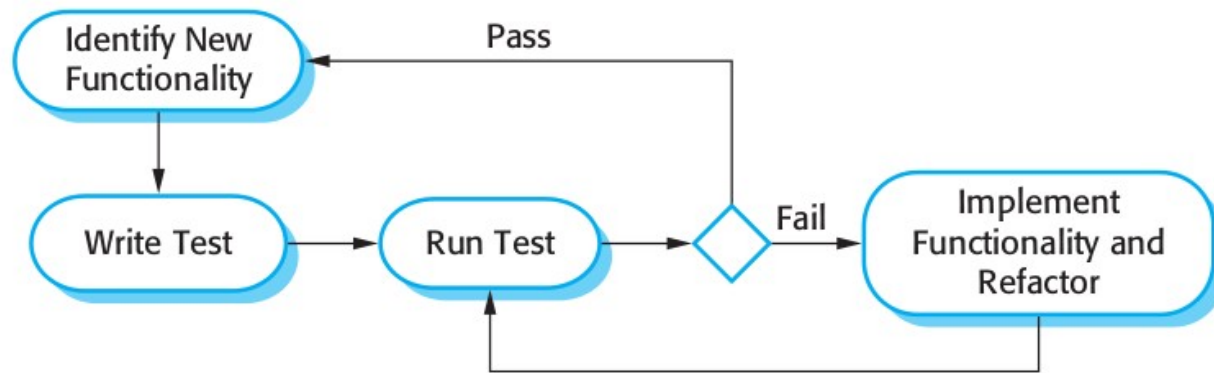
pds@uevora.pt

CLV-256



- Test-driven development
 - TDD
- Metodologia de desenvolvimento baseada em testes
 - Escrita do código é intercalada com testes de software
- Testes são escritos antes da implementação do código
 - O “passar dos testes” é critico no processo de desenvolvimento
- O código é escrito de forma incremental
 - Em conjunto com os testes para esse incremento
 - Apenas se prossegue para o próximo incremento, depois de escrito todo o código que faz passar os testes
- Metodologia introduzida como parte das metodologias ágeis
 - Extreme Programming
 - Pode ser usada em metodologias baseadas em planos

Test-driven development





Processo TDD

Atividades

- Começar por identificar o incremento de funcionalidades necessário
 - Pequeno e possível de implementar em poucas linhas de código
- Escrever os testes para a funcionalidade
 - implementá-lo como um teste autónomo
- Executar o teste
 - Junto com todos os outros testes existentes
 - Inicialmente não existe código implementado para a funcionalidade
 - O teste vai falhar
- Implementar a funcionalidade
 - Voltar a correr os testes
- Quando todos os testes passarem
 - Continuar para a próxima funcionalidade



Vantagens

- Cobertura do código com testes
 - Todos os fragmentos de código têm pelo menos um teste associado
 - Todo o código tem pelo menos um teste
- Teste de regressão
 - Conjunto de testes criado de forma incremental, à medida que o sistema é desenvolvido
- Debug mais simples
 - Quando um teste falha, torna-se fácil encontrar o local do problema.
 - Código novo/alterado tem de ser verificado e alterado
- Documentação do sistema
 - Os testes são uma forma de documentação
 - Descrevem o que o código deve fazer



Exemplo

- Criar uma calculadora de Strings com o método
 - `Int add(string numbers)`
- Requisitos
 - O método pode receber 0, 1 ou 2 números e devolver a sua soma
 - Exemplos: “”, “1”, “1,2”
 - Para uma string vazia, deve devolver 0
 - Permitir que o método receba um número desconhecido de números
 - Permitir que o método trate o “new line”(\n) como forma de separar os números



- Requisito:
 - “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
- Por onde começar
 - Começar por testar os casos mais simples
 - Existência do método add
 - Verificar se os argumentos estão correctos
 - Verificar os casos especiais
 - Devolvido 0 se string vazia
 - Verificar o comportamento normal do método add
 - Devolve a soma dos números



- Requisito: “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
 - 1ª iteração de testes

```
public class StringCalculatorTest {  
    @Test(expected = RuntimeException.class)  
    public final void whenMoreThan2NumbersAreUsedThenExceptionIsThrown() {  
        StringCalculator.add("1, 2, 3");  
    }  
    @Test  
    public final void when2NumbersAreUsedThenNoExceptionIsThrown() {  
        StringCalculator.add("1, 2");  
        Assert.assertTrue(true);  
    }  
    @Test(expected = RuntimeException.class)  
    public final void whenNonNumberIsUsedThenExceptionIsThrown() {  
        StringCalculator.add("1, X");  
    }  
}
```




- Requisito: “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
 - 1ª iteração de implementação

```
public class StringCalculator {  
    public static final void add(final String numbers) {  
        String[] numbersArray = numbers.split(",");  
        if (numbersArray.length > 2) {  
            throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");  
        } else {  
            for (String number : numbersArray) {  
                // If it is not a number, parseInt will throw an exception  
                Integer.parseInt(number);  
            }  
        }  
    }  
}
```



- Requisito: “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
 - 2ª iteração de testes
 - Verificar casos especiais

```
@Test
public final void whenEmptyStringIsUsedThenReturnValueIs0() {
    Assert.assertEquals(0, StringCalculator.add(""));
}
```



- Requisito: “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
 - 2ª iteração de implementação

```
public static final int add(final String numbers) { // Changed void to int
    String[] numbersArray = numbers.split(",");
    if (numbersArray.length > 2) {
        throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
    } else {
        for (String number : numbersArray) {
            if (!number.isEmpty()) {
                Integer.parseInt(number);
            }
        }
    }
    return 0; // Added return
}
```



- Requisito: “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
 - 3ª iteração de testes
 - Verificar o comportamento normal

```
@Test
public final void whenOneNumberIsUsedThenReturnValueIsThatSameNumber() {
    Assert.assertEquals(3, StringCalculator.add("3"));
}

@Test
public final void whenTwoNumbersAreUsedThenReturnValueIsTheirSum() {
    Assert.assertEquals(3+6, StringCalculator.add("3,6"));
}
```



- Requisito: “O método pode receber 0, 1 ou 2 números e devolver a sua soma”
 - 3ª iteração de implementação
 - Verificar o comportamento normal

```
public static int add(final String numbers) {  
    int returnValue = 0;  
    String[] numbersArray = numbers.split(",");  
    if (numbersArray.length > 2) {  
        throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");  
    }  
    for (String number : numbersArray) {  
        if (!number.trim().isEmpty()) { // After refactoring  
            returnValue += Integer.parseInt(number);  
        }  
    }  
    return returnValue;  
}
```



- Software Engineering, Ian Sommerville, 10th Edition, Addison-Wesley, 2016. Capítulo 8