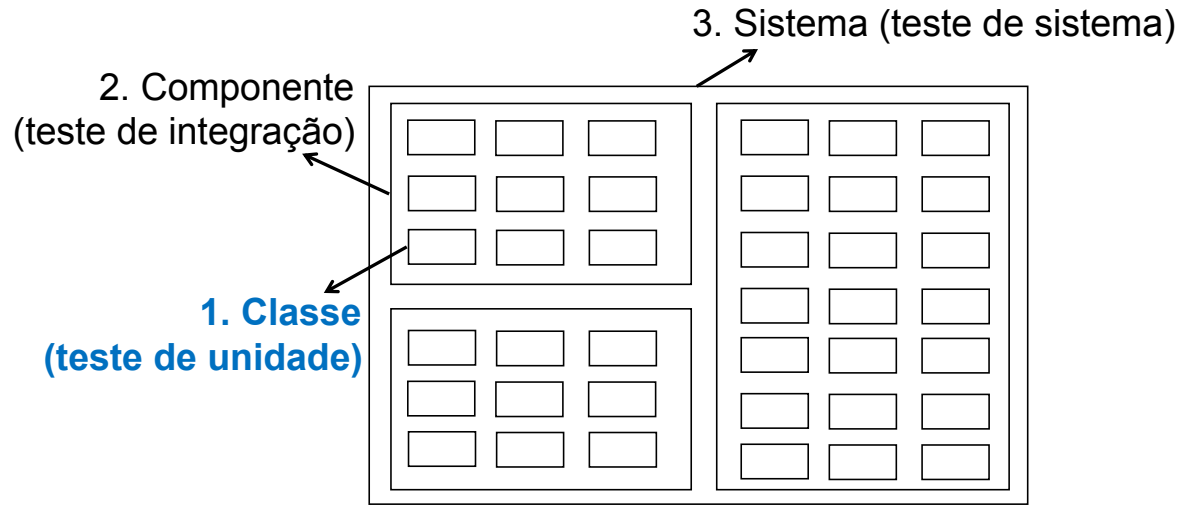
Teste de Software

Prof. Dr. Alan Souza

alan.souza@unama.br

2020

2.2.1. Teste Unitário



2.1.1 Teste unitário

b. Exemplo prático #2:

Criar um projeto em Java que receba uma String de números inteiros separados por vírgula, que efetue e mostre o somatório dos números informados para o usuário. Exs de funcionamento:

Exemplo 1:
Entrada: 3,7,1,2,-4
Saída: 9

Exemplo 2:
Entrada: 3
Saída: 3

Exemplo 3:
Entrada:
Saída: 0

Exemplo 4:
Entrada: 1,3,9,
Saída: 13

Exemplo 5:
Entrada: 5, 1, 9
Saída: 15

Usar JUnit

2.1.1 Teste unitário

b. Resolução - passo 1/6 - criar a classe **Dados** e o método que realiza o somatório:

```
public class Dados {
    private String dados;

    public double somaDados() {
        String[] numeros = dados.split(",");
        double soma = 0.0;
        for(int i = 0; i < numeros.length; i++ ) {
            if( ! numeros[i].isEmpty() )
                soma = soma + Double.parseDouble( numeros[i] );
        }
        return soma;
    }

    public void setDados(String dados) {
        this.dados = dados;
    }
}
```

2.1.1 Teste unitário

b. Resolução - passo 2/6 - criar a classe de testes **DadosTeste** e os casos de teste (usar JUnit):

```
public class DadosTeste {

    Dados d;

    public DadosTeste() {
        d = new Dados();
    }

    @Test
    public void deveRetornarSoma() {
        d.setDados("5,7,9");
        assertEquals(21.0, d.somaDados(), 0);
    }

    @Test
    public void deveRetornarSomaComNumerosReais() {
        d.setDados("5.3,7.2,9.5");
        assertEquals(22.0, d.somaDados(), 0);
    }
```

Um caso de teste

Valor esperado

Valor calculado

Outro caso de teste

2.1.1 Teste unitário

b. Resolução - passo 3/6 - continuação da classe **DadosTeste**:

```
@Test
public void deveRetornarSomaComVirgulaNoFinal() {
    d.setDados("5,7,9,");
    assertEquals(21.0, d.somaDados(), 0);
}

@Test
public void deveRetornarSomaComVirgulaNoInicio() {
    d.setDados(",5,7,9");
    assertEquals(21.0, d.somaDados(), 0);
}

@Test
public void deveRetornarSomaUnicoNumero() {
    d.setDados("5");
    assertEquals(5.0, d.somaDados(), 0);
}
```

2.1.1 Teste unitário

b. Resolução - passo 4/6 - continuação da classe **DadosTeste**:

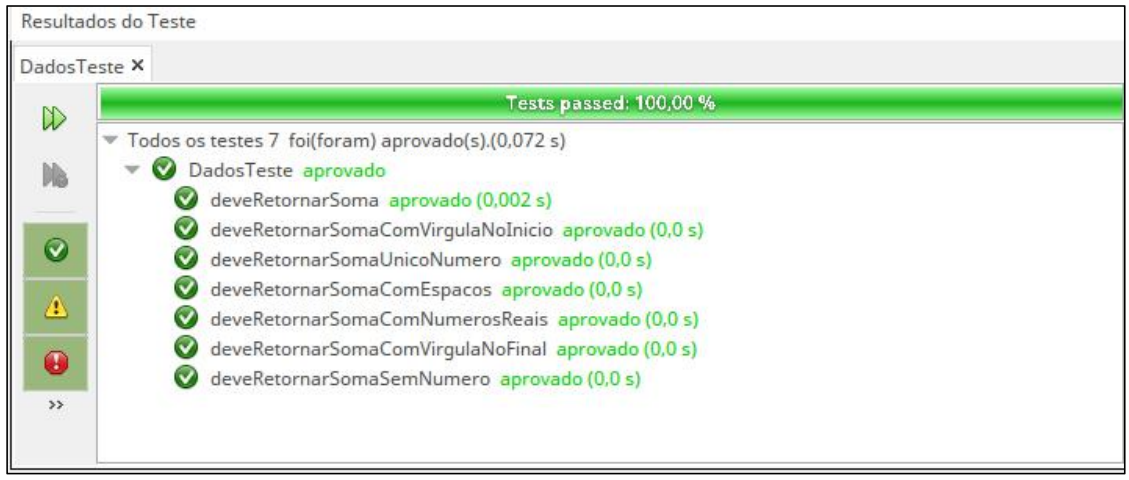
```
    @Test
    public void deveRetornarSomaUnicoNumero() {
        d.setDados("5");
        assertEquals(5.0, d.somaDados(), 0);
    }

    @Test
    public void deveRetornarSomaSemNumero() {
        d.setDados("");
        assertEquals(0.0, d.somaDados(), 0);
    }

    @Test
    public void deveRetornarSomaComEspacos() {
        d.setDados("5, 7, 9");
        assertEquals(21.0, d.somaDados(), 0);
    }
} // fim da classe DadosTeste
```

2.1.1 Teste unitário

b. Resolução - passo 5/6 - executando a classe **DadosTeste**:



2.1.1 Teste unitário

b. Resolução - passo 6/6 - usando a classe **Dados** no método main:

```
public static void main(String[] args) {
    Dados d = new Dados();
    Scanner sc = new Scanner(System.in);
    while(true) {
        System.out.println("Informe uma string de números "
                + "separados por vírgula (/s para sair)");
        String numeros = sc.nextLine();
        if(numeros.equalsIgnoreCase("/s")) break;
        d.setDados( numeros );
        System.out.println("Somatório: " + d.somaDados());
    }
} // fim do main
```

2.1.1 Teste unitário

b. Exemplo prático #2:

• Os casos de teste cobrem todas as possibilidades de entrada?

R: Não, porque são muitas e até infinitas neste caso.

• Casos de teste que não estão cobertos:

. Números negativos: `-5,-7,-2`

. Vírgulas a mais no meio da String: `1,3,,5,7` ou `1,3,,,5,7`

. Vírgulas a mais no meio da String com espaço: `1,3, ,, ,5,7`

. Ponto-vírgula invés de vírgula: `2; 4; 6`

. E vááários outros...

Teste de Software

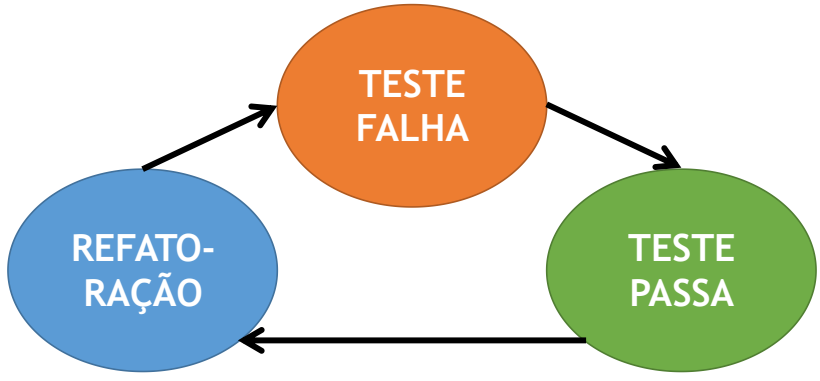
Prof. Dr. Alan Souza

alan.souza@unama.br

2020

2.1.2 Test-Driven Development (TDD)

1. Escreve e executa o teste, o teste falha;
2. Conserta o sistema e executa o teste, o teste passa;
3. Agora, melhora a solução (refatoração).



2.1.1 Teste unitário

Exercício de teste unitário com TDD e Github:

- Baixe e abra o projeto feito no Netbeans através do link a seguir (acessar o projeto pelo Github). Trata-se de um sistema que insere código, nome, email, CPF de clientes no banco de dados SQLite.
- Analise o projeto como um todo e use a metodologia do TDD para testar a classe Cliente que compõe o sistema.

2.1.1 Teste unitário

Exercício de teste unitário com TDD e Github:

Histórias dos Usuários para balizar os testes:

- O usuário deve cadastrar um novo cliente usando um email válido;
- O usuário deve cadastrar um novo cliente usando um CPF válido;
- O usuário deve informar o nome de um novo cliente com no mínimo 5 caracteres.
- O código do cliente deve ser gerado automaticamente pelo banco de dados SQLite e não pode se repetir.
- Teste todas as operações em relação aos dados do cliente:
  - seleção de um registro pelo ID e de todos os registros;
  - inserção, atualização e remoção de registro.

2.1.1 Teste unitário

Exercício de teste unitário com TDD e Github:

- Depois de finalizar o exercício, faça um **Pull Request (PR)** das novas funcionalidades através do Github.
- Link que ensina como fazer PR: http://bit.ly/tutorialgithub1
- Link do repositório do projeto no Github: http://bit.ly/testep3

Quem conseguir fazer PR adequadamente, ganhará1,0 ponto extra para a 1a avaliação!