



TESTES AUTOMATIZADOS

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

herculano.debiasi@unoesc.edu.br



TÓPICOS

- Bibliografia
- Motivação
- Introdução
- Tipos de testes
- Testes sem *framework*
- Testes automatizados
- Testes com o *framework* JUnit

Segunda Lei de Weinberg

“Se os engenheiros construíssem prédios como os programadores escrevem programas, o primeiro pica-pau destruiria toda a civilização”

Gerald Weinberg

BIBLIOGRAFIA

- ANICHE, Maurício. **Testes automatizados de software**: Um guia prático. São Paulo: Casa do Código, 2015.



MOTIVAÇÃO

- Importância de testes e inspeções

MOTIVAÇÃO

■ Ariane Rocket Goes Boom (1996): Projeto da Agência Nacional Europeia (ESA)

■ Custo

- US\$ 8 bilhões e 10 anos de desenvolvimento
- \$500 milhões (foguetes e carga)

■ Desastre

- Ariane 5, o mais novo foguete da Europa não-tripulado, foi intencionalmente destruído 37 segundos após seu lançamento em seu voo inaugural
- Também foram destruídos quatro satélites científicos para estudar como o campo magnético da Terra interage com os ventos solares

MOTIVAÇÃO

■ Ariane Rocket Goes Boom (1996): Projeto da Agência Nacional Europeia (ESA)

■ Causa

- O desligamento ocorreu quando o computador de orientação tentou converter a velocidade do foguete de 64-bits para um formato de 16 bits
- O número era muito grande, resultando em erro de estouro
- Falha no sistema (software) que calculava a trajetória e atitude, leva a uma pane nos sistemas adjacentes
- Estes, enviam sinais de diagnóstico para os motores que os interpreta como dados comuns.....
- Quando o sistema de orientação desligou, o controle passou para uma unidade idêntica redundante, que também falhou porque nele estava correndo o mesmo algoritmo



MOTIVAÇÃO

■ [SpaceX](#)



MOTIVAÇÃO

- Físico Ivair Gontijo, livro 'A Caminho de Marte' (2018)



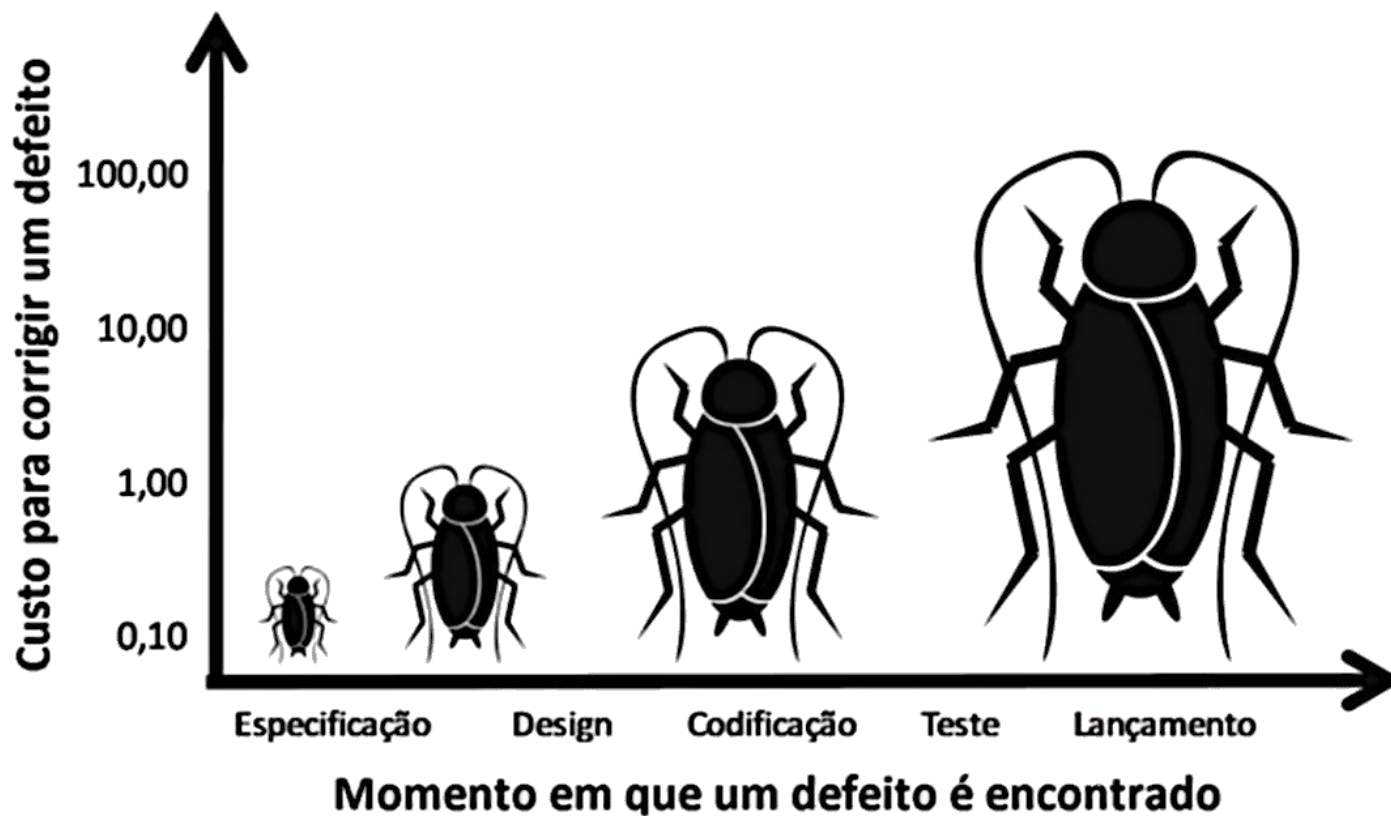
MOTIVAÇÃO

- Palestra da [Campus Party 2019](#) (São Paulo)

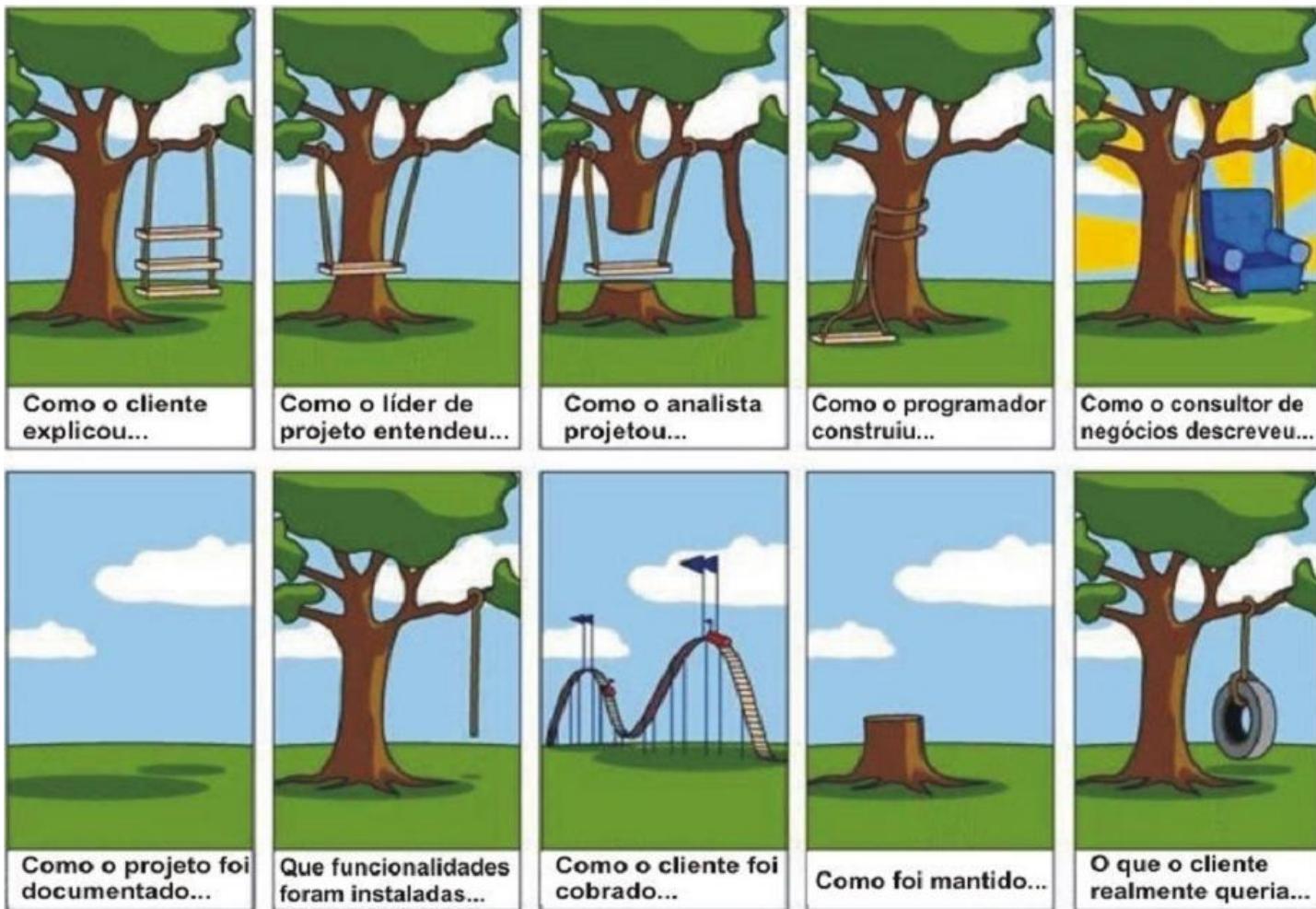


MOTIVAÇÃO

- Evolução do custo de um defeito



INTRODUÇÃO



(Paradigm Innovations, 2005)

INTRODUÇÃO



O que a equipe de testes recebeu



Qual é o plano de recuperação de desastre



Como o marketing anunciou



Quando foi entregue

INTRODUÇÃO

- Testar *software* é fundamental!!! Então por que não se testamos como se deve?
 - É trabalhoso...
 - É caro...
 - É chato...
 - É demorado...
 - É difícil...

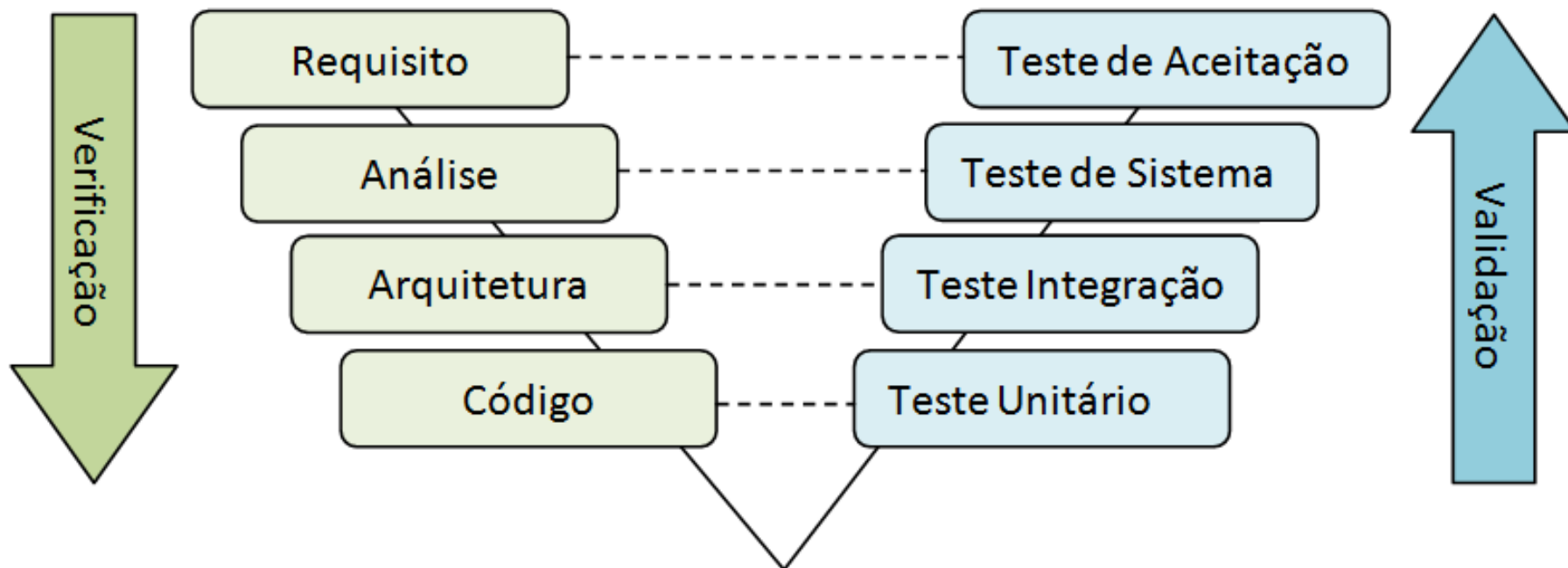
TIPOS DE TESTES

- Existem dezenas de tipos de testes, abaixo é mostrada uma lista parcial

Tipo de Teste	Descrição
Teste de unidade (unitário)	Teste do menor componente possível de um sistema – normalmente em orientação a objetos é uma classe ou método.
Teste de integração	<p>Garante que um ou mais componentes combinados (unidades) funcionam.</p> <p>Um teste de integração é composto por diversos testes de unidade, devendo testar a integração entre elas e não suas funcionalidades específicas.</p>
Teste de sistema	<p>Visa determinar se os componentes de um sistema computacional (o que pode envolver outros componentes de <i>software</i> e/ou de <i>hardware</i>) se integram bem e realizam as funcionalidades que lhe foram especificadas.</p> <p>Tem como objetivo executar o software sob o ponto de vista do seu usuário final, realizando o teste do sistema.</p>
Testes de aceitação	Testa se a solução está sendo bem vista/aceita pelo usuário, procurando determinar se o <i>software</i> funciona da maneira esperada, conforme consta na especificação dos requisitos.

TIPOS DE TESTES

- Escopo (abrangência) dos principais testes



TESTES SEM AUXÍLIO DE *FRAMEWORK*

- Classes
 - Modelo/domínio: Calculadora
 - Classe de teste correspondente CalculadoraTestesV1

```
modulo2 - Calculadora.java
1 public class Calculadora {
2     public int somar(int num1, int num2) {
3         return num1 + num2;
4     }
5 }
```

```
modulo2 - CalculadoraTestesV1.java
1 public class CalculadoraTestesV1 {
2
3     public static void main(String[] args) {
4         Calculadora calc = new Calculadora();
5         int soma = 0;
6
7         soma = calc.somar(41, 1);
8         System.out.println(soma);
9
10        soma = calc.somar(10, 0);
11        System.out.println(soma);
12
13        soma = calc.somar(5, -5);
14        System.out.println(soma);
15
16        soma = calc.somar(-2, -3);
17        System.out.println(soma);
18    }
19
20 }
```

TESTES SEM AUXÍLIO DE *FRAMEWORK*

■ Classe

■ Classe de teste correspondente CalculadoraTestesV2

```
modulo2 - CalculadoraTestesV2.java
1 public class CalculadoraTestesV2 {
2
3     public static void main(String[] args) {
4         Calculadora calc = new Calculadora();
5         int soma = 0;
6
7         soma = calc.somar(41, 1);
8         if (soma == 42) { System.out.println("Resultado " + soma + " está correto!"); }
9
10        soma = calc.somar(10, 0);
11        if (soma == 10) { System.out.println("Resultado " + soma + " está correto!"); }
12
13        soma = calc.somar(5, -5);
14        if (soma == 0) { System.out.println("Resultado " + soma + " está correto!"); }
15
16        soma = calc.somar(-2, -3);
17        if (soma == -5) { System.out.println("Resultado " + soma + " está correto!"); }
18    }
19
20 }
```

TESTES SEM AUXÍLIO DE *FRAMEWORK*

■ Classe

■ Classe de teste correspondente CalculadoraTestesV3

```
modulo2 - CalculadoraTestesV2.java
1 public class CalculadoraTestesV3 {
2
3     public static void main(String[] args) {
4         Calculadora calc = new Calculadora();
5         int soma = 0;
6
7         soma = calc.somar(41, 1);
8         if (soma == 42) { System.out.println("Resultado " + soma + " está correto!"); }
9
10        soma = calc.somar(10, 0);
11        if (soma == 10) { System.out.println("Resultado " + soma + " está correto!"); }
12
13        soma = calc.somar(5, -5);
14        if (soma == 0) { System.out.println("Resultado " + soma + " está correto!"); }
15
16        soma = calc.somar(-2, -3);
17        if (soma == -5) { System.out.println("Resultado " + soma + " está correto!"); }
18
19        soma = calc.somar(2147483647, 1);
20        if (soma == 2147483648L) { System.out.println("Resultado " + soma + " está correto!"); }
21    }
22
23 }
```

TESTES SEM AUXÍLIO DE

■ Classe

■ Classe de teste
correspondente
CalculadoraTestesV4

```
modulo2 - CalculadoraTestesV4.java
1 public class CalculadoraTestesV4 {
2
3     public static void main(String[] args) {
4         Calculadora calc = new Calculadora();
5         int soma = 0;
6
7         soma = calc.somar(41, 1);
8         if (soma == 42) {
9             System.out.println("Resultado " + soma + " está correto!");
10        } else {
11            System.out.println("Problema detectado!");
12        }
13
14        soma = calc.somar(10, 0);
15        if (soma == 10) {
16            System.out.println("Resultado " + soma + " está correto!");
17        } else {
18            System.out.println("Problema detectado!");
19        }
20
21        soma = calc.somar(5, -5);
22        if (soma == 0) {
23            System.out.println("Resultado " + soma + " está correto!");
24        }
25
26        soma = calc.somar(-2, -3);
27        if (soma == -5) {
28            System.out.println("Resultado " + soma + " está correto!");
29        } else {
30            System.out.println("Problema detectado!");
31        }
32
33        soma = calc.somar(2147483647, 1);
34        if (soma == 2147483648L) {
35            System.out.println("Resultado " + soma + " está correto!");
36        } else {
37            System.out.println("Erro - Problema detectado!");
38        }
39    }
40
41 }
```

TESTES AUTOMATIZADOS

- Testes automatizados de unidades (unitários) testam o menor trecho de código possível de um sistema, isso significa um objeto ou método em POO
- Princípio F.I.R.S.T.
 - **Fast** (rápido): Se forem lentos, não iremos executá-los com a frequência necessária
 - **Independent / Isolated** (independente/isolado): Cada teste é independente dos outros
 - **Repeatable** (repetível/determinístico): Deve apresentar sempre o mesmo resultado
 - **Self-verifying** (autoverificável/autovalidável): Devem ser autoexplicativos (passou/falhou)
 - **Thorough and Timely**
 - Completos (cobertura), abrangendo várias condições e ‘caminhos’ diferentes do código
 - Feitos e executados na hora certa, devendo ser criados na hora do desenvolvimento ou então quando aparecem os *bugs*

TESTES AUTOMATIZADOS

■ Padrão AAA (*triple A*)

- *Arrange* (cenário): Consiste na preparação do teste, como configuração, criação de variáveis, instanciação de objetos, etc
- *Act* (execução): Execução propriamente dita
- *Assert* (verificação): Onde se faz o *assert* (asserção), procurando verificar se a operação realizada na etapa anterior (*act*) surtiu o resultado esperado, determinando desta forma se o teste passou ou falhou

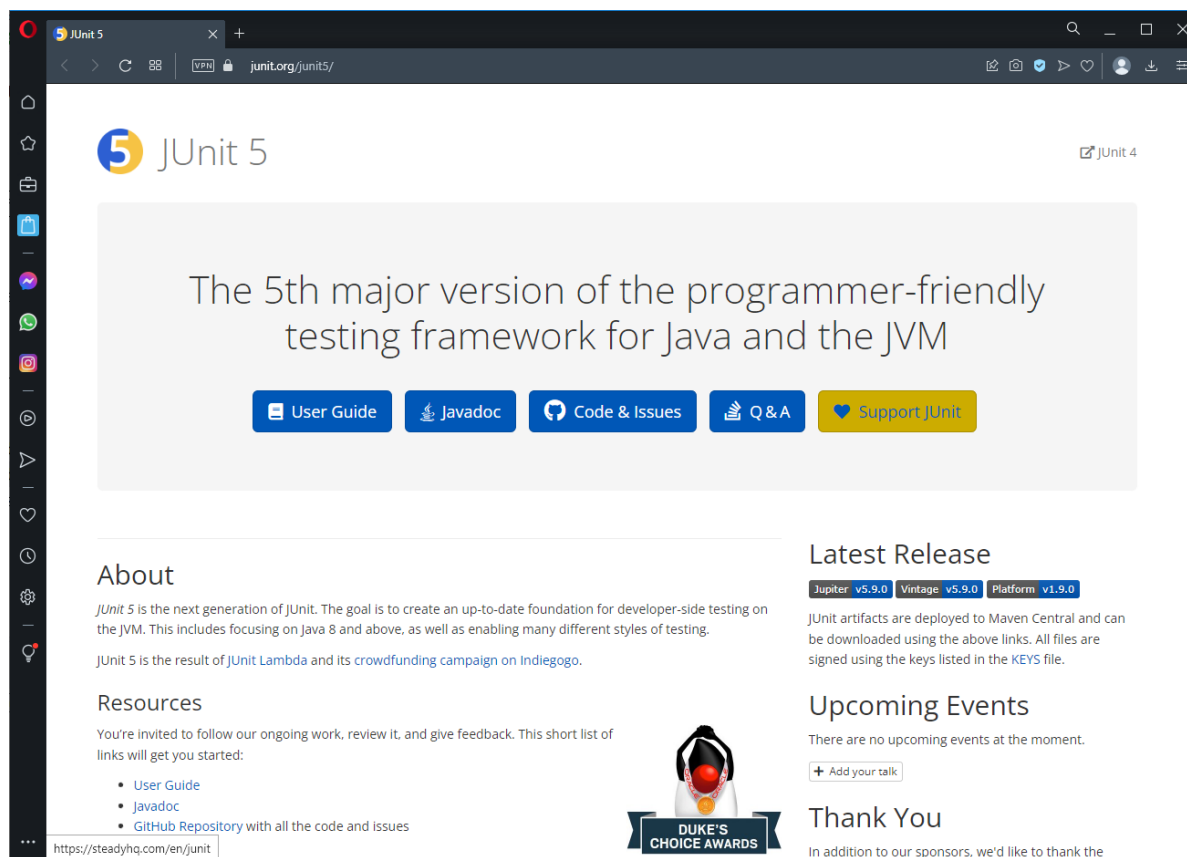


TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

- **JUnit** é um *framework* de testes *open-source*, criado originalmente por Kent Beck e Erich Gamma em 1995 para facilitar a criação, desenvolvimento e execução de testes unitários em código Java



- É a biblioteca padrão para a escrita de testes automatizados em Java



TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

■ Características

- Ele fornece um suporte completo para construir os testes e aplicações gráficas e em modo console para executar os testes criados
- Foco em testes de unidade (testes unitários)
- Pode verificar se cada unidade de código funciona da forma esperada
- Facilita a criação, execução automática de testes e a apresentação dos resultados
- É orientado a objetos

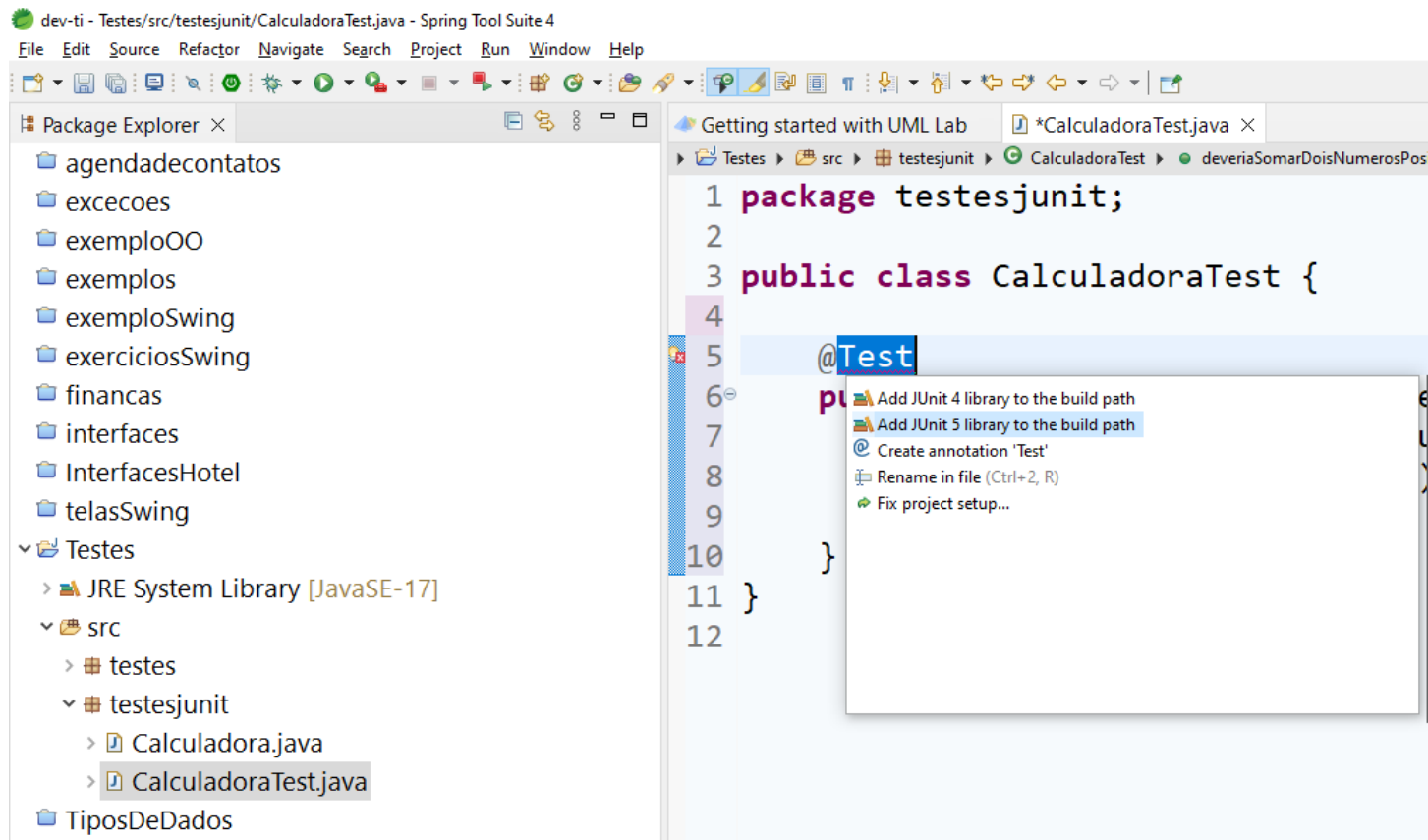
TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

- Exemplo de método de teste com a anotação `@Test`

```
1 package testesjunit;
2
3 public class CalculadoraTest {
4
5     @Test
6     public void deveriaSomarDoisNumerosPositivos() {
7         // Cenário (arrange)
8         Calculadora calc = new Calculadora();
9
10        // Execução (act)
11        int soma = calc.somar(41, 1);
12
13    }
14 }
```

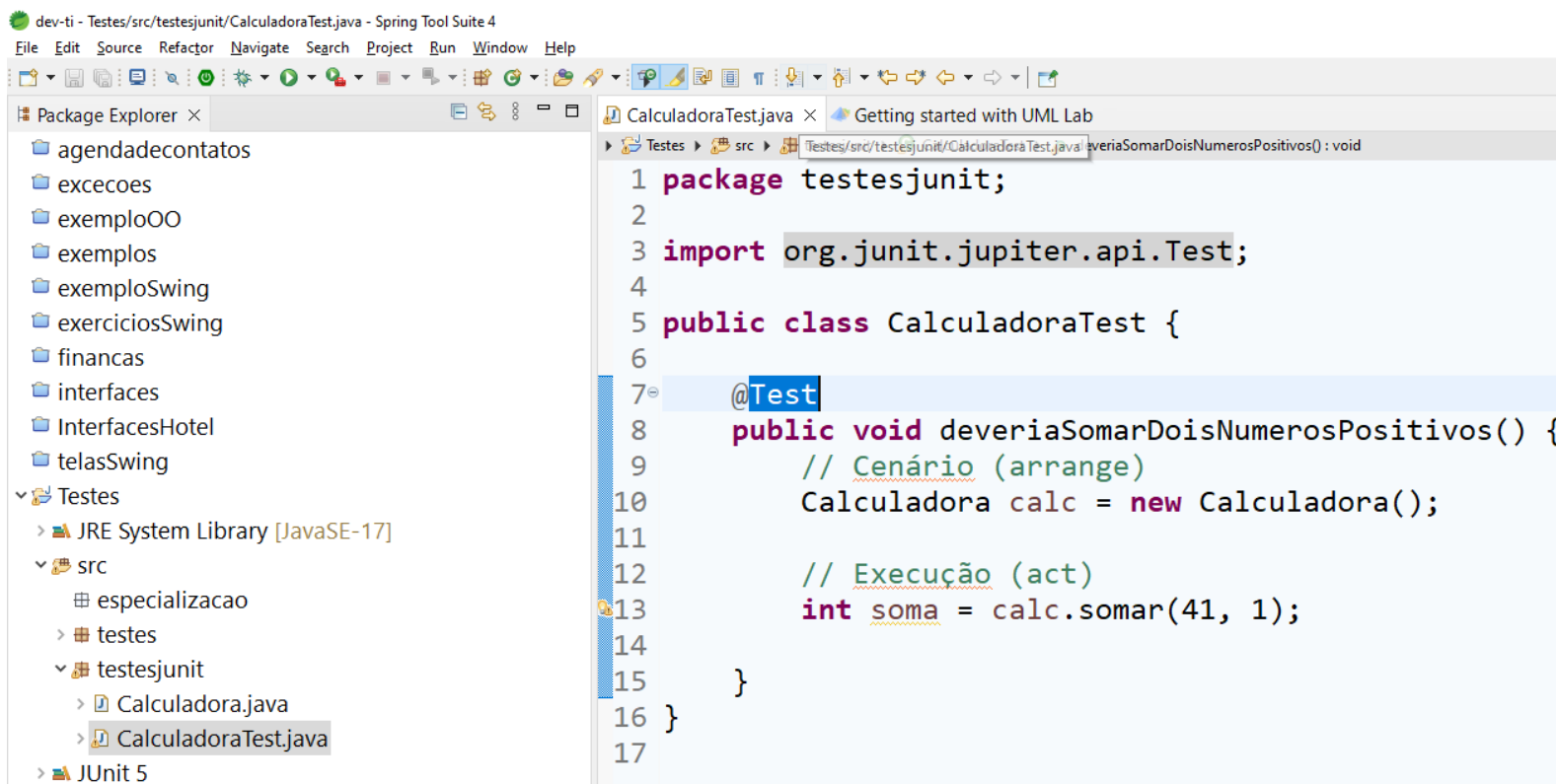
TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

■ Adicionando suporte para o JUnit 5



TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

■ Adicionando suporte para o JUnit 5



The screenshot shows the Spring Tool Suite 4 IDE. On the left, the Package Explorer displays a project structure with folders like 'agendadecontatos', 'excecoes', 'exemploOO', 'exemplos', 'exemploSwing', 'exerciciosSwing', 'financas', 'interfaces', 'InterfacesHotel', 'telasSwing', and a 'Testes' folder. The 'Testes' folder is expanded, showing 'JRE System Library [JavaSE-17]', 'src', and 'testesjunit'. The 'testesjunit' folder is expanded, showing 'Calculadora.java' and 'CalculadoraTest.java'. The main editor displays the source code of 'CalculadoraTest.java'.

```
dev-ti - Testes/src/testesjunit/CalculadoraTest.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer x
  agendadecontatos
  excecoes
  exemploOO
  exemplos
  exemploSwing
  exerciciosSwing
  financas
  interfaces
  InterfacesHotel
  telasSwing
  Testes
    JRE System Library [JavaSE-17]
    src
      especializacao
      testes
      testesjunit
        Calculadora.java
        CalculadoraTest.java
    JUnit 5

CalculadoraTest.java x Getting started with UML Lab
  Testes src Testes/src/testesjunit/CalculadoraTest.java deveriaSomarDoisNumerosPositivos(): void
1 package testesjunit;
2
3 import org.junit.jupiter.api.Test;
4
5 public class CalculadoraTest {
6
7     @Test
8     public void deveriaSomarDoisNumerosPositivos() {
9         // Cenário (arrange)
10        Calculadora calc = new Calculadora();
11
12        // Execução (act)
13        int soma = calc.somar(41, 1);
14
15    }
16 }
17
```

TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

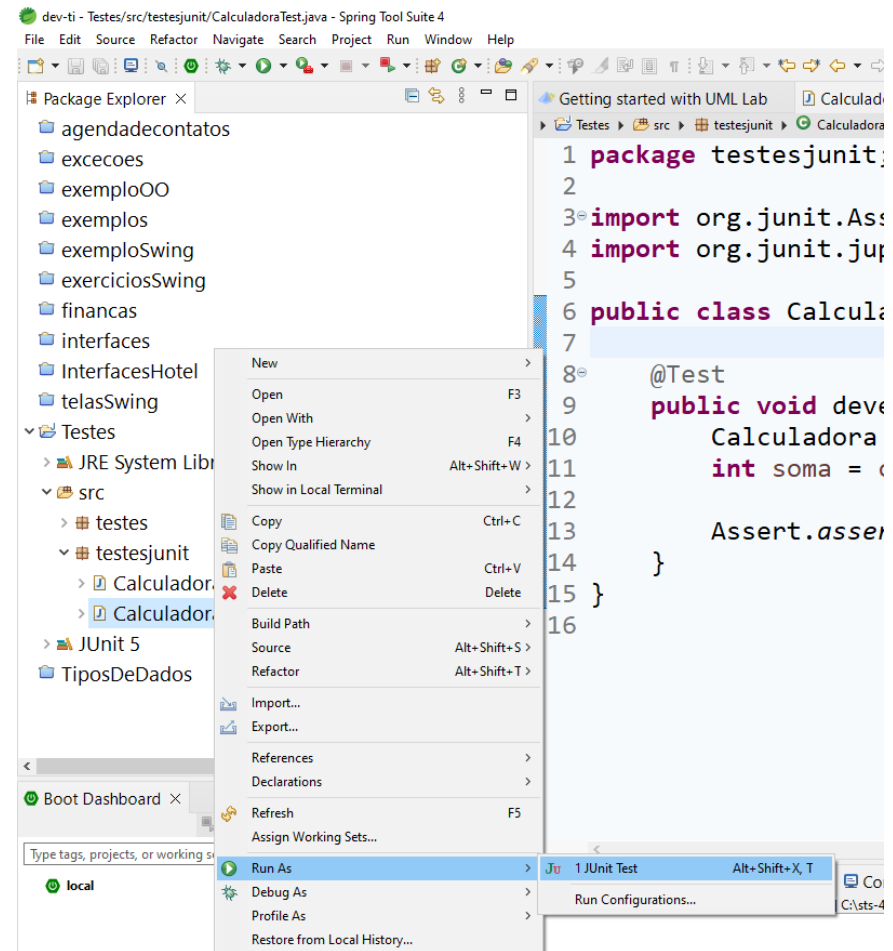
- Realizando a verificação com assertivas (Assert)

```
assertEquals(esperado, calculado)
```

```
1 package testesjunit;
2
3 import org.junit.jupiter.api.Test;
4 import org.junit.Assert;
5
6 public class CalculadoraTest {
7
8     @Test
9     public void deveriaSomarDoisNumerosPositivos() {
10         // Cenário (arrange)
11         Calculadora calc = new Calculadora();
12
13         // Execução (act)
14         int soma = calc.somar(41, 1);
15
16         // Verificação (assert)
17         Assert.assertEquals(42, soma);
18     }
19 }
```

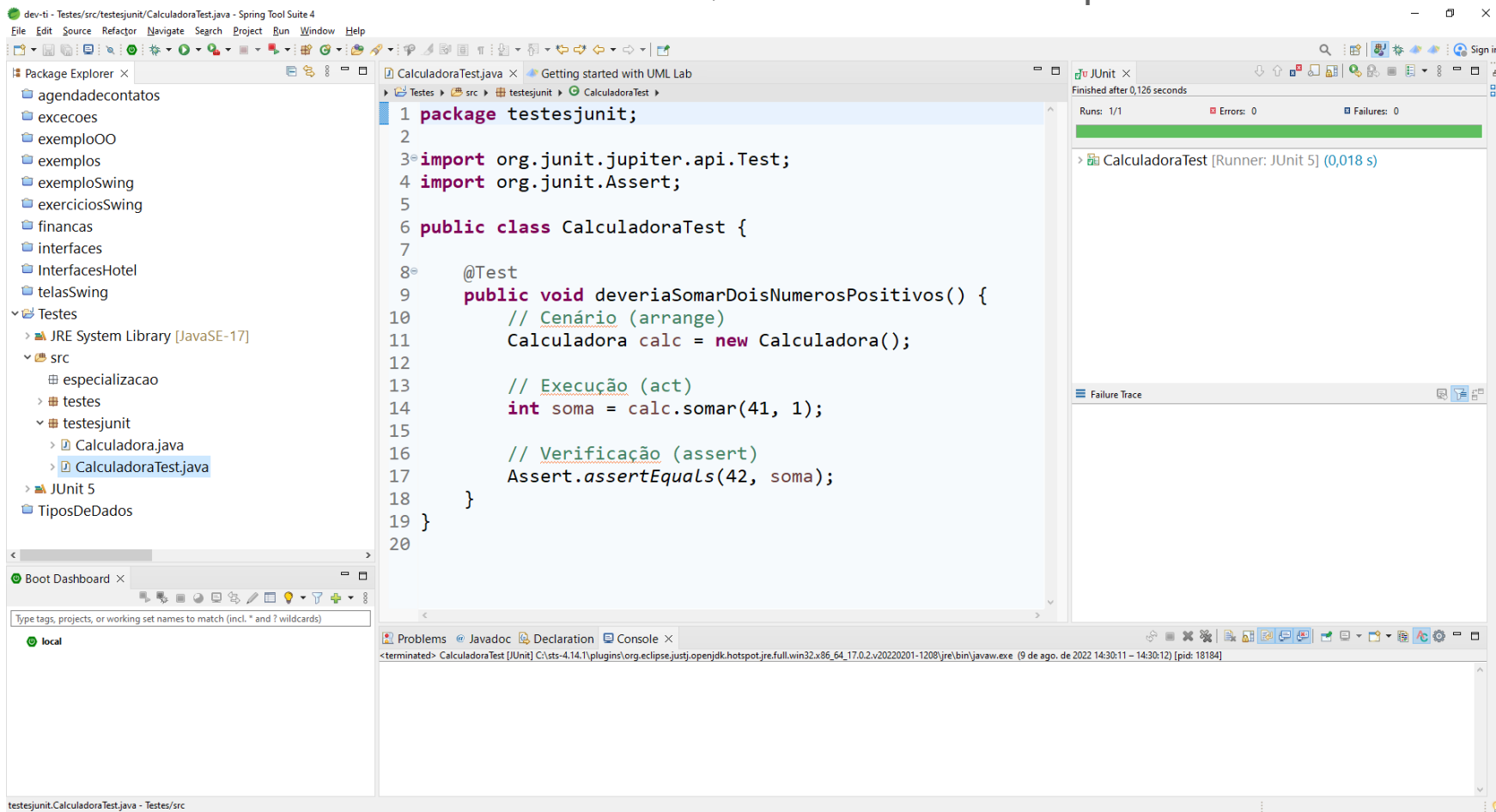
TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

- Para executar a unidade de teste basta clicar com o botão direito sobre ela e escolher *Run As* → *JUnit Test*
- Ou então clicar no menu *Run* e então *Run As* → *JUnit Test*



TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

Se o teste foi realizado com sucesso, uma barra verde irá aparecer



The screenshot displays an IDE interface with the following components:

- Package Explorer:** Shows a project structure with folders like 'agendadecontatos', 'excecoes', 'exemploOO', 'exemplos', 'exemploSwing', 'exerciciosSwing', 'financas', 'interfaces', 'InterfacesHotel', 'telasSwing', and a 'Testes' folder containing 'JUnit 5' and 'TiposDeDados'.
- Editor:** Displays the code for 'CalculadoraTest.java'. The code includes package declarations, imports for JUnit, and a test method 'deveriaSomarDoisNumerosPositivos()' that uses 'Arrange-Act-Assert'.
- JUnit Runner:** A panel on the right showing the test execution results. It indicates 'Finished after 0,126 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. A green progress bar is visible, and the test name 'CalculadoraTest [Runner: JUnit 5] (0,018 s)' is listed.
- Console:** At the bottom, it shows a 'terminated' message for the test execution.

TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

Se o teste foi realizado com sucesso, uma barra verde irá aparecer

The screenshot displays an IDE window titled "dev-ti - Testes/src/testesjunit/CalculadoraTest.java - Spring Tool Suite 4". The main editor shows the source code for `CalculadoraTest.java`, which includes imports for `org.junit.jupiter.api.Test` and `org.junit.Assert`, and a test method `deveriaSomarDoisNumerosPositivos()` that uses `Assert.assertEquals(40, soma)`. The Package Explorer on the left shows the project structure, including the `Testes` package and the `CalculadoraTest.java` file. The JUnit runner window on the right indicates the test was "Finished after 0,124 seconds" with "Runs: 1/1", "Errors: 0", and "Failures: 1". The failure trace shows a `java.lang.AssertionError: expected:<40> but was:<42>`. The console at the bottom shows the test execution details, including the path to the JUnit runner and the PID.

```
1 package testesjunit;
2
3 import org.junit.jupiter.api.Test;
4 import org.junit.Assert;
5
6 public class CalculadoraTest {
7
8     @Test
9     public void deveriaSomarDoisNumerosPositivos() {
10         // Cenário (arrange)
11         Calculadora calc = new Calculadora();
12
13         // Execução (act)
14         int soma = calc.somar(41, 1);
15
16         // Verificação (assert)
17         Assert.assertEquals(40, soma);
18     }
19 }
20
```

JUnit Summary:
Finished after 0,124 seconds
Runs: 1/1
Errors: 0
Failures: 1

Failure Trace:
java.lang.AssertionError: expected:<40> but was:<42>
at testesjunit.CalculadoraTest.deveriaSomarDoisNumerosPositivos()
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Console:
<terminated> CalculadoraTest [JUnit] C:\sts-4.14.1\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (9 de ago. de 2022 14:29:20 - 14:29:21) [pid: 17968]

TESTES AUTOMATIZADOS COM O *FRAMEWORK* JUNIT

- Exercícios: Implemente os seguintes testes automatizados nesta mesma unidade
 - `Somar(10, 0);`
 - `Somar(5, -5);`
 - `Somar(-2, -3);`
- O problema do estouro do número inteiro será resolvido na próxima aula