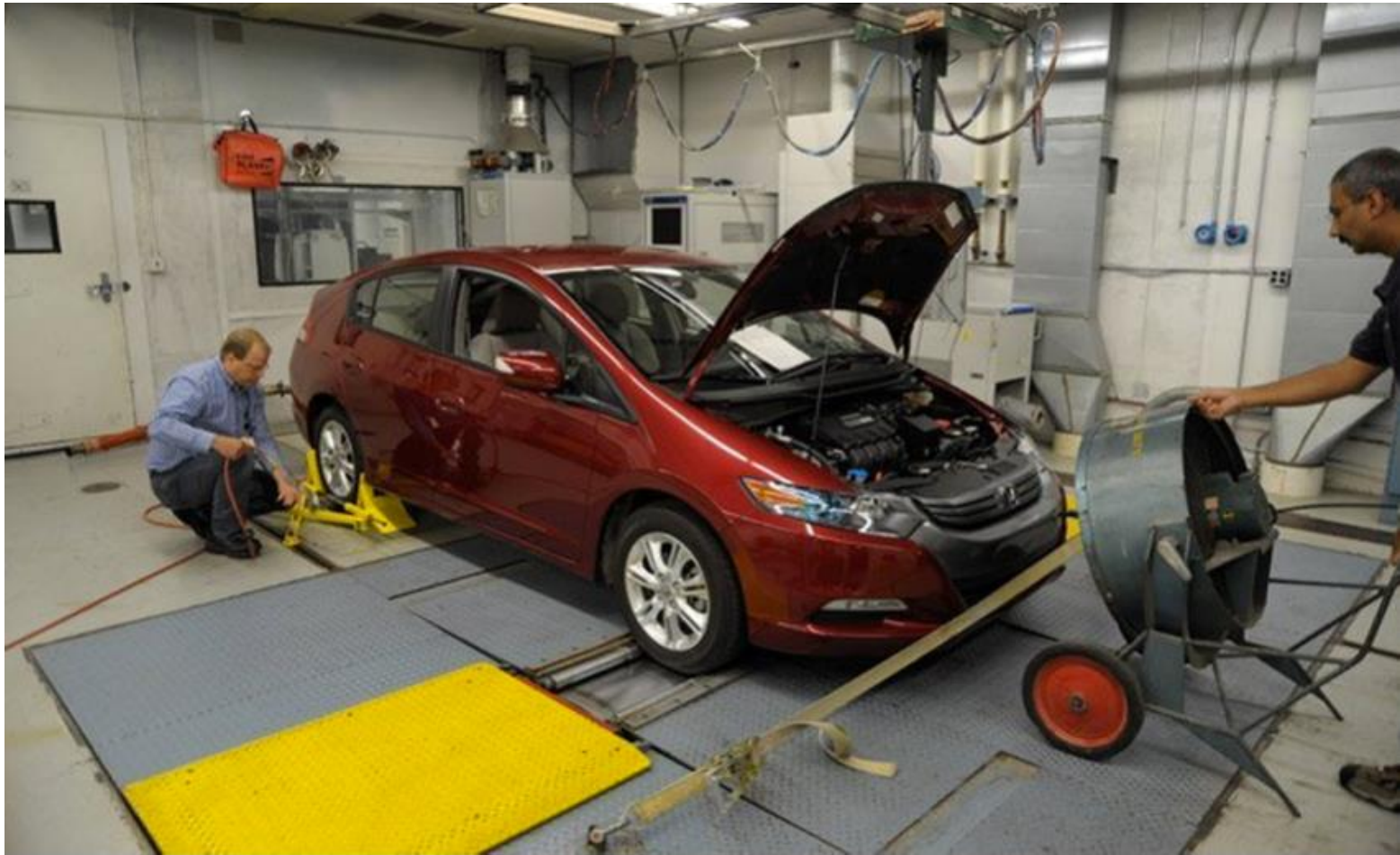


# Introdução aos Testes Aplicacionais


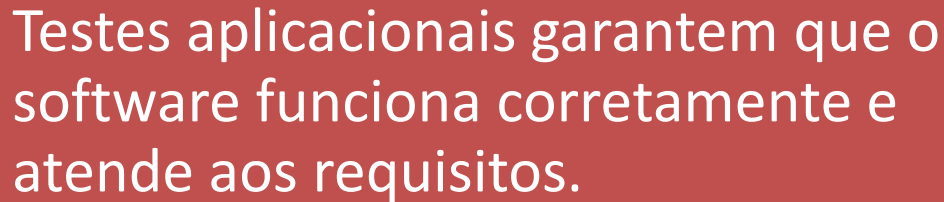
UFCD 10789 - Metodologias de Desenvolvimento  
de Software

# Testes



# O que são Testes Aplicacionais?

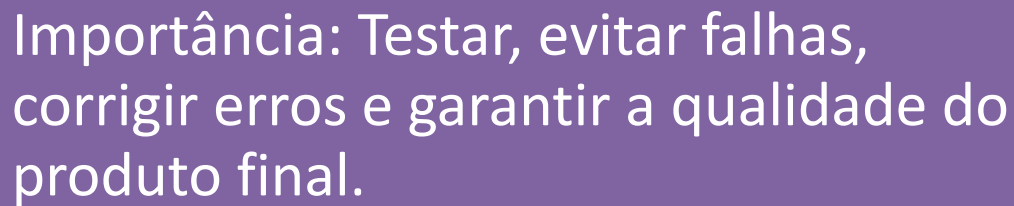
Testes aplicacionais garantem que o software funciona corretamente e atende aos requisitos.



Tipos de Testes: Unitário, Integração, Carga, Stress, Aceitação



Importância: Testar, evitar falhas, corrigir erros e garantir a qualidade do produto final.



# Teste Unitário

## **O Que É o Teste Unitário?**

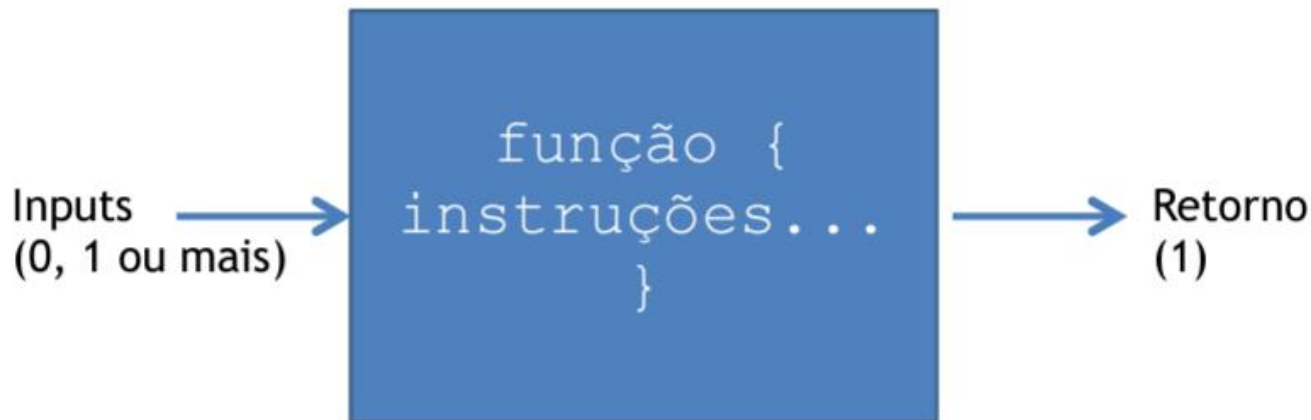
O teste unitário verifica se pequenas partes individuais do programa (como funções ou métodos) estão a funcionar corretamente. Testa-se uma parte de cada vez, de forma isolada.

## **Por Que É Importante?**

Ajuda a encontrar erros no código logo no início, durante o desenvolvimento. Quanto mais cedo os problemas forem identificados, mais fácil e rápido é corrigi-los.

# Teste Unitário

Como posso garantir que uma função faz o que é suposto fazer?



# Teste Unitário

Exemplo para a função  
**int soma(int num1, int num2) em java**

Cenário	Entrada de dados	Retorno(saída de dados)
Soma normal	num1=5, num2=9	9
Soma com 0	num1 = 0, num2 = 8	8
Soma com números negativos	num1 = -7, num2 = -5	-12

# Teste Unitário

© Betânia Queta 2024

```
public class Main {  
  
    // Função que realiza a soma  
    6 usages  
    public static int soma(int num1, int num2) {  
        return num1 + num2;  
    }  
  
    public static void main(String[] args) {  
        // Cenário 1: Soma normal  
        int resultado1 = soma(5, 9);  
        System.out.println("Soma normal (5 + 9): " + resultado1);  
  
        // Cenário 2: Soma com 0  
        int resultado2 = soma(0, 8);  
        System.out.println("Soma com 0 (0 + 8): " + resultado2);  
  
        // Cenário 3: Soma com números negativos  
        int resultado3 = soma(-7, -5);  
        System.out.println("Soma com números negativos (-7 + -5): " + resultado3);  
    }  
}
```

# Teste Unitário

© Betânia Queta 2024

```
import static org.junit.Assert.*;
import org.junit.Test;

public class MainTest {

    // Cenário 1: Soma normal
    @Test
    public void testSomaNormal() {
        int resultado = Main.soma(5, 9);
        assertEquals("expected: 14, resultado);", resultado);
    }

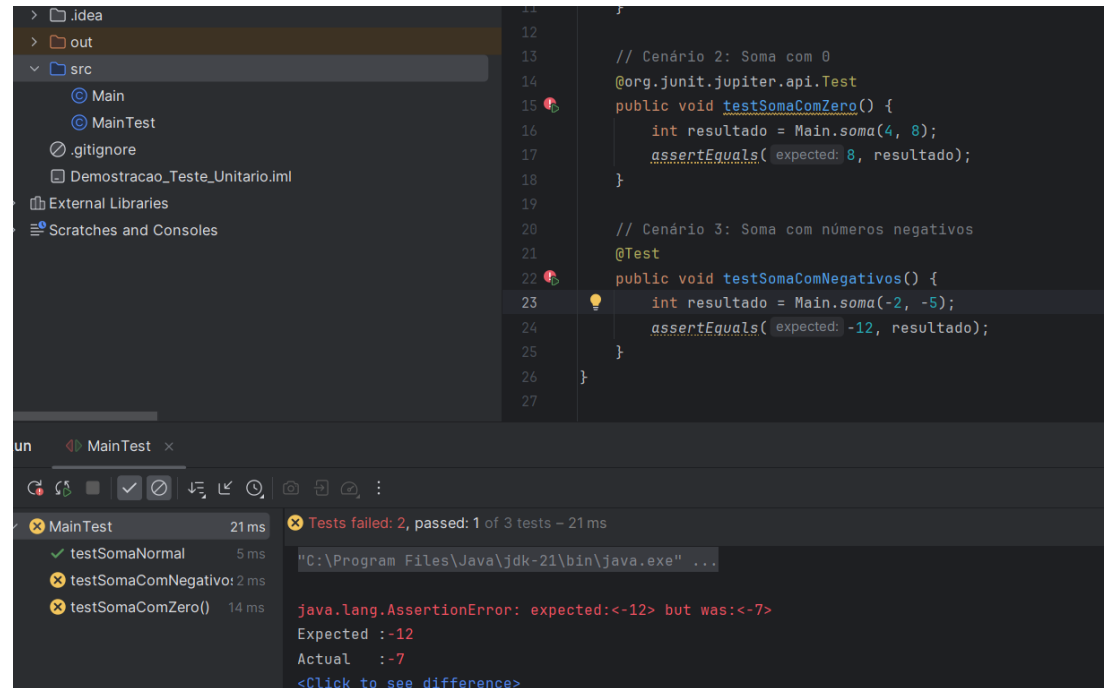
    // Cenário 2: Soma com 0
    @org.junit.jupiter.api.Test
    public void testSomaComZero() {
        int resultado = Main.soma(0, 8);
        assertEquals("expected: 8, resultado);", resultado);
    }

    // Cenário 3: Soma com números negativos
    @Test
    public void testSomaComNegativos() {
        int resultado = Main.soma(-7, -5);
        assertEquals("expected: -12, resultado);", resultado);
    }
}
```



# Teste Unitário

- © Betânia Queta 2024



```
11  
12  
13 // Cenário 2: Soma com 0  
14 @org.junit.jupiter.api.Test  
15 public void testSomaComZero() {  
16     int resultado = Main.soma(4, 8);  
17     assertEquals(expected: 8, resultado);  
18 }  
19  
20 // Cenário 3: Soma com números negativos  
21 @Test  
22 public void testSomaComNegativos() {  
23     int resultado = Main.soma(-2, -5);  
24     assertEquals(expected: -12, resultado);  
25 }  
26 }  
27
```

Test Results:

Test Name	Duration	Status
MainTest	21 ms	Failed
testSomaNormal	5 ms	Passed
testSomaComNegativo	2 ms	Failed
testSomaComZero	14 ms	Failed

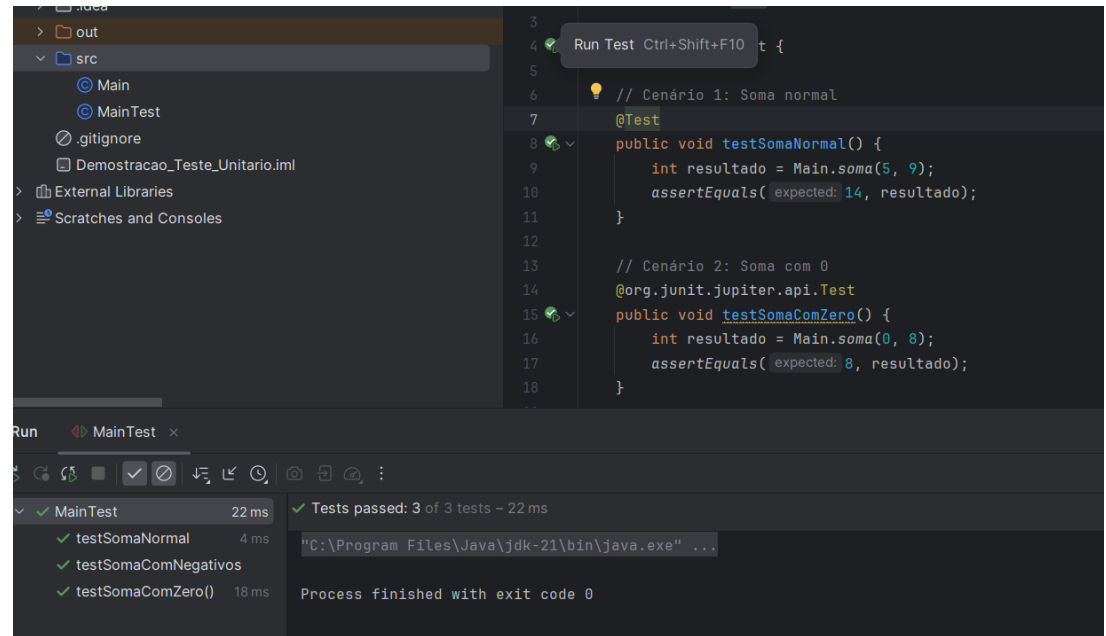
Test Summary: Tests failed: 2, passed: 1 of 3 tests - 21 ms

Error Message:

```
java.lang.AssertionError: expected:<-12> but was:<-7>  
Expected : -12  
Actual   : -7  
<Click to see difference>
```

# Teste Unitário

- © Betânia Queta 2024



```
3  
4 ✓ Run Test Ctrl+Shift+F10 t {  
5  
6 // Cenário 1: Soma normal  
7 @Test  
8 ✓ public void testSomaNormal() {  
9     int resultado = Main.soma(5, 9);  
10    assertEquals(expected: 14, resultado);  
11 }  
12  
13 // Cenário 2: Soma com 0  
14 @org.junit.jupiter.api.Test  
15 ✓ public void testSomaComZero() {  
16     int resultado = Main.soma(0, 8);  
17     assertEquals(expected: 8, resultado);  
18 }  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

Run MainTest x

✓ MainTest 22 ms

- ✓ testSomaNormal 4 ms
- ✓ testSomaComNegativos
- ✓ testSomaComZero() 18 ms

✓ Tests passed: 3 of 3 tests - 22 ms

"C:\Program Files\Java\jdk-21\bin\java.exe" ...

Process finished with exit code 0

# Teste de Carga




- O teste de carga serve para ver como um software se comporta quando muitas pessoas o utilizam ao mesmo tempo. A ideia é aumentar o número de acessos gradualmente para descobrir o quanto o programa aguenta antes de começar a ficar lento ou travar.

# Teste de Carga

**Objetivo principal:** Encontrar o limite do software, ou seja, até quantos utilizadores ele consegue atender sem problemas.



**O que é verificado?:** Podemos descobrir se o problema está no código do programa, no hardware (computadores ou servidores), ou se o tempo que o programa demora para responder é muito longo.



**Importante:** Esse teste não é feito para encontrar erros no funcionamento do programa, mas sim para ver como ele lida com muitos utilizadores ao mesmo tempo.

# Teste de Carga

## Algumas Respostas Encontradas com o Teste de Carga:

### 1. Throughput (Taxa de Transferência):

O throughput é a quantidade de dados que o sistema consegue processar por segundo. Com o teste de carga, podemos descobrir o volume de dados que o sistema consegue gerir antes de começar a ficar mais lento.

### 2. Limite de Capacidade do Hardware:

O teste também mostra até que ponto o hardware (computadores e servidores) consegue suportar o sistema antes de começar a ter problemas, como demorar a responder ou até parar.

### 3. Requisitos de Balanceamento de Carga:

Podemos perceber se o sistema precisa de dividir a carga entre vários servidores para garantir que nenhum fica sobrecarregado. Isto ajuda a melhorar a performance quando há muitos utilizadores.

### 4. Número de Transações ou Utilizadores Simultâneos:

Uma das informações mais importantes que o teste de carga nos dá é quantas transações (operações) o sistema pode realizar ao mesmo tempo, ou quantos utilizadores podem usar o sistema simultaneamente sem que ele fique lento ou deixe de funcionar. Por exemplo, num site, podemos ver quantos utilizadores conseguem aceder ao mesmo tempo sem problemas.

# Como São Feitos os Testes de Carga na Prática:

## **Simular Utilizadores:**

Usa-se uma ferramenta para imitar muitas pessoas a usar o sistema ao mesmo tempo, como se fossem 1000 utilizadores a aceder a um site.

Ferramentas mais usadas: *JMeter*, *LoadRunner*.

## **Aumentar Gradualmente:**

O número de utilizadores simulados vai aumentando aos poucos, para ver até onde o sistema aguenta sem ficar lento ou travar.

## **Recolher Dados**

Durante o teste, recolhem-se informações como o tempo que o sistema demora a responder e quantos utilizadores ele suporta ao mesmo tempo.

## **Analisar e Melhorar**

No final, os resultados mostram se o sistema precisa de ajustes no código ou nos servidores para suportar mais utilizadores.

# Como São Feitos os Testes de Carga na Prática:

- <https://youtu.be/IEm3UCfUn2s?si=XinXMWuBMlazynoq>



# Teste de Stress

## **Testar Sobrecarga no Sistema**

No teste de stress, colocamos o sistema a funcionar com muito mais utilizadores ou tarefas do que ele aguenta normalmente, para ver até onde ele resiste antes de falhar.

## **Testar Falhas e Recuperação**

Além de sobrecarregar, fazemos o sistema falhar de propósito, como desligar um servidor ou o banco de dados. O objetivo é ver se o sistema consegue voltar a funcionar sozinho depois da falha.

## **Teste de Pico (Spike Test)**

Um tipo especial de teste de stress é o spike test, onde fazemos várias ações ao mesmo tempo. Por exemplo, todos os funcionários de um call center a fazer login ao mesmo tempo numa troca de turnos.

## **Avaliar o Tempo de Recuperação**

Depois de todos os testes, verificamos quanto tempo o sistema demora para voltar ao normal, sem precisar de ajuda.



# Teste de Integração

## O Que É o Teste de Integração?

O teste de integração verifica se as diferentes partes de um sistema (como módulos ou componentes) funcionam bem juntas. Mesmo que cada parte funcione sozinha, é importante garantir que elas também funcionem corretamente quando combinadas.

## Por Que É Importante?

Mesmo que cada parte do software funcione de forma isolada, quando juntamos todas, pode haver problemas de comunicação entre elas. O teste de integração ajuda a encontrar e resolver esses problemas.

## Exemplo Prático

Se um sistema de login (onde os utilizadores entram com nome e senha) precisa comunicar-se com uma base de dados (onde as informações estão guardadas), o teste de integração garante que essa ligação entre as duas partes está a funcionar corretamente.

## Resultado do Teste

Se o teste mostrar que todas as partes estão a funcionar bem juntas, o sistema pode passar para outros testes, como os de aceitação.



# Teste de Aceitação

## **O Que É o Teste de Aceitação?**

O teste de aceitação é feito para garantir que o sistema ou aplicação atende às necessidades do cliente ou do utilizador final. É o teste que verifica se o produto está pronto para ser usado.

## **Quem Faz o Teste?**

Normalmente, este teste é feito pelo cliente ou pelos utilizadores finais, que verificam se o sistema faz tudo o que foi pedido e esperado.

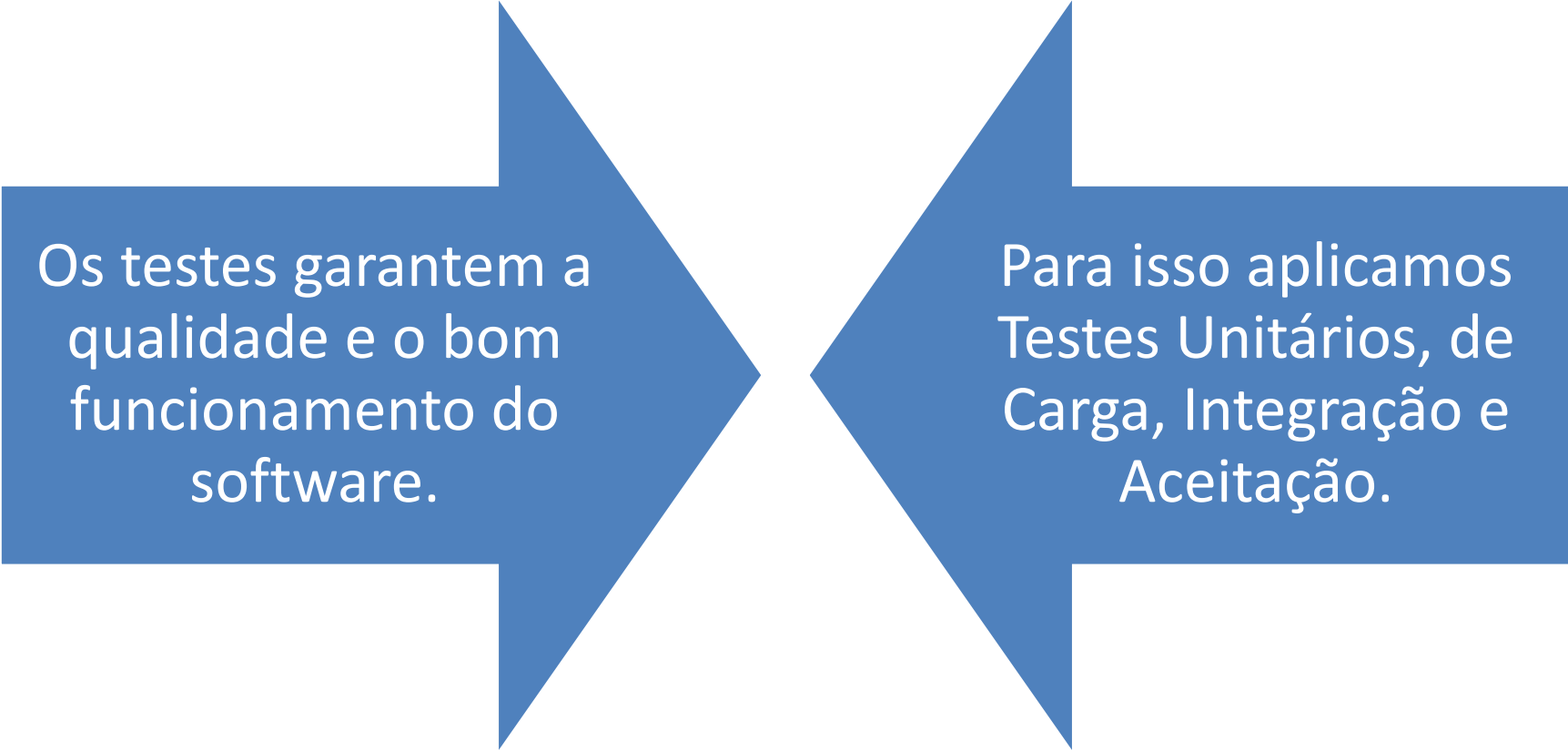
## **Exemplo Prático**

Num site de compras online, o teste de aceitação verifica se os clientes conseguem adicionar produtos ao carrinho, fazer o pagamento e receber a confirmação de compra corretamente.

## **Resultado do Teste**

Se o sistema passar no teste de aceitação, quer dizer que está pronto para ser lançado ou entregue. Se houver problemas, são feitas correções antes de o sistema ser usado.

# Conclusão



Os testes garantem a qualidade e o bom funcionamento do software.

The diagram consists of two large blue arrows pointing towards each other, meeting at a central point. The left arrow contains the text 'Os testes garantem a qualidade e o bom funcionamento do software.' and the right arrow contains the text 'Para isso aplicamos Testes Unitários, de Carga, Integração e Aceitação.'

Para isso aplicamos Testes Unitários, de Carga, Integração e Aceitação.