

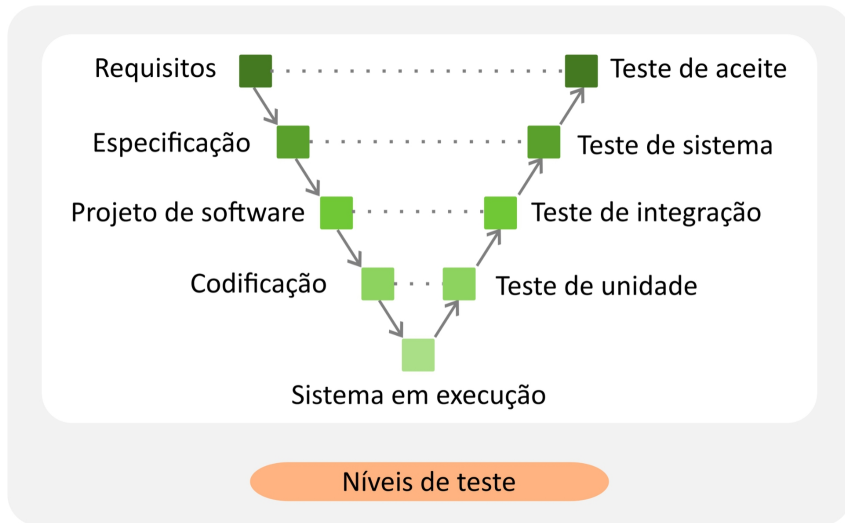
Simulação e Teste de Software (CC8550)  
Aula 05 - Níveis de testes:  
Aprofundamento em Testes de Unidade e Integração

Prof. Luciano Rossi

Ciência da Computação  
Centro Universitário FEI

1º Semestre de 2025

## Modelo V que descreve os níveis de teste



## Definição

- ▶ O teste de unidade objetiva a verificação de erros existentes nas **unidades** de projeto;
- ▶ É importante utilizar as informações do **documento de projeto** que servirão de guia para sua aplicação;
- ▶ O teste de unidade é uma técnica de teste de **caixa branca**, podendo ser realizado em paralelo sobre diferentes módulos.

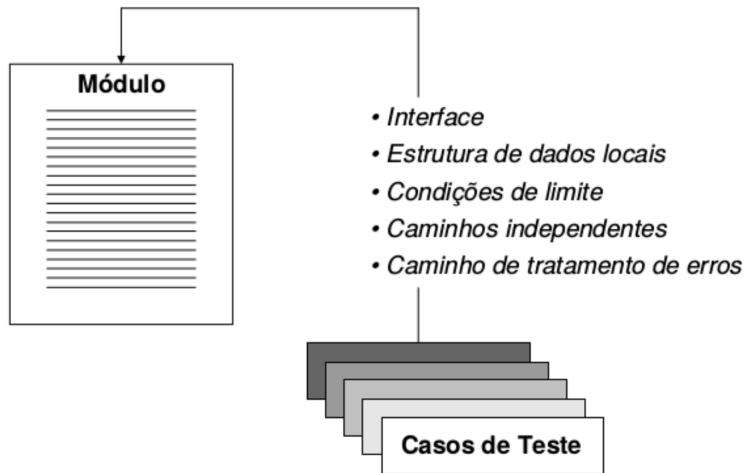
### O que são as unidades de projeto?

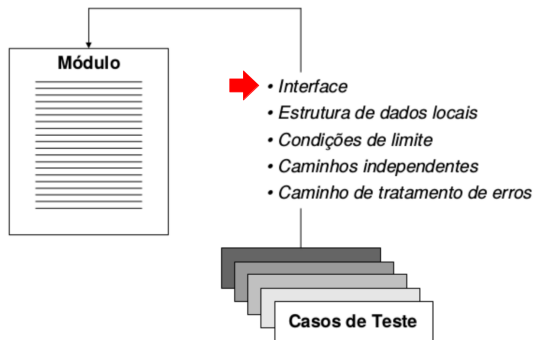
- ▶ **Função:** Bloco de código que executa uma tarefa específica e pode retornar um valor.
- ▶ **Classe:** Estrutura que encapsula atributos e métodos relacionados, representando objetos no paradigma orientado a objetos.
- ▶ **Módulo:** Arquivo ou conjunto de funções e classes agrupadas para modularizar o software.
- ▶ **Componente:** Unidade maior que pode conter múltiplos módulos e é projetada para reutilização e independência dentro do sistema.

### O que são os documentos de projeto?

- ▶ Especificação de Requisitos
- ▶ Documento de Arquitetura de Software (DAS)
- ▶ Diagramas UML
- ▶ Design Detalhado do Módulo
- ▶ Casos de Uso e Cenários de Teste
- ▶ Plano de Testes de Unidade

# Teste de Unidade



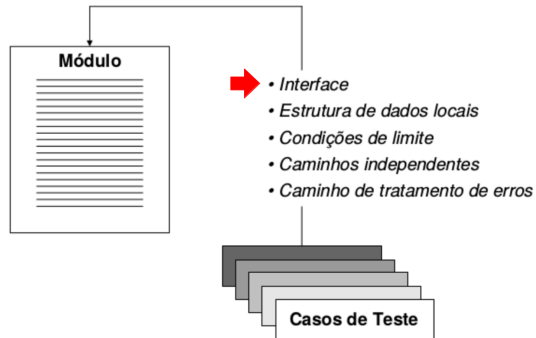


## Interface

- ▶ É o ponto de comunicação entre o módulo e outros componentes;
- ▶ Busca garantir que os dados recebidos e enviados pelo módulo sejam corretos;
- ▶ E que os contratos estabelecidos pelo design do software sejam respeitados.

# Teste de Unidade

## Tipos de testes aplicados à interface



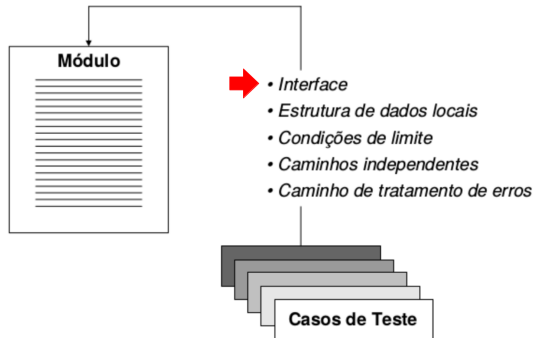
### Testes de Entrada e Saída

- ▶ Valida se os parâmetros passados para o módulo são corretamente interpretados.
- ▶ Garante que os valores retornados estejam dentro do esperado.



# Teste de Unidade

## Tipos de testes aplicados à interface

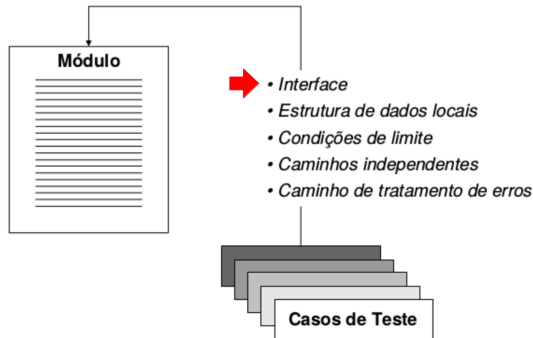


## Testes de Tipagem

- ▶ Confirma que os tipos de dados recebidos e enviados estão corretos.
- ▶ Exemplo: Se um módulo espera um inteiro, deve rejeitar strings.

# Teste de Unidade

## Tipos de testes aplicados à interface



## Testes de Mocking e Stubbing

- ▶ Se o módulo interage com outros sistemas, usar mocks (simulações) para verificar se as chamadas à interface ocorrem corretamente.
- ▶ Exemplo: Se o módulo faz uma requisição HTTP, simular a resposta esperada sem depender de um servidor real.

# Teste de Unidade

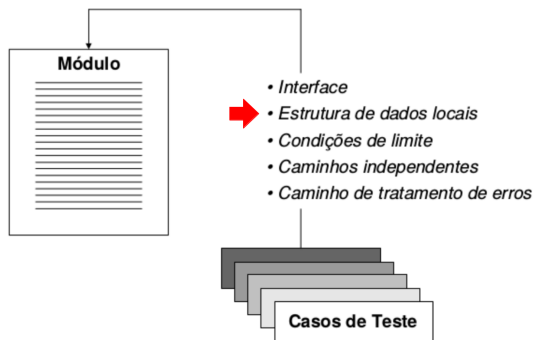
## Exemplo Prático de Teste de Interface

- Código de um módulo simples em Python

```
1 def dobrar_valor(numero: int) -> int:
2     if not isinstance(numero, int):
3         raise ValueError("Entrada deve ser um número inteiro")
4     return numero * 2
```

- Testes de Interface para este módulo

```
1 import unittest
2
3 class TestInterface(unittest.TestCase):
4     def test_entrada_valida(self):
5         self.assertEqual(dobrar_valor(5), 10)
6
7     def test_tipo_de_entrada(self):
8         with self.assertRaises(ValueError):
9             dobrar_valor("texto") # Testa se rejeita string
10
11
12 if __name__ == '__main__':
13     unittest.main()
```

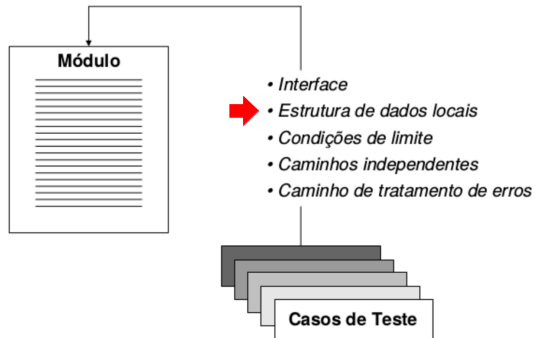


## Estruturas de Dados Locais

- ▶ São variáveis, listas, dicionários, objetos e outras estruturas utilizadas dentro do módulo.
- ▶ O teste visa garantir que os dados armazenados mantêm sua integridade ao longo da execução.
- ▶ Identifica possíveis inconsistências ou corrupção de dados durante o processamento.

# Teste de Unidade

Tipos de testes aplicados às estruturas de dados locais

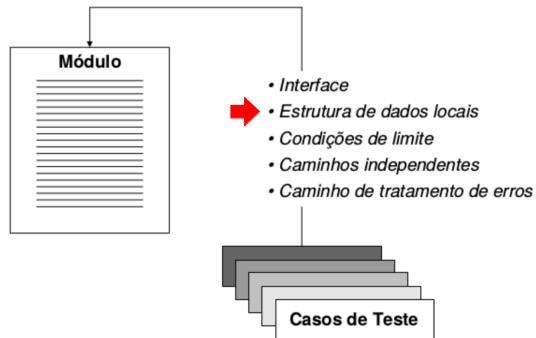


## Testes de Consistência

- ▶ Verificam se os dados permanecem consistentes após operações de leitura e escrita.
- ▶ Exemplo: Após uma operação de soma, o valor esperado deve estar correto na memória.

# Teste de Unidade

Tipos de testes aplicados às estruturas de dados locais

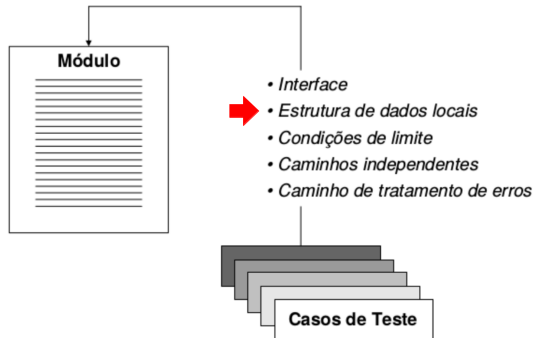


## Testes de Inicialização e Liberação de Memória

- ▶ Garante que as estruturas de dados são inicializadas corretamente.
- ▶ Avalia se os recursos alocados são liberados adequadamente.
- ▶ Exemplo: Uma lista vazia deve ser inicializada sem valores inesperados.

# Teste de Unidade

Tipos de testes aplicados às estruturas de dados locais

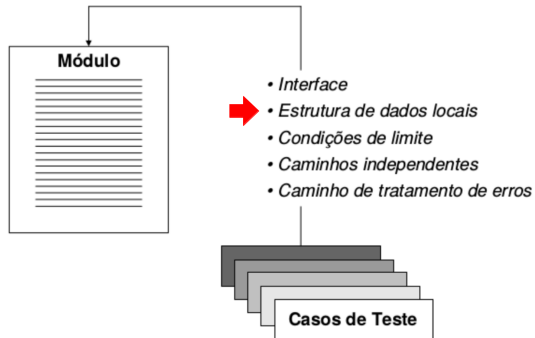


## Testes de Modificação de Dados

- ▶ Garante que as modificações feitas nas estruturas são coerentes e previsíveis.
- ▶ Exemplo: Se um item for adicionado a uma lista, ele deve estar acessível na posição correta.

# Teste de Unidade

Tipos de testes aplicados às estruturas de dados locais



## Testes de Concorrência e Acesso Simultâneo

- ▶ Avalia se múltiplas threads/processos acessam a estrutura sem causar inconsistências.
- ▶ Exemplo: Dois processos alterando um dicionário simultaneamente sem corrompê-lo.



# Teste de Unidade

## Exemplo Prático de Teste de Estruturas de Dados Locais

- Código de um módulo que gerencia uma lista em Python

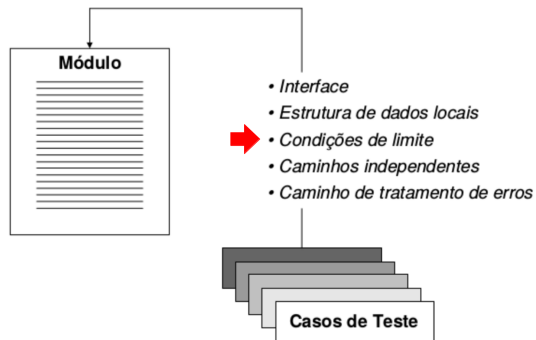
```
1 class GerenciadorLista:
2     def __init__(self):
3         self.lista = []
4
5     def adicionar(self, item):
6         self.lista.append(item)
7
8     def remover(self, item):
9         if item in self.lista:
10            self.lista.remove(item)
11        else:
12            raise ValueError("Item não encontrado")
13
14    def obter_lista(self):
15        return self.lista
```

# Teste de Unidade

## Exemplo Prático de Teste de Estruturas de Dados Locais

### ► Testes de Unidade para este módulo

```
1 import unittest
2
3 class TestGerenciadorLista(unittest.TestCase):
4     def setUp(self):
5         self.gerenciador = GerenciadorLista()
6
7     def test_adicionar(self):
8         self.gerenciador.adicionar(10)
9         self.assertIn(10, self.gerenciador.obter_lista())
10
11    def test_remove_existente(self):
12        self.gerenciador.adicionar(20)
13        self.gerenciador.remove(20)
14        self.assertNotIn(20, self.gerenciador.obter_lista())
15
16    def test_remove_inexistente(self):
17        with self.assertRaises(ValueError):
18            self.gerenciador.remove(30)
19
20 if __name__ == '__main__':
21     unittest.main()
```

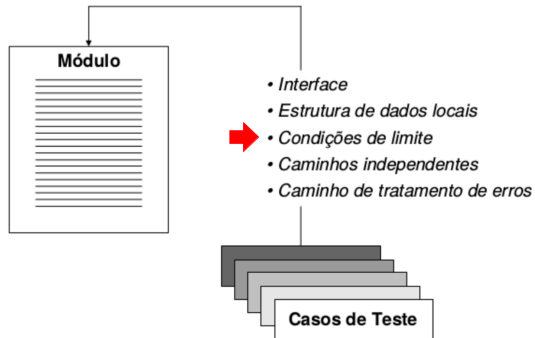


## Condições Limite

- ▶ Testam valores próximos dos limites máximo e mínimo permitidos pelo módulo.
- ▶ Identificam falhas quando o módulo recebe entradas no extremo dos intervalos esperados.
- ▶ Garantem que o módulo não falhe inesperadamente com valores extremos.

# Teste de Unidade

Tipos de testes aplicados às condições limite

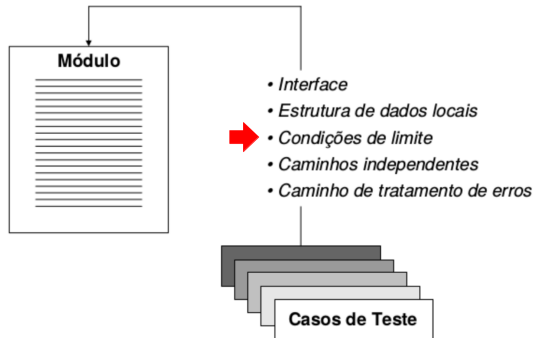


## Testes de Limite Inferior

- ▶ Avaliam o comportamento do módulo quando recebe o menor valor permitido.
- ▶ Exemplo: Se um sistema aceita idades de 18 a 65 anos, testar com 18 e menores valores.

# Teste de Unidade

Tipos de testes aplicados às condições limite

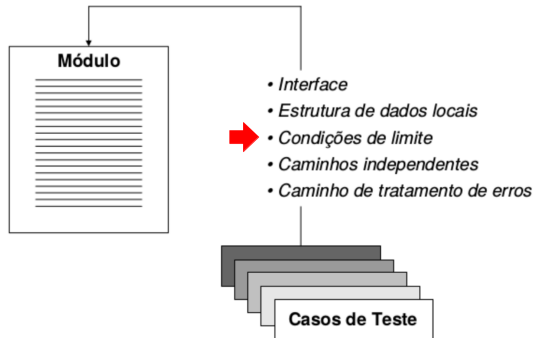


## Testes de Limite Superior

- ▶ Avaliam o comportamento do módulo quando recebe o maior valor permitido.
- ▶ Exemplo: Se um sistema aceita um salário de até R\$10.000, testar com 10.000 e valores superiores.

# Teste de Unidade

Tipos de testes aplicados às condições limite

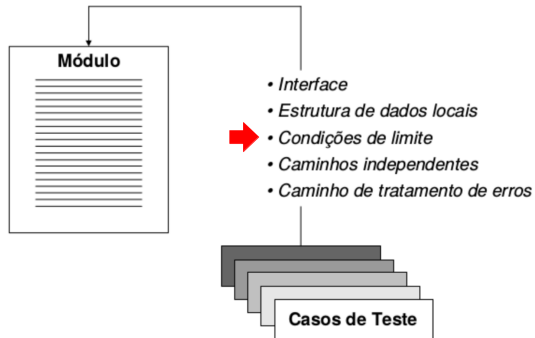


## Testes de Valores Fora do Intervalo

- ▶ Verificam como o módulo reage a valores menores e maiores que os permitidos.
- ▶ Exemplo: Se um módulo aceita números entre 1 e 100, testar 0 e 101.

# Teste de Unidade

Tipos de testes aplicados às condições limite



## Testes de Precisão em Valores Limítrofes

- ▶ Avaliam como o módulo lida com valores muito próximos aos limites.
- ▶ Exemplo: Se um sistema aceita até R\$10.000, testar com 9.999,99.

# Teste de Unidade

## Exemplo Prático de Teste de Condições Limite

► Código de um módulo que valida uma entrada numérica em Python

```
1 def validar_numero(n):  
2     if n < 1 or n > 100:  
3         raise ValueError("Número fora do intervalo permitido")  
4     return True
```

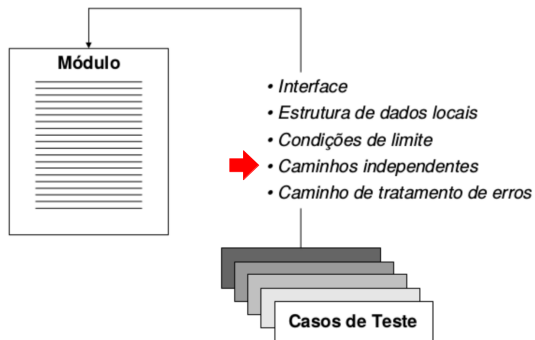


# Teste de Unidade

## Exemplo Prático de Teste de Condições Limite

### ► Testes de Unidade para este módulo

```
1 import unittest
2
3 class TestValidarNumero(unittest.TestCase):
4     def test_limite_inferior(self):
5         self.assertTrue(validar_numero(1)) # Valor mínimo válido
6
7     def test_limite_superior(self):
8         self.assertTrue(validar_numero(100)) # Valor máximo válido
9
10    def test_abaixo_do_limite(self):
11        with self.assertRaises(ValueError):
12            validar_numero(0) # Fora do intervalo
13
14    def test_acima_do_limite(self):
15        with self.assertRaises(ValueError):
16            validar_numero(101) # Fora do intervalo
17
18    def test_proximidade_limite(self):
19        self.assertTrue(validar_numero(99)) # Próximo ao limite superior
20
21 if __name__ == '__main__':
22     unittest.main()
```

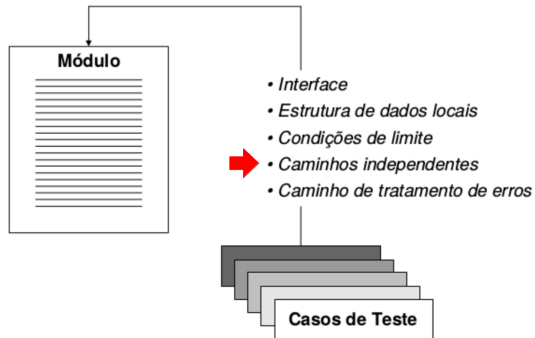


## Caminhos Independentes

- ▶ Testam os caminhos básicos da estrutura de controle do módulo.
- ▶ Garantem que todas as instruções sejam executadas pelo menos uma vez.
- ▶ Identificam fluxos não testados e potenciais falhas lógicas no código.

# Teste de Unidade

Tipos de testes aplicados aos caminhos independentes

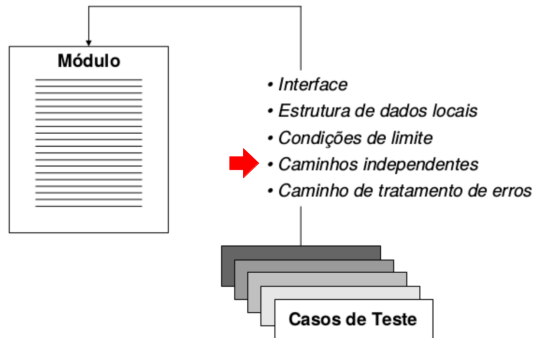


## Testes de Fluxos de Controle

- ▶ Avaliam cada caminho possível dentro do módulo.
- ▶ Exemplo: Para uma estrutura condicional 'if-else', testar os dois caminhos.

# Teste de Unidade

Tipos de testes aplicados aos caminhos independentes

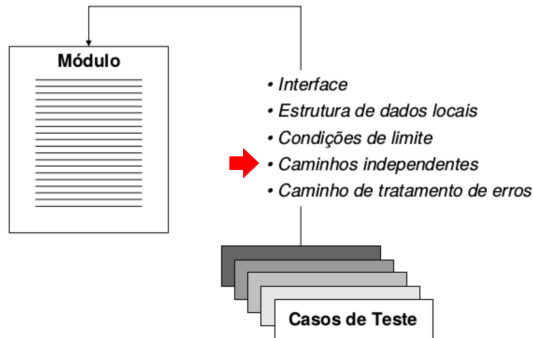


## Testes de Cobertura de Decisões

- ▶ Garantem que todas as decisões do código (if, loops) sejam avaliadas em todas as condições.
- ▶ Exemplo: Testar `'if (x > 0)'` tanto para x positivo quanto negativo.

# Teste de Unidade

Tipos de testes aplicados aos caminhos independentes

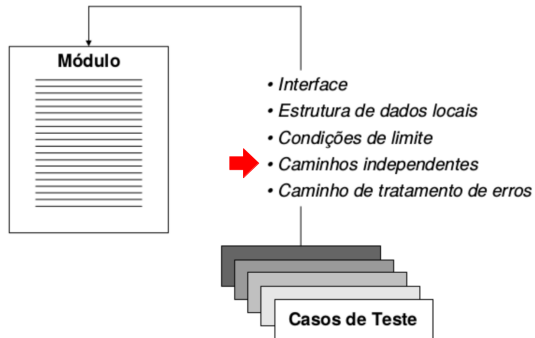


## Testes em Estruturas de Repetição

- ▶ Validam loops e iterações garantindo execução mínima e máxima.
- ▶ Exemplo: Para um `'for i in range(5)'`, testar `'i=0'` e `'i=4'` (última iteração).

# Teste de Unidade

Tipos de testes aplicados aos caminhos independentes



## Testes de Cobertura Total

- ▶ Garante que todas as linhas de código sejam testadas.
- ▶ Exemplo: Usar ferramentas como `'coverage.py'` para medir cobertura de código.

# Teste de Unidade

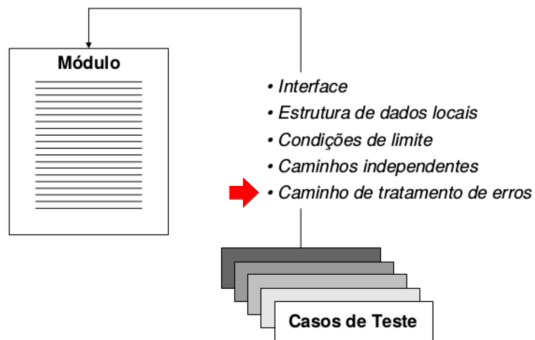
## Exemplo Prático de Teste de Caminhos Independentes

- Código de um módulo com múltiplos caminhos em Python

```
1 def avaliar_numero(n):  
2     if n > 10:  
3         return "Maior que 10"  
4     elif n == 10:  
5         return "Igual a 10"  
6     else:  
7         return "Menor que 10"
```

- Testes de Unidade para este módulo

```
1 import unittest  
2  
3 class TestAvaliarNumero(unittest.TestCase):  
4     def test_maior_que_10(self):  
5         self.assertEqual(avaliar_numero(15), "Maior que 10")  
6  
7     def test_igual_a_10(self):  
8         self.assertEqual(avaliar_numero(10), "Igual a 10")  
9  
10    def test_menor_que_10(self):  
11        self.assertEqual(avaliar_numero(5), "Menor que 10")  
12  
13    if __name__ == '__main__':  
14        unittest.main()
```



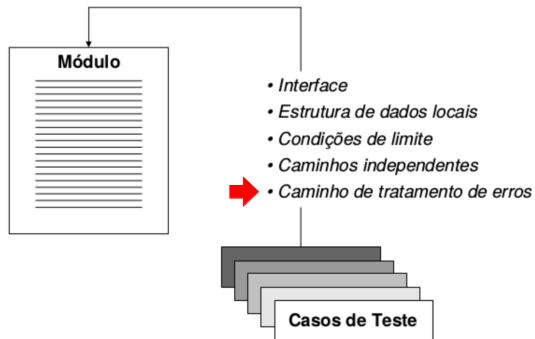
## Caminhos de Tratamento de Erros

- ▶ São os caminhos executados quando erros ocorrem dentro do módulo.
- ▶ O teste avalia a robustez do código diante de falhas inesperadas.
- ▶ Garante que o módulo responde adequadamente a entradas inválidas ou falhas no sistema.



# Teste de Unidade

Tipos de testes aplicados ao tratamento de erros

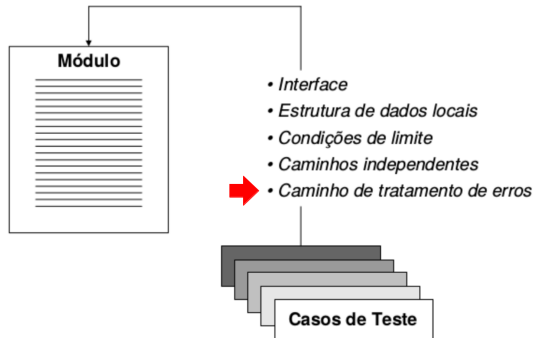


## Testes de Entradas Inválidas

- ▶ Avaliam como o módulo lida com entradas inesperadas ou malformadas.
- ▶ Exemplo: Se um módulo espera um número, testar com string ou 'None'.

# Teste de Unidade

Tipos de testes aplicados ao tratamento de erros

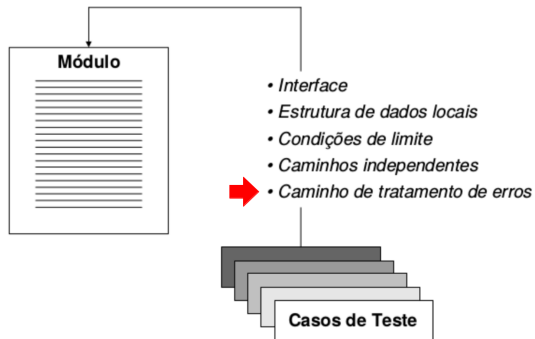


## Testes de Exceções e Falhas Internas

- ▶ Avaliam se as exceções são tratadas corretamente sem falhas inesperadas.
- ▶ Exemplo: Se uma operação pode falhar (divisão por zero), o módulo deve capturar e tratar o erro.

# Teste de Unidade

## Tipos de testes aplicados ao tratamento de erros

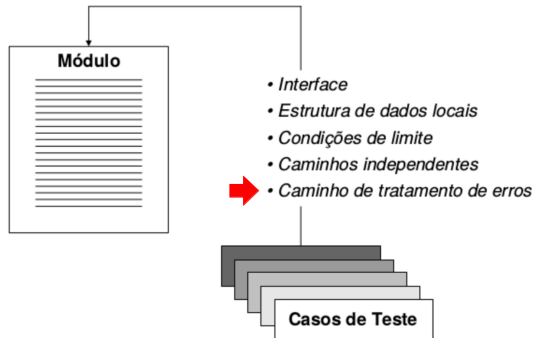


### Testes de Erros em Dependências Externas

- ▶ Simulam falhas em bancos de dados, APIs ou arquivos externos.
- ▶ Exemplo: Se uma API não responde, o módulo deve lidar com isso sem falhar.

# Teste de Unidade

Tipos de testes aplicados ao tratamento de erros



## Testes de Mensagens de Erro

- ▶ Avaliam se as mensagens de erro são claras e informativas.
- ▶ Exemplo: Em caso de erro, a mensagem deve orientar o usuário sobre o problema.

# Teste de Unidade

## Exemplo Prático de Teste de Tratamento de Erros

- Código de um módulo que lida com erros em Python

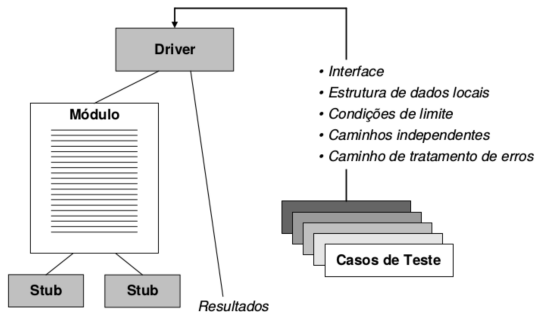
```
1 def dividir(a, b):  
2     try:  
3         return a / b  
4     except ZeroDivisionError:  
5         return "Erro: Divisão por zero não permitida"
```

- Testes de Unidade para este módulo

```
1 import unittest  
2  
3 class TestDivisao(unittest.TestCase):  
4     def test_divisao_valida(self):  
5         self.assertEqual(dividir(10, 2), 5.0)  
6  
7     def test_divisao_por_zero(self):  
8         self.assertEqual(dividir(10, 0), "Erro: Divisão por zero não permitida")  
9  
10    def test_divisao_com_strings(self):  
11        with self.assertRaises(TypeError):  
12            dividir("10", 2)  
13  
14    if __name__ == '__main__':  
15        unittest.main()
```

# Teste de Unidade

## Ambiente de Teste de Unidade



## Drivers e Stubs

- ▶ Drivers permitem testar um módulo que não está completamente integrado ao sistema.
- ▶ Stubs simulam módulos auxiliares, permitindo a testagem isolada da funcionalidade principal.
- ▶ O teste de unidade visa verificar todos os aspectos críticos do módulo antes da integração completa no sistema.

## Definição

- ▶ Tem por objetivo a busca de erros surgidos quando da integração de diferentes módulos;
- ▶ A análise dos módulos individualmente não garante o funcionamento integrado;
- ▶ Uma das maiores causas de erros encontrados durante o teste de integração são os chamados erros de interface;
- ▶ Devido, principalmente, às incompatibilidades de interface entre módulos que deverão trabalhar de forma cooperativa.

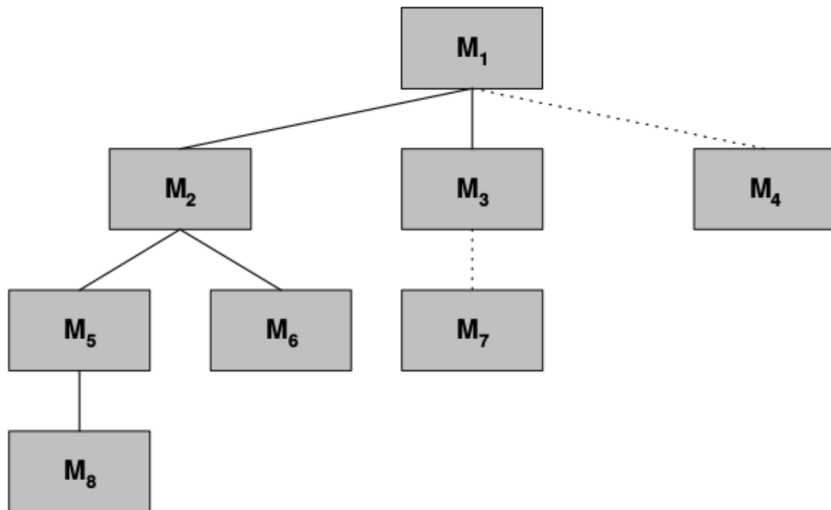
## Abordagens

- ▶ **Big bang:** processo de integração não-incremental na qual todos os módulos são associados e o programa é testado como um todo;
- ▶ **Incremental:** tem-se mostrado mais eficiente pois o programa vai sendo construído aos poucos e testado por partes.



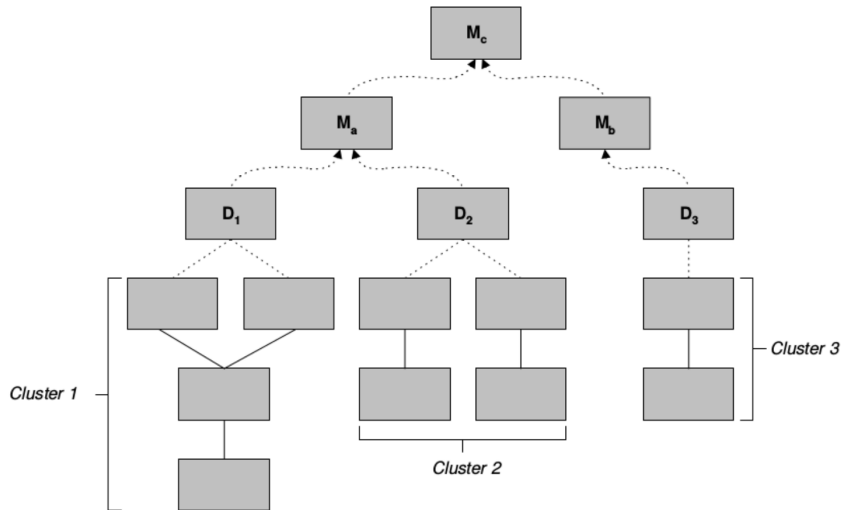
# Teste de Integração

## Integração Top-Down



# Teste de Integração

## Integração Bottom-Up



# Teste de Integração

Exemplo: Sistema de Pagamentos

## Contexto

- ▶ Um sistema tem dois módulos: Autenticação e Pagamento.
- ▶ O módulo de pagamento depende da autenticação para validar usuários.
- ▶ Um erro de integração pode ocorrer devido à diferença no formato de retorno.

# Teste de Integração

Código original - Com erro de integração

```
1  # Módulo de Autenticação
2  def verificar_usuario(id_usuario: int) -> bool:
3      usuarios_validos = {101, 102, 103}
4      return id_usuario in usuarios_validos
5
6  # Módulo de Pagamento (Erro de integração)
7  def processar_pagamento(id_usuario: int, valor: float) -> str:
8      if verificar_usuario(id_usuario): # Espera string, mas recebe booleano
9          return "Pagamento aprovado"
10     return "Pagamento negado"
```

## Problema

- ▶ O módulo de pagamento espera "Aprovado"/"Negado", mas recebe True/False.
- ▶ Isso pode causar falhas na lógica de decisão.

# Teste de Integração

Código corrigido - Interface padronizada

```
1  # Correção no módulo de autenticação
2  def verificar_usuario(id_usuario: int) -> str:
3      usuarios_validos = {101, 102, 103}
4      return "Aprovado" if id_usuario in usuarios_validos else "Negado"
5
6  # Módulo de Pagamento corrigido
7  def processar_pagamento(id_usuario: int, valor: float) -> str:
8      status = verificar_usuario(id_usuario)  # Agora recebe string compatível
9      if status == "Aprovado":
10         return f"Pagamento de R${valor:.2f} aprovado"
11     return "Pagamento negado"
```

## Solução

- ▶ O retorno do módulo de autenticação foi alterado para ser compatível.
- ▶ Agora, os módulos podem ser integrados sem falhas na comunicação.

# Teste de Integração

Testando a integração entre os módulos

```
1 import unittest
2
3 class TesteIntegracao(unittest.TestCase):
4     def test_pagamento_aprovado(self):
5         self.assertEqual(processar_pagamento(101, 50.0), "Pagamento de R$50.00 aprovado")
6
7     def test_pagamento_negado(self):
8         self.assertEqual(processar_pagamento(200, 50.0), "Pagamento negado") # Usuário
9                                     inexistente
10
11 if __name__ == '__main__':
12     unittest.main()
```

## Resultado esperado

- ▶ Teste deve passar sem erros, garantindo a correta integração entre módulos.
- ▶ Se um teste falhar, indica erro na comunicação entre os módulos.

Simulação e Teste de Software (CC8550)  
Aula 05 - Níveis de testes:  
Aprofundamento em Testes de Unidade e Integração

Prof. Luciano Rossi

Ciência da Computação  
Centro Universitário FEI

1º Semestre de 2025