Simulação e Teste de Software (CC8550)

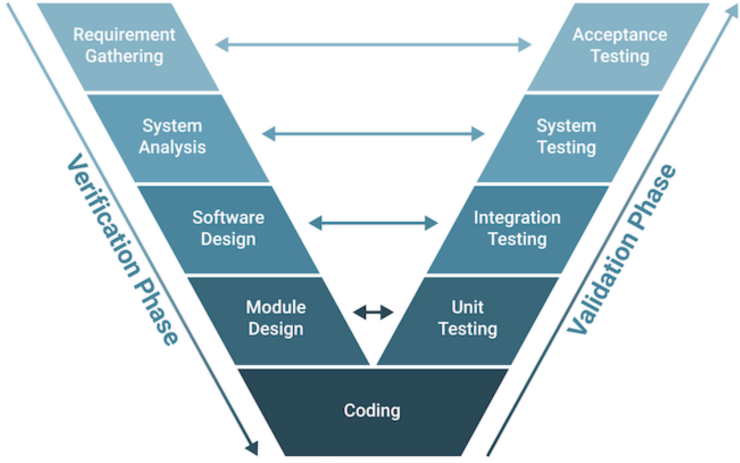
Aula 05 – Níveis de teste: unidade, integração, sistema e aceitação.

Prof. Luciano Rossi

Ciência da Computação
Centro Universitário FEI

2° Semestre de 2025

Modelo V que descreve os níveis de teste



Teste de Unidade

Teste de Unidade

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.

centro universitário FEI

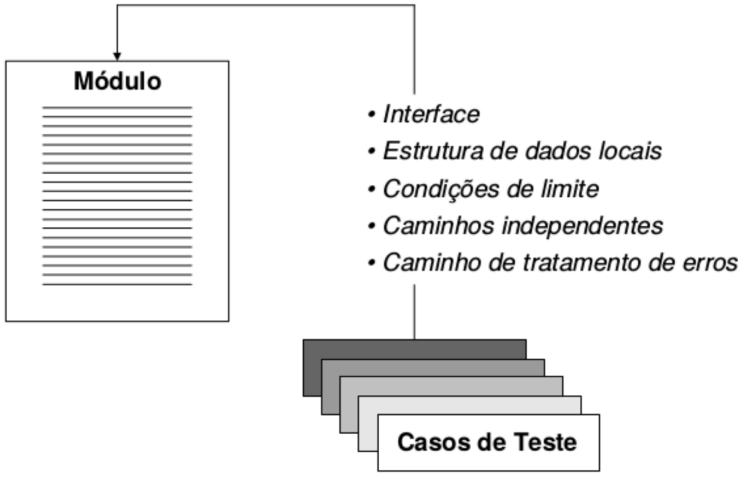
O que são as unidades de projeto?

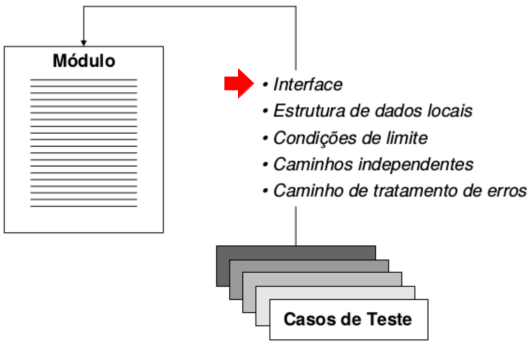
- **Função**: Bloco de código que executa uma tarefa específica e pode retornar um valor.
- **Classe**: Estrutura que encapsula atributos e métodos relacionados, representando objetos no paradigma orientado a objetos.
- **Módulo**: Arquivo ou conjunto de funções e classes agrupadas para modularizar o 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



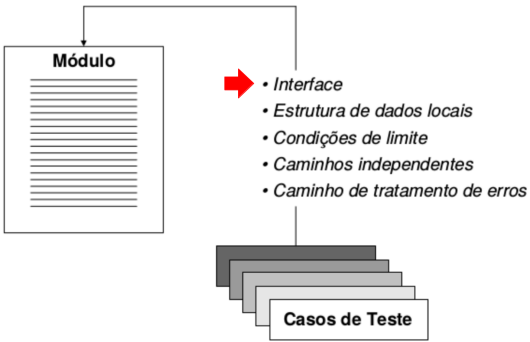


Interface

- 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.

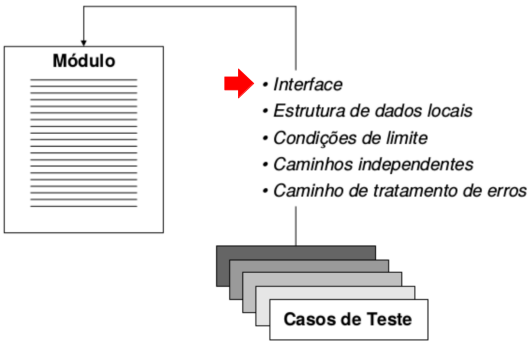
Teste de Unidade

Tipos de testes aplicados à interface



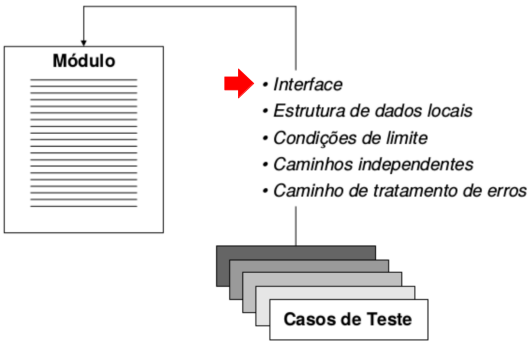
Testes de Entrada e Saída

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



Testes de Tipagem

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



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

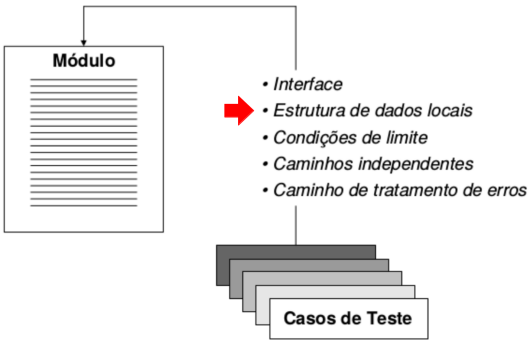
Exemplo Prático de Teste de Interface

- Código de um módulo simples em Python

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

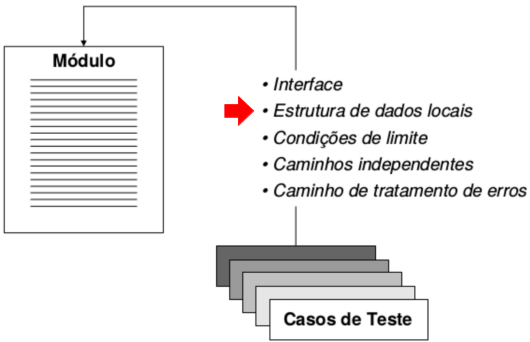
- Testes de Interface para este módulo

```python
import unittest

class TestInterface(unittest.TestCase):
    def test_entrada_valida(self):
        self.assertEqual(dobrar_valor(5), 10)

    def test_tipo_de_entrada(self):
        with self.assertRaises(ValueError):
        dobrar_valor("texto")  # Testa se rejeita string


if __name__ == '__main__':
    unittest.main()
```



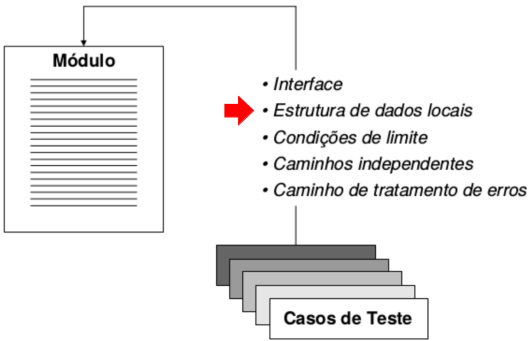
Estruturas de Dados Locais

- 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.



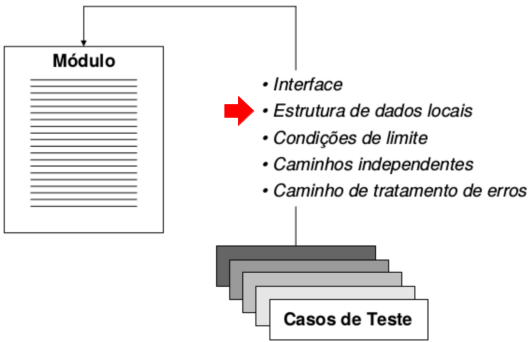
Testes de Consistência

- 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.



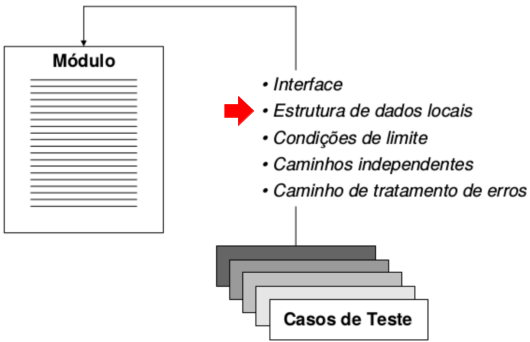
Testes de Inicialização e Liberação de Memória

- 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.



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.



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.

- Código de um módulo que gerencia uma lista em Python

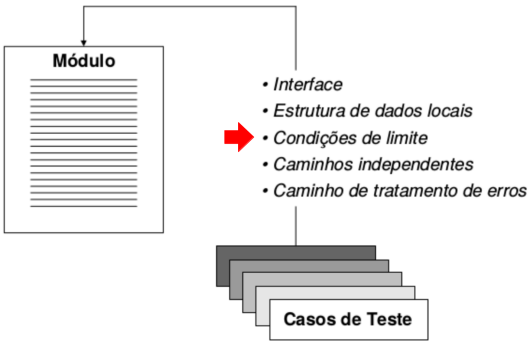
```python
class GerenciadorLista:
    def __init__(self):
        self.lista = []

    def adicionar(self, item):
        self.lista.append(item)

    def remover(self, item):
        if item in self.lista:
            self.lista.remove(item)
        else:
            raise ValueError("Item não encontrado")

    def obter_lista(self):
        return self.lista
```

Teste de Unidade

Exemplo Prático de Teste de Estruturas de Dados Locais

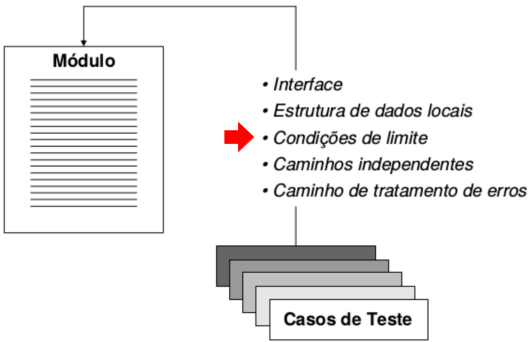
- Testes de Unidade para este módulo

```python
import unittest

class TestGerenciadorLista(unittest.TestCase):
    def setUp(self):
        self.gerenciador = GerenciadorLista()

    def test_adicionar(self):
        self.gerenciador.adicionar(10)
        self.assertIn(10, self.gerenciador.obter_lista())

    def test_remover_existente(self):
        self.gerenciador.adicionar(20)
        self.gerenciador.remover(20)
        self.assertNotIn(20, self.gerenciador.obter_lista())

    def test_remover_inexistente(self):
        with self.assertRaises(ValueError):
            self.gerenciador.remover(30)

if __name__ == '__main__':
    unittest.main()
```



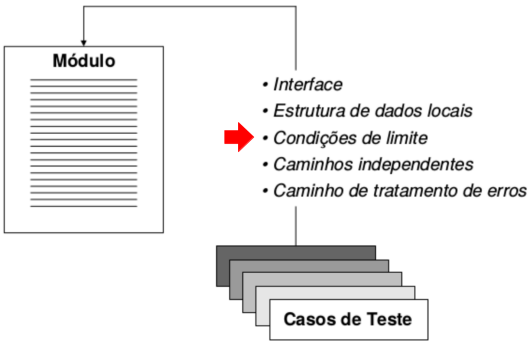
Condições Limite

- 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.



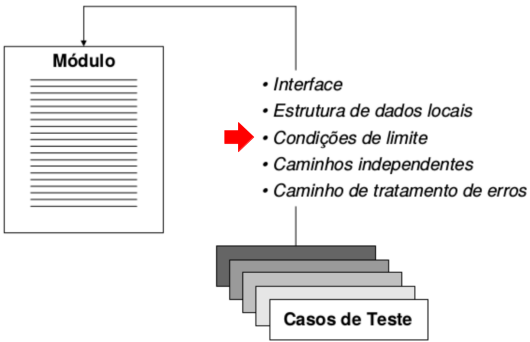
Testes de Limite Inferior

- 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.



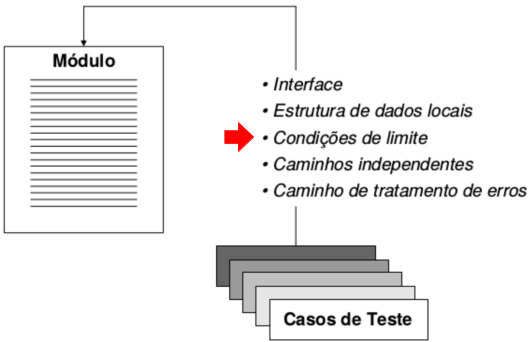
Testes de Limite Superior

- 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.



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.



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.

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

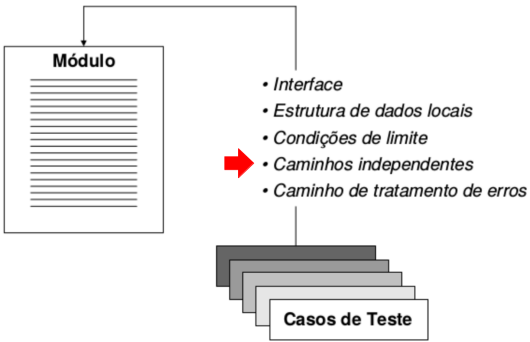
```python
def validar_numero(n):
    if n < 1 or n > 100:
        raise ValueError("Número fora do intervalo permitido")
    return True
```

Teste de Unidade

Exemplo Prático de Teste de Condições Limite

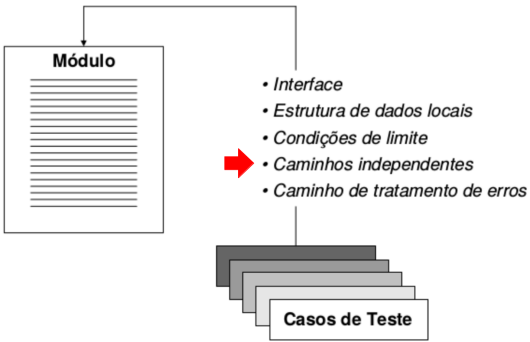
- Testes de Unidade para este módulo

```python
import unittest

    class TestValidarNumero(unittest.TestCase):
        def test_limite_inferior(self):
            self.assertTrue(validar_numero(1))  # Valor mínimo válido

        def test_limite_superior(self):
            self.assertTrue(validar_numero(100))  # Valor máximo válido

        def test_abaixo_do_limite(self):
            with self.assertRaises(ValueError):
                validar_numero(0)  # Fora do intervalo

        def test_acima_do_limite(self):
            with self.assertRaises(ValueError):
                validar_numero(101)  # Fora do intervalo

        def test_proximidade_limite(self):
            self.assertTrue(validar_numero(99))  # Próximo ao limite superior

if __name__ == '__main__':
    unittest.main()
```



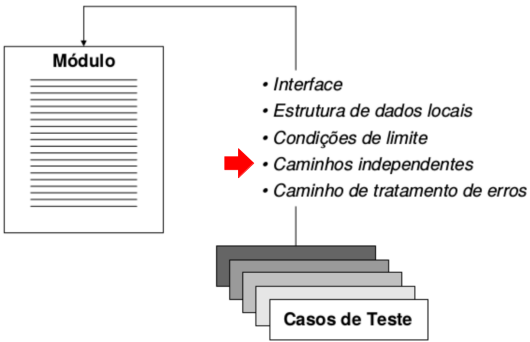
Caminhos Independentes

- 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.



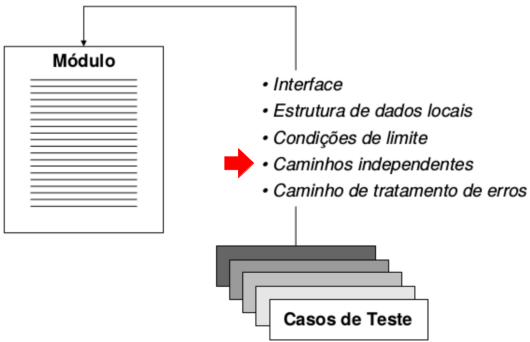
Testes de Fluxos de Controle

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



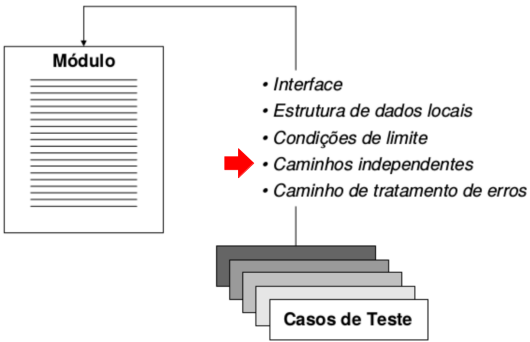
Testes de Cobertura de Decisões

- 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.



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).



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

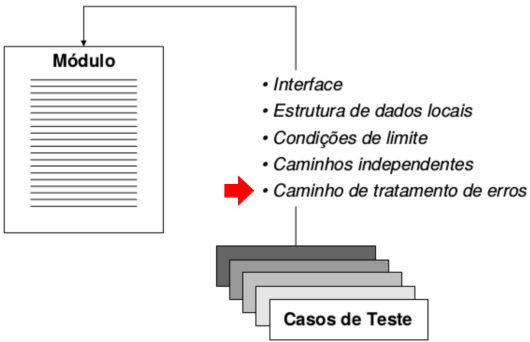
Exemplo Prático de Teste de Caminhos Independentes

- Código de um módulo com múltiplos caminhos em Python

```python
def avaliar_numero(n):
    if n > 10:
        return "Maior que 10"
    elif n == 10:
        return "Igual a 10"
    else:
        return "Menor que 10"
```

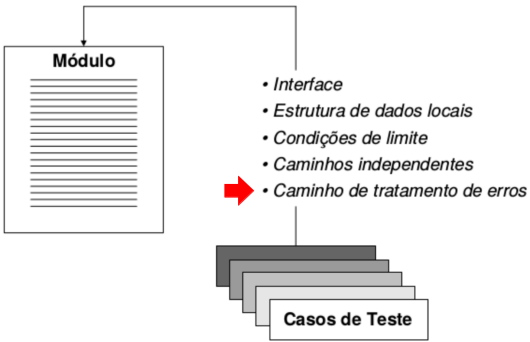
- Testes de Unidade para este módulo

```python
import unittest

class TestAvaliarNumero(unittest.TestCase):
    def test_maior_que_10(self):
        self.assertEqual(avaliar_numero(15), "Maior que 10")

    def test_igual_a_10(self):
        self.assertEqual(avaliar_numero(10), "Igual a 10")

    def test_menor_que_10(self):
        self.assertEqual(avaliar_numero(5), "Menor que 10")

if __name__ == '__main__':
    unittest.main()
```



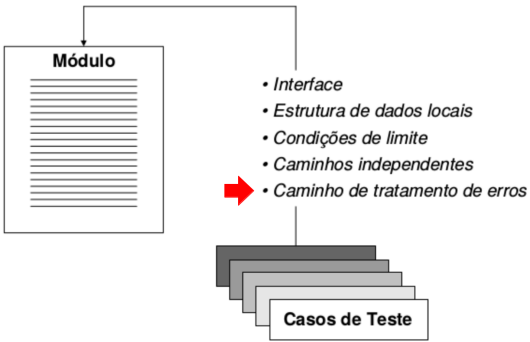
Caminhos de Tratamento de Erros

- 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.



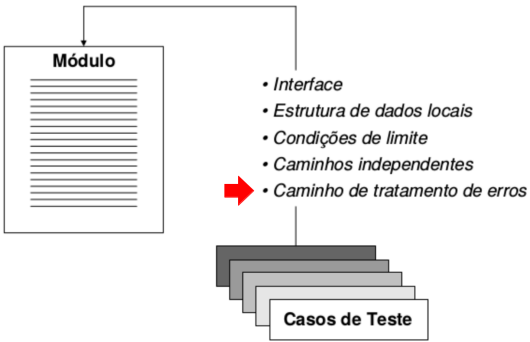
Testes de Entradas Inválidas

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



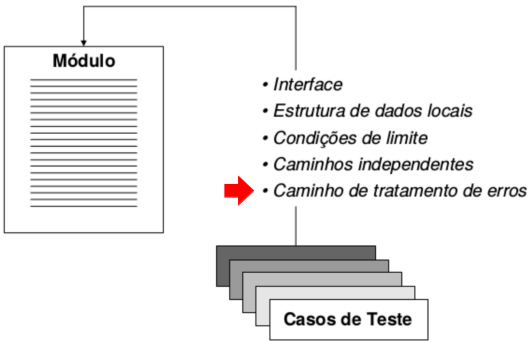
Testes de Exceções e Falhas Internas

- 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.



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.



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

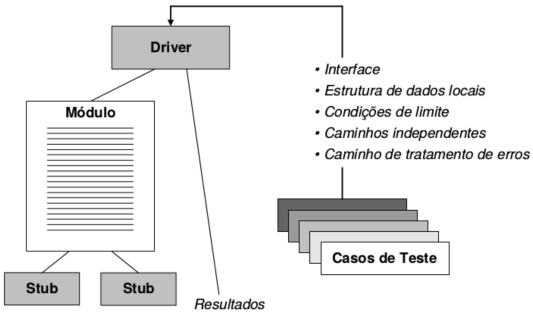
Exemplo Prático de Teste de Tratamento de Erros

- Código de um módulo que lida com erros em Python

```python
def dividir(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Erro: Divisão por zero não permitida"
```

- Testes de Unidade para este módulo

```python
import unittest

class TestDivisao(unittest.TestCase):
    def test_divisao_valida(self):
        self.assertEqual(dividir(10, 2), 5.0)

    def test_divisao_por_zero(self):
        self.assertEqual(dividir(10, 0), "Erro: Divisão por zero não permitida")

    def test_divisao_com_strings(self):
        with self.assertRaises(TypeError):
            dividir("10", 2)

if __name__ == '__main__':
    unittest.main()
```



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.

Teste de Integração

Definição

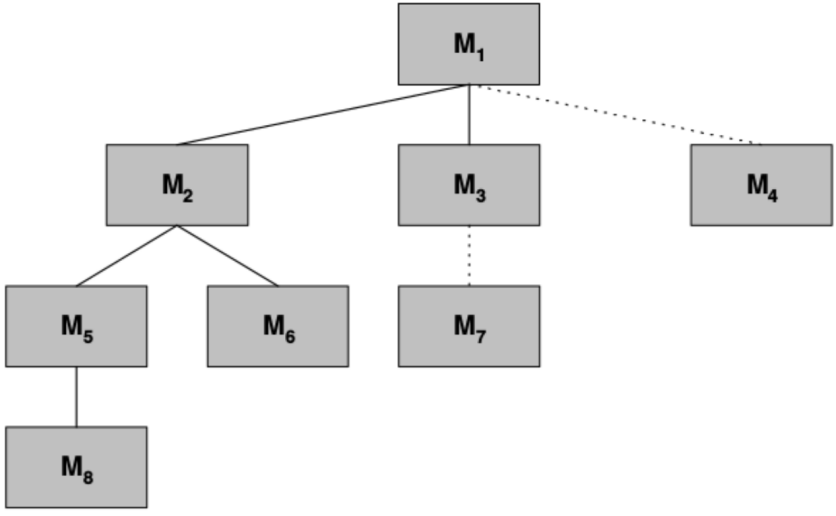
- Tem por objetivo a busca de erros surgidos quando da integração de diferentes módulos;
- A análise dos módulos individualmente não garante o funcionamento integrado;
- Uma das maiores causas de erros encontrados durante o teste de integração são os chamados erros de interface;
- Devido, principalmente,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

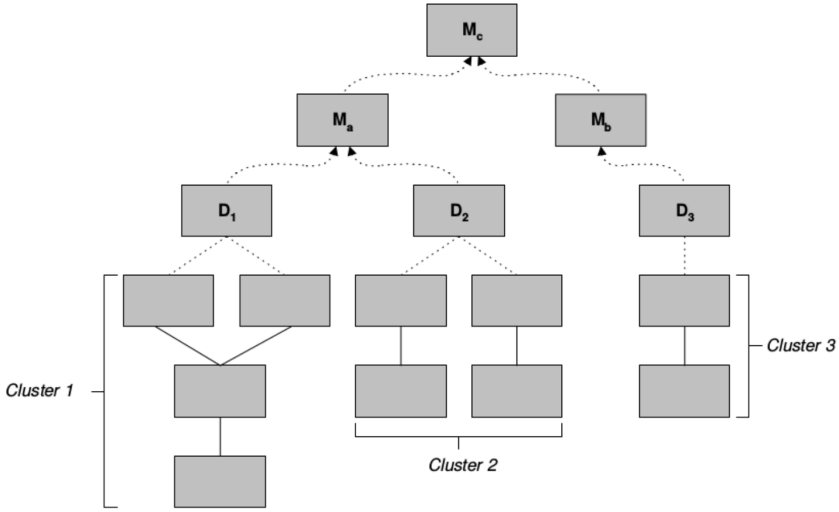
Teste de Integração

Integração Top-Down



Teste de Integração

Integração Bottom-Up



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

```python
# Módulo de Autenticação
def verificar_usuario(id_usuario: int) -> bool:
    usuarios_validos = {101, 102, 103}
    return id_usuario in usuarios_validos

# Módulo de Pagamento (Erro de integração)
def processar_pagamento(id_usuario: int, valor: float) -> str:
    if verificar_usuario(id_usuario):  # Espera string, mas recebe booleano
        return "Pagamento aprovado"
    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

```python
# Correção no módulo de autenticação
def verificar_usuario(id_usuario: int) -> str:
    usuarios_validos = {101, 102, 103}
    return "Aprovado" if id_usuario in usuarios_validos else "Negado"

# Módulo de Pagamento corrigido
def processar_pagamento(id_usuario: int, valor: float) -> str:
    status = verificar_usuario(id_usuario)  # Agora recebe string compatível
    if status == "Aprovado":
        return f"Pagamento de R${valor:.2f} aprovado"
    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.

centro universitário FEI

Teste de Integração

Testando a integração entre os módulos

```python
import unittest

class TesteIntegracao(unittest.TestCase):
    def test_pagamento_aprovado(self):
        self.assertEqual(processar_pagamento(101, 50.0), "Pagamento de R$50.00 aprovado")

    def test_pagamento_negado(self):
        self.assertEqual(processar_pagamento(200, 50.0), "Pagamento negado")  # Usuário
            inexistente

if __name__ == '__main__':
    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.

Teste de Sistema

Objetivo

Validar o comportamento do sistema completo e integrado em um ambiente que simula a produção, verificando o atendimento aos requisitos funcionais e não-funcionais.

- Testa o sistema como uma caixa preta
- Verifica fluxos end-to-end completos
- Avalia requisitos não-funcionais (performance, segurança, usabilidade)
- Executa em ambiente similar à produção
- Utiliza dados representativos do mundo real

Teste de Sistema

Exemplo Prático - E-commerce

```python
def test_fluxo_compra_completo():
    """Teste end-to-end do processo de compra"""

    # 1. Autenticacao
    usuario = fazer_login("cliente@test.com", "senha123")
    assert usuario.autenticado == True

    # 2. Busca de produto
    produto = buscar_produto("Notebook Dell")
    assert produto.disponivel == True

    # 3. Carrinho de compras
    carrinho = adicionar_ao_carrinho(produto, quantidade=1)
    assert len(carrinho.itens) == 1

    # 4. Checkout e pagamento
    pedido = finalizar_compra(
    carrinho=carrinho,
    endereco="Rua ABC, 123",
    cartao="4111111111111111"
    )

    # 5. Validacoes finais
    assert pedido.status == "CONFIRMADO"
    assert verificar_email_confirmacao(usuario.email)
    assert verificar_reducao_estoque(produto.id)
```

Teste de Aceitação

Objetivo

Validar se o sistema atende aos critérios de aceitação definidos pelo cliente e está pronto para ser colocado em produção.

- **Perspectiva do usuário:** Testa do ponto de vista do cliente
- **Critérios de negócio:** Valida regras e processos
- **Aprovação final:** Último gate antes da produção
- **Participação do cliente:** Usuários finais executam testes
- **Cenários reais:** Usa casos de uso do mundo real

Simulação e Teste de Software (CC8550)

Aula 05 – Níveis de teste: unidade, integração, sistema e aceitação.

Prof. Luciano Rossi

Ciência da Computação
Centro Universitário FEI

2º Semestre de 2025