Automação de Testes e Análise de Cobertura

Boas Práticas de Programação - 2025.2

Prof. [Fernando Figueira]

Universidade Federal do Rio Grande do Norte

07 de Novembro de 2025

Agenda

Por que Automatizar Testes?

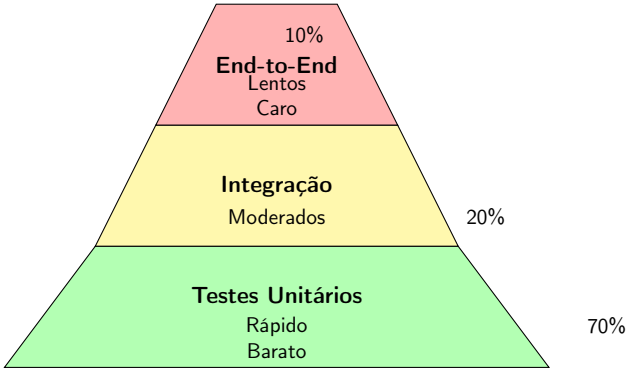
Benefícios:

- Detecção precoce de bugs
- Redução de custos
- Documentação viva
- Confiança para refatorar
- Integração com CI/CD

Estatística

Bugs encontrados na produção custam **15x mais** para corrigir do que bugs encontrados durante o desenvolvimento.

A Pirâmide de Testes



Tipos de Testes

Testes Unitários (70%)

Testam **funções e métodos isolados**. Rápidos e focados.

Testes de Integração (20%)

Testam **interação entre componentes**. Banco de dados, APIs, módulos.

Testes End-to-End (10%)

Testam **fluxo completo do usuário**. Interface gráfica, simulação real.

Características de Bons Testes: FIRST

Fast	Executam rapidamente (segundos, não minutos)
Independent	Não dependem de outros testes
Repeatable	Resultados consistentes em qualquer ambiente
Self-validating	Pass/Fail claro, sem verificação manual
Timely	Escritos junto com o código (idealmente antes!)

Atenção

Um teste que não segue FIRST é um teste frágil!

O que é Cobertura de Código?

Definição

Percentual do código que é **executado** pelos testes automatizados.

O que medir:

- Linhas executadas
- Branches (if/else)
- Funções chamadas
- Condições booleanas

Importante

100% de cobertura
$\neq$
100% de qualidade

Tipos de Cobertura

1. **Cobertura de Linhas**
   - Quantas linhas foram executadas?
   - Meta: 70-80% para código geral
2. **Cobertura de Branches**
   - Quantos caminhos (if/else) foram testados?
   - Meta: 60-70% para código geral
3. **Cobertura de Funções**
   - Quantas funções foram chamadas?
   - Meta: 80-90% para módulos críticos

Exemplo: Código com Baixa Cobertura

```python
def calcular_desconto(preco, cupom):
    if cupom == "DESC10":
        return preco * 0.9
    elif cupom == "DESC20":
        return preco * 0.8
    elif cupom == "FRETE_GRATIS":
        return preco
    else:
        raise ValueError("Cupom invalido")
```

```python
def test_desconto():
    assert calcular_desconto(100, "DESC10") == 90
```

Problema

Cobertura de linhas: 37.5% | Branches: 25%

Exemplo: Código com Boa Cobertura

```python
def test_desconto_10():
    assert calcular_desconto(100, "DESC10") == 90

def test_desconto_20():
    assert calcular_desconto(100, "DESC20") == 80

def test_frete_gratis():
    assert calcular_desconto(100, "FRETE_GRATIS") == 100

def test_cupom_invalido():
    with pytest.raises(ValueError):
        calcular_desconto(100, "INVALIDO")
```

Resultado

Cobertura de linhas: 100% | Branches: 100%

Métricas de Qualidade

Categoria	Mínimo	Ideal
Cobertura de Linhas	70%	80-90%
Cobertura de Branches	60%	70-80%
Módulos Críticos	85%	95%+
Código Legado	50%	70%

Boas Práticas

- ✔ Priorize código crítico e complexo
- ✔ Use cobertura como guia, não como meta absoluta
- ✖ Não infle cobertura com testes vazios

Ferramentas por Linguagem

Linguagem	Framework	Cobertura
Python	pytest, unittest	coverage.py
JavaScript	Jest, Mocha	Istanbul, nyc
Java	JUnit, TestNG	JaCoCo
C/C++	Google Test, Catch2	gcov, lcov
C#	NUnit, xUnit	OpenCover
Go	testing (built-in)	go test -cover

Padrão AAA (Arrange-Act-Assert)

Estrutura Clara de Testes

Todo teste deve seguir três fases bem definidas:

```python
def test_calcular_desconto():
    # ARRANGE - Preparar dados e dependencias
    produto = Produto(nome="Notebook", preco=3000)
    desconto = 0.10

    # ACT - Executar a acao
    preco_final = produto.aplicar_desconto(desconto)

    # ASSERT - Verificar resultado
    assert preco_final == 2700
    assert produto.preco == 3000  # nao modifica original
```

O que Testar?

✔ **Priorize testar:**

- Lógica de negócio
- Cálculos e transformações
- Validações
- Casos limite (edge cases)
- Tratamento de erros
- Funções complexas

✖ **Não priorize:**

- Getters/setters simples
- Código de terceiros
- Constantes
- Configurações triviais
- Framework/biblioteca

Vamos à Prática! 

- Exercício 1: Testes Unitários
- Exercício 2: Cobertura de Código
- Exercício 3: TDD na Prática

Ferramenta: Replit (replit.com) ou Google Colab

Exercício 1: Sistema de Senhas

Contexto: Validação de senhas seguras

Sua tarefa:

- Implementar testes para validação de senha
- Testar casos válidos e inválidos
- Testar cálculo de força da senha

Requisitos:

- Senha deve ter no mínimo 8 caracteres
- Deve conter maiúscula, minúscula e número
- Força: Fraca, Média ou Forte

Exercício 2: Carrinho de Compras

Contexto: Sistema com descontos e cupons

Sua tarefa:

- Escrever testes que cubram todos os cenários
- Analisar cobertura com coverage.py
- Identificar código não coberto
- Criar testes para atingir 100% de cobertura

Desafio

Após rodar `coverage report`, identifique as linhas não cobertas e crie testes para elas!

Exercício 3: TDD - Calculadora de IMC

Contexto: Implementar seguindo Test-Driven Development

Ciclo TDD:

1. RED - Escrever teste que falha
2. GREEN - Implementar código mínimo para passar
3. REFACTOR - Melhorar o código

Funcionalidades:

- Calcular IMC: peso / altura^2
- Classificar: Abaixo do peso, Normal, Sobrepeso, Obesidade
- Validar entradas (peso e altura > 0)

Requisitos para U3

Automação de Testes (30%)

- Mínimo 15 testes unitários
- Mínimo 5 testes de integração
- Seguir padrão AAA
- Princípios FIRST

Cobertura de Código (20%)

- Cobertura de linhas $\geq 70\%$
- Cobertura de branches $\geq 60\%$
- Módulos críticos $\geq 85\%$
- Justificar código não coberto

Comandos Essenciais

pytest

`pytest -v`	Executar com verbose
`pytest tests/unit/`	Testar pasta específica
`pytest -k "senha"`	Testar por nome
`pytest -maxfail=1`	Parar no primeiro erro

coverage.py

`coverage run -m pytest`	Executar com cobertura
`coverage report`	Relatório no terminal
`coverage html`	Relatório HTML detalhado
`coverage report -show-missing`	Mostrar linhas não cobertas

Principais Aprendizados

1. Testes automatizados são **investimento**, não custo
2. Siga a **pirâmide de testes**: 70% unitários
3. Use **FIRST** como guia de qualidade
4. Cobertura é uma **métrica**, não uma meta absoluta
5. **TDD** força design melhor e testes mais focados

Lembre-se

"Código sem testes é código legado por definição"

— Michael Feathers

Recursos e Referências

Documentação:

- pytest: `https://docs.pytest.org/`
- coverage.py: `https://coverage.readthedocs.io/`
- Jest (JS): `https://jestjs.io/`

Livros:

- Clean Code, Robert C. Martin (Cap. 9)
- Test Driven Development, Kent Beck
- The Art of Unit Testing, Roy Osherove

Próximos Passos

Hoje

- Completar os 3 exercícios práticos
- Experimentar com coverage.py

Esta Semana

- Adicionar testes ao seu projeto
- Analisar cobertura atual
- Identificar gaps de teste

Para U3 (05/12)

- Suite completa de testes
- Cobertura $\geq 70\%$
- Documentação de qualidade

Dúvidas?

Atendimento: Segundas 14h-16h (online)

Discord: `https://discord.gg/bbMFJBQRT8`

GitHub: `https://github.com/fmarquesfilho/bpp-2025-2`

Bons testes!