INSTITUTO FEDERAL
Rio Grande do Norte | Campus Pau dos Ferros

TDD NA PRÁTICA

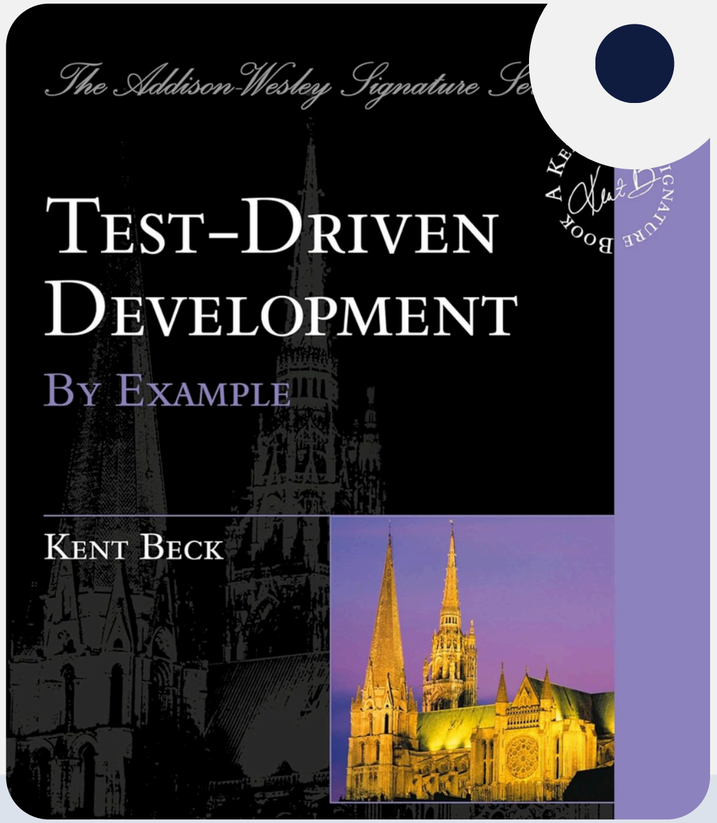
O poder do Red-Green-Refactor

TESTE DE SOFTWARE | JOEL JÚNIOR

O que é TDD?

Test-Driven Development é uma técnica onde os testes são escritos **antes** do código.

- **Origem:** Kent Beck (2002), livro "Test Driven Development by Example";
- **Ideia Central:** O teste define o design. O código só é escrito para satisfazer o teste.



Os 5 Passos do TDD

1. Escrever Testes

- Baseado nas especificações;
- Antes do código real.

2. Rodar Testes (Falha)

- Garante que o teste funciona;
- Confirma que a feature é nova.

3. Escrever Código

- O mais simples possível;
- Apenas para passar no teste.

Os 5 Passos do TDD

4. Rodar Testes (Sucesso)

- Garante adesão à especificação;
- Verifica regressões.

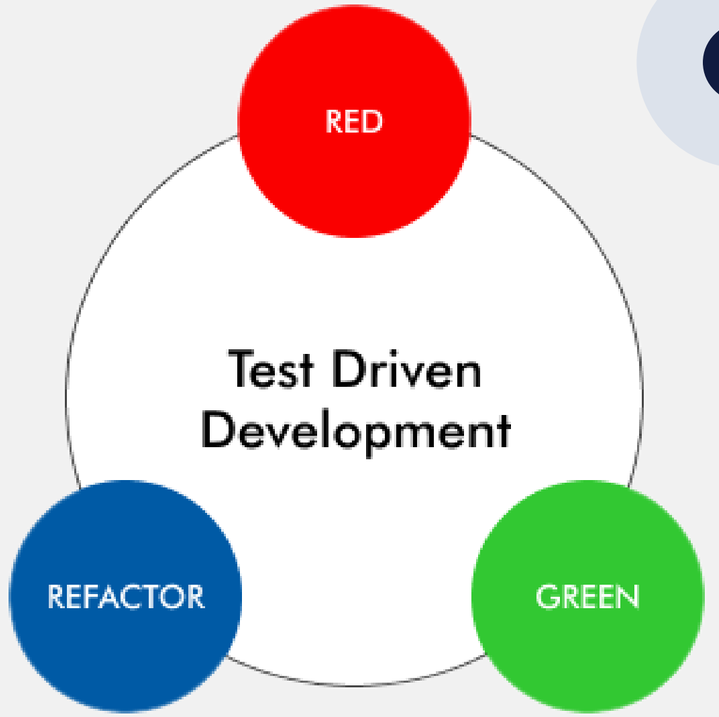
5. Refatorar

- Melhorar o código;
- Manter testes passando (Red-Green-Refactor).

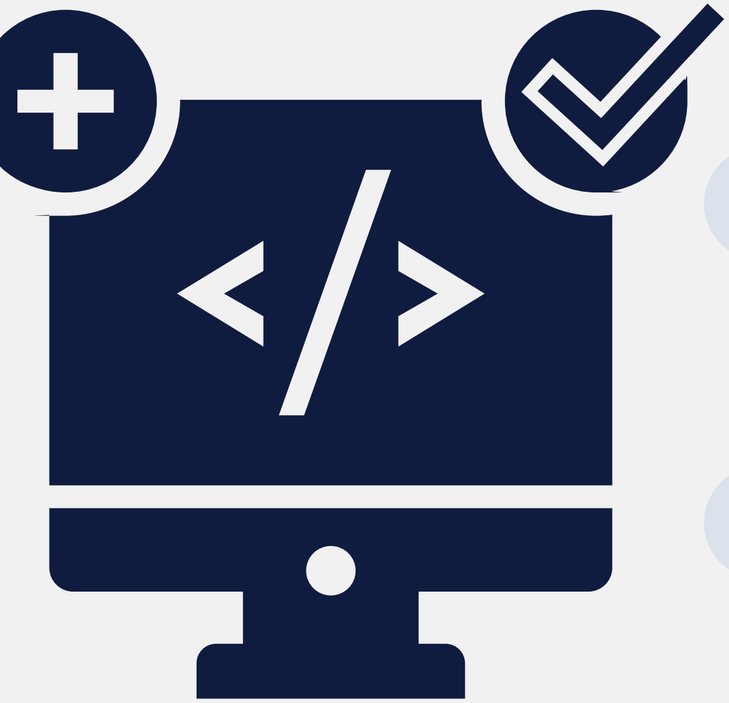
Ciclo Red-Green-Refactor

- **Red:** Escrever teste que falha;
- **Green:** Implementar solução mínima;
- **Refactor:** Limpar e melhorar o código.

Repetir em pequenos incrementos.



Por que TDD é popular? (Prós)



Foco nos Requisitos

- Define o "o que" antes do "como";
- Design de interfaces primeiro.

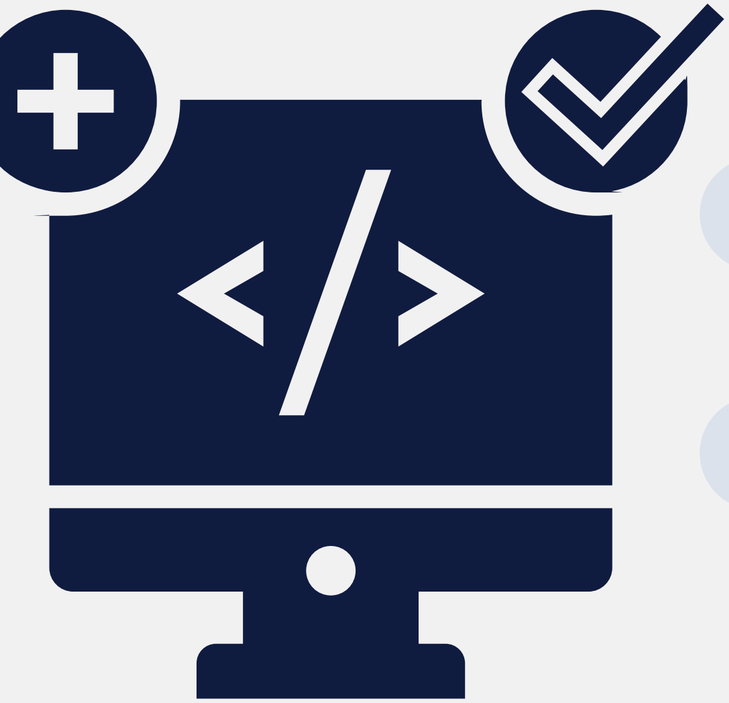
Detecção Precoce de Bugs

- Menos tempo debugando;
- Feedback rápido.

Melhor Design de Código

- Código testável por natureza;
- Encoraja desacoplamento.

Cuidados e Desafios (Contras)



Tempo e Custo

- Overhead de escrever/manter testes;
- Balancear em startups/fases iniciais.

Falsa Segurança

- Não substitui testes de integração/E2E.

Pontos Cegos

- Mesmo dev escrevendo teste e código;
- Risco de interpretar errado a spec.

O que NÃO fazer (1/3)

1. Reutilizar dados entre testes

❌ **Errado:** Instâncias compartilhadas que guardam estado.

```python
class TesteFuncionario(unittest.TestCase):
    # Instância compartilhada entre testes (Perigo!)
    funcionario = Funcionario("João")

    def test_bonus(self):
        # Altera o estado do objeto compartilhado
        self.funcionario.adicionar_bonus(500)
        self.assertEqual(self.funcionario.salario, 5500)

    def test_salario(self):
        # Falha pois o teste anterior alterou o valor
        self.assertEqual(self.funcionario.salario, 5000)
```

O que NÃO fazer (2/3)

2. Testar funções Built-in

❌ **Errado:** Testar se o Python funciona.

```python
def test_atributos(self):
    func = Funcionario(nome="João", id=1)

    # Desnecessário: O construtor apenas atribui variáveis.
    # Você está testando se o Python sabe atribuir variáveis.
    self.assertEqual(func.nome, "João")
    self.assertEqual(func.id, 1)
```

O que NÃO fazer (3/3)

3. Comparar com lógica reimplementada

❌ **Errado:** Copiar a lógica do método para o teste.

```
def test_pagamento_ruim(self):
    # Se a lógica estiver errada aqui e na classe, o teste passa falsamente
    esperado = self.func.taxa * self.func.horas
    self.assertEqual(self.func.calcular_pagamento(), esperado)

def test_pagamento_bom(self):
    # ✅ Correto: Comparar com valor fixo conhecido (Hardcoded)
    self.assertEqual(self.func.calcular_pagamento(), 5000)
```

Hora da Prática

Conclusão

- TDD ajuda a equilibrar velocidade e qualidade;
- Permite refatoração segura;
- Exige prática e disciplina;
- Separe a criação de testes da implementação quando possível.

INSTITUTO FEDERAL
Rio Grande do Norte | Campus Pau dos Ferros

OBRIGADO!

TESTE DE SOFTWARE | JOEL JÚNIOR