RELATÓRIO DE ANÁLISE E REFATORAÇÃO DE TEST SMELLS

Disciplina: Teste de Software
Aluno: Pedro Franco

---

1. ANÁLISE DE SMELLS IDENTIFICADOS

1.1 Teste Frágil (Brittle Test)

Localização: Teste "deve gerar um relatório de usuários formatado"

O teste verifica strings exatas incluindo formatação: `ID: ${usuario1.id}, Nome: Alice, Status: ativo\n` e valida o início com `startsWith('--- Relatório de Usuários ---')`. Qualquer mudança cosmética (espaços, pontuação, ordem) quebra o teste mesmo que a funcionalidade esteja correta.

Riscos: Manutenção custosa, falsos negativos, resistência a melhorias de UI.

1.2 Lógica Condicional no Teste

Localização: Teste "deve desativar usuários se eles não forem administradores"

Usa loop `for` com `if/else` para determinar asserções: `for (const user of todosOsUsuarios) { if (!user.isAdmin) { expect(...) } else { expect(...) } }`. Testes devem ser lineares e simples.

Riscos: Dificulta debugging, cobertura incerta, múltiplos motivos para falhar.

1.3 Asserção Ignorada (Silent Test)

Localização: Teste "deve falhar ao criar usuário menor de idade"

Usa `try/catch` com asserção apenas no `catch`. Se a exceção não for lançada, o teste passa silenciosamente sem executar verificações.

Riscos: Falsos positivos, regressões não detectadas, validações críticas quebradas em produção.

---

2. COMPARAÇÃO: ANÁLISE MANUAL vs. ESLint

Resultado do ESLint:

44:9 error Avoid calling `expect` conditionally jest/no-conditional-expect

46:9 error Avoid calling `expect` conditionally jest/no-conditional-expect

49:9 error Avoid calling `expect` conditionally jest/no-conditional-expect

73:7 error Avoid calling `expect` conditionally jest/no-conditional-expect

77:3 warning Tests should not be skipped jest/no-disabled-tests

77:3 warning Test has no assertions jest/expect-expect

Convergência: ESLint detectou com precisão a Lógica Condicional (linhas 44, 46, 49) e Asserção Ignorada (linha 73).

Limitações: A ferramenta não identificou Teste Frágil nem Teste Ansioso - problemas que requerem compreensão semântica do código.

Conclusão: Análise manual e automatizada são complementares, não substitutas.

---

3. PROCESSO DE REFATORAÇÃO

Exemplo: Correção da Lógica Condicional

ANTES:

```
test('deve desativar usuários se eles não forem administradores', () => {

 const usuarioComum = userService.createUser('Comum', 'comum@teste.com', 30);

 const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 40, true);

  const todosOsUsuarios = [usuarioComum, usuarioAdmin];
```

```
  for (const user of todosOsUsuarios) {
    const resultado = userService.deactivateUser(user.id);
    if (!user.isAdmin) {
      expect(resultado).toBe(true);
      const usuarioAtualizado = userService.getUserById(user.id);
      expect(usuarioAtualizado.status).toBe('inativo');
    } else {
      expect(resultado).toBe(false);
    }
  }
});
```

DEPOIS:

```
describe('Desativação de usuários', () => {
  test('deve desativar usuário comum com sucesso', () => {
    // Arrange
    const usuarioComum = userService.createUser('Comum',
'comum@teste.com', 30);
    // Act
    const resultado = userService.deactivateUser(usuarioComum.id);
    // Assert
    expect(resultado).toBe(true);
    const usuarioAtualizado = userService.getUserById(usuarioComum.id);
    expect(usuarioAtualizado.status).toBe('inativo');
```

```
  });


  test('deve impedir desativação de usuário administrador', () => {
    // Arrange
    const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 40,
true);
    // Act
    const resultado = userService.deactivateUser(usuarioAdmin.id);
    // Assert
    expect(resultado).toBe(false);
    const usuarioAtualizado = userService.getUserById(usuarioAdmin.id);
    expect(usuarioAtualizado.status).toBe('ativo');
  });
});
```

Decisões tomadas:

1. **Separação em testes independentes** - cada cenário tem seu próprio teste focado
2. **Aplicação do padrão AAA** - Arrange, Act, Assert claramente separados
3. **Eliminação de lógica** - removido loop e condicionais
4. **Organização com describe()** - agrupamento lógico dos testes relacionados

Outras correções aplicadas:

- **Teste Ansioso:** Separei criação e busca em testes distintos
- **Asserção Ignorada:** Substituí try/catch por `expect(() => {...}).toThrow()`
- **Teste Frágil:** Verifico comportamento (presença de dados) em vez de formatação exata
- **Teste Skipado:** Implementei o teste completo

---

4. VALIDAÇÃO FINAL

ESLint no arquivo refatorado:

✓ No errors or warnings found

Execução dos testes:

PASS __tests__/userService.smelly.test.js

PASS __tests__/userService.clean.test.js

Tests: 15 passed, 15 total

Todos os testes passam, comprovando que a refatoração manteve a funcionalidade enquanto eliminou os smells.

---

5. CONCLUSÃO

A experiência demonstrou que a qualidade dos testes é tão importante quanto a do código de produção. Testes mal escritos dificultam manutenção, diminuem confiança e impedem melhorias.

Principais aprendizados:

- Testes devem ser simples e lineares - sem lógica condicional
- O padrão AAA facilita dramaticamente leitura e manutenção
- Verificar comportamento, não implementação, reduz fragilidade
- Ferramentas automatizadas detectam problemas técnicos, mas análise manual é essencial para smells de design

Impacto na sustentabilidade: Testes limpos permitem refatorações confiáveis, servem como documentação viva, facilitam onboarding e mantêm a equipe confiante para fazer mudanças. O investimento em qualidade de testes é investimento no futuro do software.