

**Pontifícia Universidade Católica de Minas Gerais**

Engenharia de Software

# **Refatoração de Testes e Detecção de Test Smells**

Trabalho Prático de Teste de Software

**Aluno:** Gabriel Henrique Miranda Rodrigues

**Matrícula:** 814050

**Disciplina:** Teste de Software

Belo Horizonte  
2025

# 1 Análise de Test Smells

Durante a análise do arquivo `userService.smelly.test.js`, foram identificados três principais *Test Smells*:

## 1.1 Test Smell #1: Eager Test

Um único teste verifica múltiplas funcionalidades (criar E buscar usuário):

```

1 test('deve criar e buscar um usuario corretamente', () => {
2   const usuarioCriado = userService.createUser(...);
3   expect(usuarioCriado.id).toBeDefined();
4
5   const usuarioBuscado = userService.getUserById(usuarioCriado.id);
6   expect(usuarioBuscado.nome).toBe(dadosUsuarioPadrao.nome);
7 });

```

**Por que é um Test Smell:** Viola o princípio da responsabilidade única ao testar duas operações distintas em um único teste.

**Riscos:** Se falhar, não fica claro qual operação tem problema; dificulta manutenção; cobertura enganosa.

## 1.2 Test Smell #2: Conditional Test Logic

O teste contém loops e condicionais:

```

1 test('deve desativar usuarios se nao forem admin', () => {
2   const usuarios = [usuarioComum, usuarioAdmin];
3
4   for (const user of usuarios) {
5     const resultado = userService.deactivateUser(user.id);
6     if (!user.isAdmin) {
7       expect(resultado).toBe(true);
8     } else {
9       expect(resultado).toBe(false);
10    }
11  }
12 });

```

**Por que é um Test Smell:** Loops e condicionais introduzem complexidade. Testes devem ser lineares e previsíveis.

**Riscos:** Bugs ocultos (expects dentro de ifs podem nunca executar); difícil compreensão; manutenção arriscada.

## 1.3 Test Smell #3: Fragile Test

O teste está acoplado a detalhes de formatação:

```

1 test('deve gerar um relatorio formatado', () => {
2   const relatorio = userService.generateUserReport();
3   const linhaEsperada = 'ID: ${usuario1.id}, Nome: Alice, Status: ativo\n';
4   expect(relatorio).toContain(linhaEsperada);
5 });

```

**Por que é um Test Smell:** Verifica formato exato (espaços, vírgulas, quebras de linha). Qualquer mudança mínima quebra o teste.

**Riscos:** Falsos positivos; manutenção custosa; confiança reduzida na suíte de testes.

# 2 Processo de Refatoração

O teste escolhido foi ”**deve desativar usuários se não forem administradores**”, pois concentra três problemas graves: lógica condicional, múltiplas asserções condicionais, e falta de clareza.

## 2.1 Código ANTES

```

1 test('deve desativar usuarios se nao forem admin', () => {
2   const usuarioComum = userService.createUser('Comum', 'comum@teste.com', 30);
3   const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 40, true);
4
5   for (const user of [usuarioComum, usuarioAdmin]) {
6     const resultado = userService.deactivateUser(user.id);
7     if (!user.isAdmin) {
8       expect(resultado).toBe(true);
9       expect(userService.getUserById(user.id).status).toBe('inativo');
10    } else {
11      expect(resultado).toBe(false);
12    }
13  });
14 });

```

## 2.2 Código DEPOIS

```

1 test('deve desativar usuario nao-admin com sucesso', () => {
2   // Arrange
3   const user = userService.createUser('Carlos', 'carlos@example.com', 25, false);
4   // Act
5   const result = userService.deactivateUser(user.id);
6   // Assert
7   expect(result).toBe(true);
8   expect(user.status).toBe('inativo');
9 });
10
11 test('nao deve desativar usuario admin', () => {
12   // Arrange
13   const user = userService.createUser('Admin', 'admin@example.com', 30, true);
14   // Act
15   const result = userService.deactivateUser(user.id);
16   // Assert
17   expect(result).toBe(false);
18   expect(user.status).toBe('ativo');
19 });

```

## 2.3 Justificativa

**1. Eliminação da Lógica Condisional:** Separamos em dois testes independentes, eliminando loops e condicionais.

**2. Padrão AAA:** Cada teste segue Arrange-Act-Assert, tornando a estrutura clara e previsível.

**3. Dados Inline:** Cada teste define seus próprios dados, eliminando dependências externas.

**4. Nomes Descritivos:** Os nomes comunicam claramente o comportamento esperado.

## 3 Relatório da Ferramenta ESLint

### 3.1 Configuração do ESLint

O arquivo `eslint.config.js` foi configurado com as seguintes regras:

```

1 export default [
2   {
3     files: ['**/*.test.js'],
4     plugins: { jest },
5     rules: {
6       'jest/no-disabled-tests': 'warn',
7       'jest/no-conditional-expect': 'error',
8       'jest/no-conditional-in-test': 'error',
9       'jest/expect-expect': 'warn',
10      },
11    },
12  ];

```

### 3.2 Resultado antes da Refatoração

```
PS C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly> npx eslint test/userService.smelly.test.js
C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly\test\userService.smelly.test.js
  42:7  error  Avoid having conditionals in tests      jest/no-conditional-in-test
  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

✖ 7 problems (5 errors, 2 warnings)

PS C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly>
```

Figura 1: ESLint no arquivo smelly

Output do terminal:

```
C:\...\test\userService.smelly.test.js
  42:7  error  Avoid having conditionals in tests      jest/no-conditional-in-test
  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
    7 problems (5 errors, 2 warnings)
```

**Análise:** O ESLint detectou 5 erros de lógica condicional e 2 avisos sobre testes desabilitados/incompletos.

### 3.3 Resultado depois da Refatoração

```
PS C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly> npx eslint test/userService.clean.test.js
PS C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly>
```

Figura 2: ESLint no arquivo clean

O ESLint não reportou nenhum problema, confirmando que todos os smells foram corrigidos.

### 3.4 Execução dos Testes

```
PS C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly> npm test
> test-smells-lab@1.0.0 test
> jest

  PASS  test/userService.clean.test.js
  PASS  test/userService.smelly.test.js

  Test Suites: 2 passed, 2 total
  Tests:       1 skipped, 25 passed, 26 total
  Snapshots:  0 total
  Time:        1.025 s
  Ran all test suites.

PS C:\Users\gabri\OneDrive\Área de Trabalho\test-smell\test-smelly>
```

Figura 3: Resultado do comando npm test

Todos os 25 testes passaram com sucesso:

```
1 PASS  test/userService.clean.test.js
2 PASS  test/userService.smelly.test.js
3 Test Suites: 2 passed, 2 total
4 Tests:       1 skipped, 25 passed, 26 total
5 Time:        0.722 s
```

## 4 Conclusão

Este trabalho demonstrou como *Test Smells* comprometem a qualidade dos testes. A análise manual e automática (ESLint) permitiu identificar e corrigir problemas estruturais.

### Principais aprendizados:

- Testes também precisam de qualidade - código de teste mal escrito gera mais problemas do que soluções
- Ferramentas automatizam detecção, mas análise crítica é essencial
- O padrão AAA traz clareza e organização aos testes
- Testes atômicos facilitam identificação de problemas e manutenção

A integração de ferramentas de análise estática é essencial para manter qualidade. O ESLint atua como revisor automático, mas o conhecimento sobre *Test Smells* e boas práticas é fundamental. Tratar código de teste com o mesmo rigor do código de produção cria testes que agregam valor e sustentabilidade ao projeto.