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):

```
test('deve criar e buscar um usuario corretamente', () => {
  const usuarioCriado = userService.createUser(...);
  expect(usuarioCriado.id).toBeDefined();

  const usuarioBuscado = userService.getUserById(usuarioCriado.id);
  expect(usuarioBuscado.nome).toBe(dadosUsuarioPadrao.nome);
});
```

Por que é um mau cheiro: 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:

```
test('deve desativar usuarios se nao forem admin', () => {
  const usuarios = [usuarioComum, usuarioAdmin];

  for (const user of usuarios) {
    const resultado = userService.deactivateUser(user.id);
    if (!user.isAdmin) {
      expect(resultado).toBe(true);
    } else {
      expect(resultado).toBe(false);
    }
  }
});
```

Por que é um mau cheiro: 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:

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

Por que é um mau cheiro: 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

```
test('deve desativar usuarios se nao forem admin', () => {
  const usuarioComum = userService.createUser('Comum', 'comum@teste.com', 30);
  const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 40, true);

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

2.2 Código DEPOIS

```
test('deve desativar usuario nao-admin com sucesso', () => {
  // Arrange
  const user = userService.createUser('Carlos', 'carlos@example.com', 25, false);
  // Act
  const result = userService.deactivateUser(user.id);
  // Assert
  expect(result).toBe(true);
  expect(user.status).toBe('inativo');
});

test('nao deve desativar usuario admin', () => {
  // Arrange
  const user = userService.createUser('Admin', 'admin@example.com', 30, true);
  // Act
  const result = userService.deactivateUser(user.id);
  // Assert
  expect(result).toBe(false);
  expect(user.status).toBe('ativo');
});
```

2.3 Justificativa

1. Eliminação da Lógica Condicional: 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:

```
export default [
  {
    files: ['**/*.test.js'],
    plugins: { jest },
    rules: {
      'jest/no-disabled-tests': 'warn',
      'jest/no-conditional-expect': 'error',
      'jest/no-conditional-in-test': 'error',
      'jest/expect-expect': 'warn',
    },
  },
];
```

3.2 Resultado antes da Refatoração

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

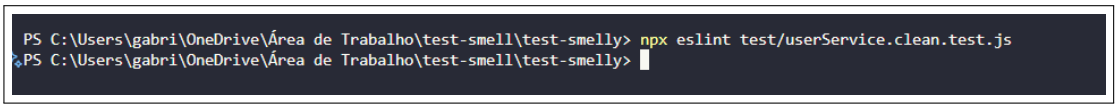

Figura 2: ESLint no arquivo clean

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

3.4 Execução dos Testes

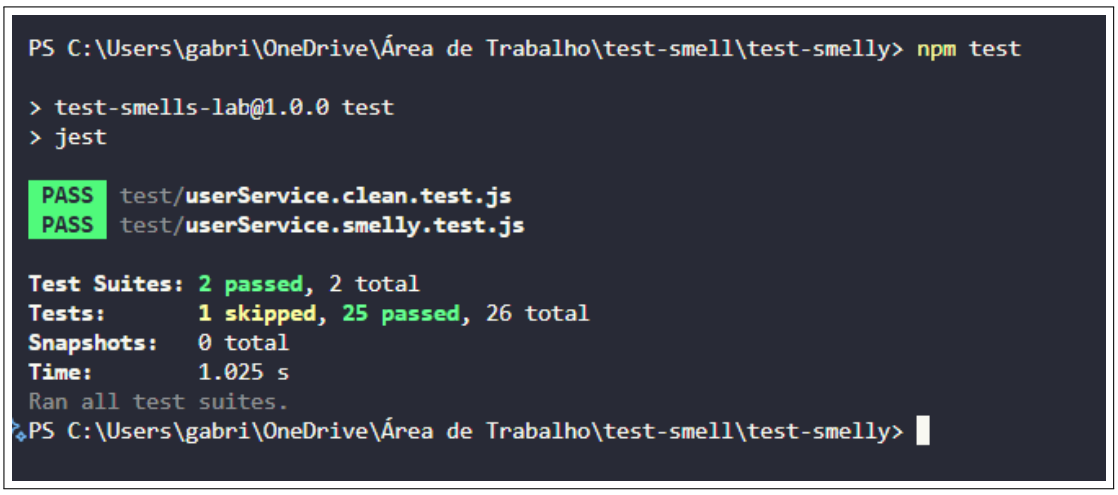

Figura 3: Resultado do comando `npm test`

Todos os 25 testes passaram com sucesso:

```
 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
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.