Teste de Software

Análise e Refatoração de Testes e Detecção de Test Smells

João Pedro Silva Braga
815051

Belo Horizonte, MG
2 de novembro de 2025

1 Análise de Smells

A análise manual da suíte de testes original, encontrada no arquivo `userService.smelly.test.js`, revelou diversos "maus cheiros"(Test Smells) que comprometem a qualidade, legibilidade e manutenibilidade dos testes. A seguir, são descritos três dos smells mais críticos identificados.

1.1 Eager Test (Teste Ansioso)

- **Onde foi encontrado:** No teste `'deve criar e buscar um usuário corretamente'`.
- **Descrição:** Este teste executa e verifica duas funcionalidades distintas em uma única asserção: a criação de um usuário (`createUser`) e a subsequente busca por esse mesmo usuário (`getUserById`).
- **Risco:** Ao acoplar múltiplos comportamentos, o teste perde o foco. Se ele falhar, não é imediatamente claro se o problema reside na funcionalidade de criação ou na de busca, tornando a depuração mais lenta e complexa. Um bom teste deve ter uma única responsabilidade.

1.2 Conditional Logic in Test (Lógica Condicional no Teste)

- **Onde foi encontrado:** No teste `'deve desativar usuários se eles não forem administradores'`.
- **Descrição:** O teste utiliza um laço de repetição (`for`) e uma estrutura condicional (`if/else`) para testar dois cenários distintos: a desativação bem-sucedida de um usuário comum e a falha ao tentar desativar um administrador.
- **Risco:** A presença de lógica complexa torna o teste difícil de ler e entender. Testes devem ser simples e lineares, seguindo o padrão Arrange-Act-Assert. A lógica condicional aumenta o risco de o próprio teste conter um bug e esconde a intenção real de cada cenário.

1.3 Fragile Test (Teste Frágil)

- **Onde foi encontrado:** No teste `'deve gerar um relatório de usuários formatado'`.
- **Descrição:** A verificação deste teste está fortemente acoplada à formatação exata de uma string de saída, incluindo espaços, vírgulas e a ordem das palavras.
- **Risco:** Qualquer alteração cosmética na formatação do relatório (como adicionar um espaço ou mudar "Nome:"para "Usuário:") quebrará o teste, mesmo que a funcionalidade principal (exibir as informações corretas) esteja intacta. Isso gera manutenção desnecessária e desencoraja refatorações no código de produção.

2 Processo de Refatoração

Para ilustrar o processo de melhoria, foi escolhido o teste com **Lógica Condicional**, pois era o mais complexo e menos legível.

2.1 Antes: Código com Smells

O código original utilizava um laço e um `if` para testar dois cenários de uma vez só.

Listing 1: Teste original com lógica condicional

```
test('deve␣desativar␣usuários␣se␣eles␣não␣forem␣administradores', () => {
  const usuarioComum = userService.createUser('Comum', 'comum@teste.com', 3
  const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 4

  const todosOsUsuarios = [usuarioComum, usuarioAdmin];

  // O teste tem um loop e um if, tornando-o complexo e menos claro.
  for (const user of todosOsUsuarios) {
    const resultado = userService.deactivateUser(user.id);
    if (!user.isAdmin) {
      // Este expect só roda para o usuário comum.
      expect(resultado).toBe(true);
      const usuarioAtualizado = userService.getUserById(user.id);
      expect(usuarioAtualizado.status).toBe('inativo');
    } else {
      // E este só roda para o admin.
      expect(resultado).toBe(false);
    }
  }
});
```

2.2 Depois: Código Refatorado

A solução foi dividir o teste original em dois testes menores, focados e com nomes descritivos. Cada um agora testa um único comportamento de forma clara e linear.

Listing 2: Testes refatorados, claros e focados

```
test('deve␣desativar␣um␣usuário␣comum␣com␣sucesso', () => {
  // Arrange
  const usuarioComum = userService.createUser('Comum', 'comum@teste.com', 3

  // Act
  const resultado = userService.deactivateUser(usuarioComum.id);
  const usuarioAtualizado = userService.getUserById(usuarioComum.id);

  // Assert
  expect(resultado).toBe(true);
  expect(usuarioAtualizado.status).toBe('inativo');
});

test('não␣deve␣desativar␣um␣usuário␣que␣seja␣administrador', () => {
  // Arrange
  const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 4
```

```
    // Act
    const resultado = userService.deactivateUser(usuarioAdmin.id);
    const usuarioAtualizado = userService.getUserById(usuarioAdmin.id);

    // Assert
    expect(resultado).toBe(false);
    expect(usuarioAtualizado.status).toBe('ativo'); // O status não deve mud
});
```

2.3 Decisões de Refatoração

A refatoração corrigiu o smell ao eliminar completamente a lógica condicional e o laço de repetição. As principais melhorias foram:

- **Responsabilidade Única:** Cada teste agora valida um único cenário.
- **Clareza:** Os nomes dos testes descrevem exatamente o comportamento esperado, e o padrão Arrange-Act-Assert é evidente.
- **Manutenibilidade:** Se a regra de negócio para desativar administradores mudar no futuro, apenas um teste pequeno e focado precisará ser ajustado, reduzindo o risco de quebrar outras validações.

3 Relatório da Ferramenta (ESLint)

Para automatizar a detecção de problemas, foi configurada a ferramenta de análise estática ESLint com o plugin `eslint-plugin-jest`. A execução da ferramenta no projeto original gerou o relatório apresentado na Figura 1.

```
PS C:\Users\jpsil\OneDrive\Área de Trabalho\Teste de Smell\test-smelly> npx eslint .

C:\Users\jpsil\OneDrive\Área de Trabalho\Teste de Smell\test-smelly\test\userService.smelly.test.js
  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

✖ 6 problems (4 errors, 2 warnings)
```

Figura 1: Resultado da execução do ESLint na suíte de testes original.

3.1 Comparação entre Análise Manual e Automática

A ferramenta automatizada foi eficaz em detectar alguns dos smells, mas não todos, reforçando a importância da combinação de ambas as abordagens.

- **Smells Detectados pelo ESLint:** A ferramenta identificou com precisão a **Lógica Condicional no Teste**, reportando erros do tipo `jest/no-conditional-expect` para cada `expect` dentro de blocos `if/else` e `try/catch`. Também detectou o **Teste Ignorado** (`test.skip`), emitindo um aviso de `jest/no-disabled-tests`.

- **Smells Não Detectados pelo ESLint:** O **Eager Test** e o **Fragile Test** não foram identificados. Isso ocorre porque são smells de natureza semântica e de design. O ESLint pode verificar se a sintaxe do teste é válida, mas não consegue interpretar a intenção do desenvolvedor ou o acoplamento excessivo à implementação.

Isso demonstra que, enquanto ferramentas automatizadas são excelentes para garantir a conformidade com regras objetivas, a análise humana continua sendo crucial para identificar problemas de design e estrutura nos testes.

4 Conclusão

Este trabalho demonstrou na prática como os Test Smells, mesmo em uma suíte de testes com 100% de aprovação, podem ocultar problemas de qualidade e aumentar o custo de manutenção do software. A utilização de ferramentas de análise estática como o ESLint se provou uma excelente aliada para detectar problemas objetivos, mas a análise crítica do desenvolvedor continua sendo indispensável para identificar smells de design.

Ao refatorar os testes para serem pequenos, focados e livres de lógica, transformamos uma suíte frágil e obscura em uma rede de segurança robusta e legível. A escrita de testes limpos não é apenas uma boa prática, mas um investimento direto na qualidade e sustentabilidade de um projeto de software.