Relatório: Refatoração de Testes e Detecção de Test Smells

Carlos Henrique Neimar Areas Ferreira

3 de novembro de 2025

Sumário

1 Análise de Smells

Conforme solicitado, a análise manual do arquivo `userService.smelly.test.js` revelou diversos "Test Smells". Abaixo, descrevemos três dos mais críticos identificados.

1.1 Lógica Condicional (Conditional Logic)

Descrição O teste `'deve desativar usuários se eles não forem administradores'` continha um laço `for` e uma estrutura `if...else` em seu corpo. As asserções (`expect`) eram executadas condicionalmente dentro desses blocos.

Risco Esse "smell" torna o teste complexo e difícil de ler. O principal risco é que, se a lógica de dados ou do teste mudar, um caminho do `if` pode nunca ser executado, escondendo uma falha. A ferramenta ESLint confirmou este problema com múltiplos erros `jest/no-conditional-expect`.

1.2 Teste Guloso (Eager Test)

Descrição Dois testes apresentavam esse "smell": `'deve criar e buscar um usuário corretamente'` e o já citado `'deve desativar usuários...'`. Ambos testavam múltiplos comportamentos (ex: criar e buscar; desativar comum e falhar com admin) em um único bloco `test`. Isso viola a diretriz de "separar testes gigantes (Eager Test) em testes menores e focados".

Risco Quando um "Eager Test" falha, é difícil identificar qual dos comportamentos testados é a causa raiz. Isso viola o Princípio da Responsabilidade Única e o padrão AAA, pois o teste possui múltiplos "Acts" e "Asserts" para diferentes cenários.

1.3 Manuseio Incorreto de Exceção

Descrição O teste `'deve falhar ao criar usuário menor de idade'` utilizava um bloco `try...catch` para validar se uma exceção era lançada, com o `expect` posicionado dentro do `catch`.

Risco Este é um "smell" perigoso que gera falsos-positivos silenciosos. Como o próprio comentário no código-fonte original alerta, se a lógica de validação de idade fosse removida (um bug), a função não lançaria uma exceção. O bloco `catch` seria ignorado, nenhuma asserção falharia, e o teste passaria "com sucesso", escondendo a regressão.

2 Processo de Refatoração

O teste mais problemático era o `'deve desativar usuários se eles não forem administradores'`, pois acumulava os "smells" de Lógica Condicional e Teste Guloso. Abaixo comparamos o "Antes" e o "Depois".

2.1 Antes: Código "Smelly"

O código original misturava dois cenários (usuário comum e admin) com um `if/else`.

```
test('deve desativar usuarios se eles nao 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];

  // 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 so roda para o usuario comum.
      expect(resultado).toBe(true);
      const usuarioAtualizado = userService.getUserById(user.id);
      expect(usuarioAtualizado.status).toBe('inativo');
    } else {
      // E este so roda para o admin.
      expect(resultado).toBe(false);
    }
  }
});
```

Listing 1: Código original com Lógica Condicional e Eager Test

2.2 Depois: Código Refatorado

A solução foi quebrar o teste "guloso" em dois testes menores e focados, cada um validando um único cenário.

```
test('deve desativar um usuario comum com sucesso', () => {
  // Arrange: Cria um usuario comum (nao-admin)
  const usuarioComum = userService.createUser('Comum', 'comum@teste
     .com', 30, false);

  // Act: Tenta desativar o usuario
  const resultado = userService.deactivateUser(usuarioComum.id);

  // Assert: Verifica se a desativacao foi permitida e o status
     mudou
```

```
  expect(resultado).toBe(true);
  const usuarioAtualizado = userService.getUserById(usuarioComum.id);
  expect(usuarioAtualizado.status).toBe('inativo');
});
```

Listing 2: Refatoração (Cenário 1): Teste focado no usuário comum

```
test('nao deve desativar um usuario administrador', () => {
  // Arrange: Cria um usuario administrador
  const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 40, true);

  // Act: Tenta desativar o usuario
  const resultado = userService.deactivateUser(usuarioAdmin.id);

  // Assert: Verifica se a desativacao foi bloqueada e o status permaneceu 'ativo'
  expect(resultado).toBe(false);
  const usuarioAtualizado = userService.getUserById(usuarioAdmin.id);
  expect(usuarioAtualizado.status).toBe('ativo');
});
```

Listing 3: Refatoração (Cenário 2): Teste focado no usuário admin

2.3 Justificativa da Refatoração

A refatoração seguiu rigorosamente o padrão Arrange, Act, Assert (AAA). Ao dividir o teste original em dois, eliminamos completamente a Lógica Condicional (`if/else`) e o `for`, resolvendo os erros `jest/no-conditional-expect` apontados pelo linter. Cada teste agora tem um propósito claro e único, tornando a suíte mais legível e fácil de manter.

3 Relatório da Ferramenta (ESLint)

A ferramenta de análise estática foi configurada com `eslint-plugin-jest`. A primeira execução no projeto (comando `npx eslint .`) retornou o seguinte resultado, focando no arquivo "smelly":

```
D:\Repositorios\Puc\Testes\Test 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)
```

Comentário sobre a Automação O ESLint foi crucial para automatizar a detecção de problemas de implementação. As regras `jest/no-conditional-expect` e `jest/no-disabled-tests` validaram imediatamente os "smells" de "Lógica Condicional", "Manuseio Incorreto de Exceção" e testes pulados que havíamos identificado manualmente.

Contudo, a ferramenta não foi capaz de identificar "smells" de *design*, como o "Teste Guloso" (Eager Test) ou o "Teste Frágil". Isso demonstra que a análise estática e a análise manual crítica são atividades complementares e essenciais para garantir a qualidade real do código de teste.

4 Conclusão

Este trabalho prático demonstrou a importância de ir além da cobertura de testes, focando na qualidade e design da suíte de testes. A passagem inicial dos testes "smelly" provou que "passar" não significa "estar correto", um conceito similar ao do Teste de Mutação visto anteriormente.

A refatoração para testes "limpos", aplicando o padrão AAA e eliminando "Test Smells", transforma a suíte de testes de uma obrigação frágil em um ativo de segurança robusto e de fácil manutenção.

A utilização de ferramentas de análise estática, como o ESLint, age como uma "rede de segurança" automatizada, garantindo que as boas práticas sejam mantidas. A combinação de testes limpos e automação de qualidade contribui diretamente para a sustentabilidade e manutenibilidade de longo prazo de qualquer projeto de software.