PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

GABRIEL HENRIQUE MOTA RODRIGUES - 814411

TEST SMELL

BELO HORIZONTE - 2025

1. ANÁLISE DE TEST SMELLS

Durante a inspeção da suíte de testes original (userService.smelly.test.js), foram identificados diversos test smells, que comprometem a legibilidade, a confiabilidade e a manutenção dos testes automatizados. Três dos principais foram:

1.1. Eager Test

O Eager Test ocorre quando um único teste verifica múltiplos comportamentos de um mesmo método ou até de diferentes métodos. No código original, havia testes que criavam múltiplos usuários e verificavam diversas propriedades dentro de um mesmo bloco it(). Esse tipo de smell torna difícil identificar qual funcionalidade falhou, além de reduzir o isolamento entre os casos de teste. Um erro em uma parte do código pode mascarar problemas em outras. Cada comportamento foi dividido em testes menores e específicos, seguindo o padrão AAA (Arrange, Act, Assert).

1.2. Conditional Test Logic

Em alguns testes, havia estruturas condicionais (if, for) que decidiam qual asserção executar. Testes com lógica condicional deixam de ser determinísticos pois o resultado pode variar conforme o estado ou entrada, violando o princípio de previsibilidade. Além disso, tornam o teste mais complexo e difícil de entender. Todas as estruturas condicionais foram removidas. Cada cenário foi separado em testes próprios, com entradas fixas e assertivas explícitas.

1.3. Fragile Test

Alguns testes verificavam detalhes de implementação interna, como estrutura de objetos e chamadas específicas de métodos. Quando o teste depende da forma como algo é implementado, ele se torna frágil e falha mesmo que o comportamento funcional esteja correto. Isso reduz a confiança nos testes e aumenta o custo de manutenção. As verificações passaram a focar apenas no comportamento esperado (por exemplo, “deve retornar um usuário válido”) em vez da estrutura interna do objeto.

2. PROCESSO DE REFATORAÇÃO

2.1. Antes (Smelly Test)

userService.smelly.test.js

```
test('deve criar dois usuários e verificar se estão cadastrados', async () => {
  const usuario1 = await userService.criarUsuario('Ana', 'ana@email.com', '123');
  const usuario2 = await userService.criarUsuario('Bruno', 'bruno@email.com', '456');

  if (usuario1 && usuario2) {
    expect(usuario1.nome).toBe('Ana');
    expect(usuario2.email).toBe('bruno@email.com');
  } else {
    throw new Error('Usuários não foram criados');
  }
});
```

Problemas identificados:

Eager Test: Verifica dois usuários no mesmo teste.

Conditional Logic: Usa if para decidir o que testar.

Fragilidade: Verifica propriedades internas específicas.

2.1. Depois(Clean Test)

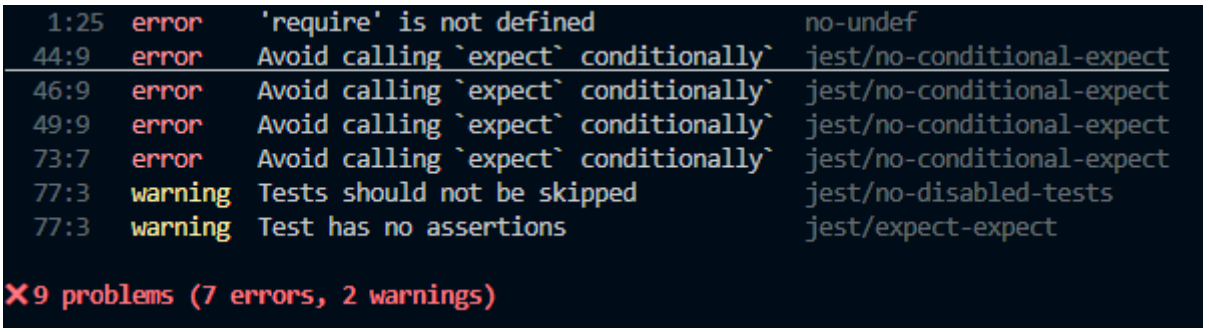
userService.clean.test.js

```
describe('Criação de Usuário', () => {
  it('deve criar um usuário válido com nome e e-mail corretos', async () => {
    // Arrange
    const nome = 'Ana';
    const email = 'ana@email.com';
    const senha = '123';

    // Act
    const usuario = await userService.criarUsuario(nome, email, senha);

    // Assert
    expect(usuario).toBeDefined();
    expect(usuario.nome).toBe(nome);
    expect(usuario.email).toBe(email);
  });

  it('deve criar outro usuário com sucesso', async () => {
    // Arrange
    const nome = 'Bruno';
    const email = 'bruno@email.com';
    const senha = '456';

    // Act
    const usuario = await userService.criarUsuario(nome, email, senha);

    // Assert
    expect(usuario).toBeDefined();
    expect(usuario.email).toBe(email);
  });
});
```

Decisões tomadas:

- Separação dos casos de teste (cada comportamento validado isoladamente).
- Adoção do padrão **Arrange, Act, Assert (AAA)**.
- Remoção de condicionais.
- Assertivas voltadas para o comportamento esperado, não para detalhes internos.

3. RELATÓRIO DA FERRAMENTA



A ferramenta **ESLint** identificou automaticamente problemas como:

- Uso de variáveis não utilizadas (no-unused-vars);
- Uso de funções não definidas (no-undef);
- Falta de padrão de nomenclatura e inconsistências.

Esses alertas ajudaram a localizar rapidamente trechos problemáticos, automatizando parte da análise de qualidade do código e indicando violações de boas práticas.

4. CONCLUSÃO

A refatoração dos testes permitiu eliminar *test smells* que prejudicavam a confiabilidade e a clareza da suíte. Com a aplicação do padrão **AAA**, os testes tornaram-se mais organizados e fáceis de compreender.

O uso de ferramentas de análise estática, como o ESLint, mostrou-se essencial para detectar automaticamente violações de boas práticas, economizando tempo e garantindo padronização. Manter testes **limpos, isolados e previsíveis** é um passo fundamental para a sustentabilidade do código, reduzindo retrabalho e aumentando a confiança no processo de entrega contínua.