PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
BACHARELADO EM ENGENHARIA DE SOFTWARE

MATHEUS NOLASCO MIRANDA SOARES

MATRÍCULA: 688826

Test Smells

Testes de Software

Análise de Smells

Durante a análise da suíte de testes original (userService.smelly.test.js), foram identificados três Test Smells principais.

Conditional Test Logic (Lógica Condicional em Teste)

Descrição:
No teste “deve desativar usuários se eles não forem administradores”, havia um loop for e condicionais if/else para verificar comportamentos diferentes no mesmo teste.

Por que é um mau cheiro:
Testes com lógica condicional são difíceis de ler e entender. Eles violam o princípio de que um teste deve ter um único caminho de execução e um único motivo para falhar.

Risco:
Quando um teste falha, não fica claro qual cenário específico quebrou. Isso dificulta a manutenção e aumenta o tempo de diagnóstico.

Eager Test (Teste Ansioso/Ganancioso)

Descrição:
No teste “deve criar e buscar um usuário corretamente”, eram verificadas duas funcionalidades distintas (createUser e getUserById) no mesmo teste.

Por que é um mau cheiro:
Misturar múltiplas responsabilidades em um único teste dificulta a identificação da causa de falhas e quebra o padrão AAA (Arrange, Act, Assert).

Risco:
Se o teste falhar, não é possível saber se o problema está na criação ou na busca do usuário sem depuração manual.

Fragile Test (Teste Frágil)

Descrição:
No teste “deve gerar um relatório de usuários formatado”, a verificação dependia da formatação exata da string de saída.

Por que é um mau cheiro:
Testes acoplados a detalhes de implementação (como formatação) quebram facilmente com mudanças não essenciais, como alteração de espaços ou ordem de campos.

Risco:
Aumenta a ocorrência de falsos negativos, onde o teste falha mesmo sem haver um bug real.

Processo de Refatoração

Escolhi como exemplo o teste com lógica condicional “deve desativar usuários se eles não forem administradores”, pois ele concentrava mais de um smell.

Antes (arquivo smelly)

```
test('deve desativar usuários se eles não 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];

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

Depois (arquivo clean)

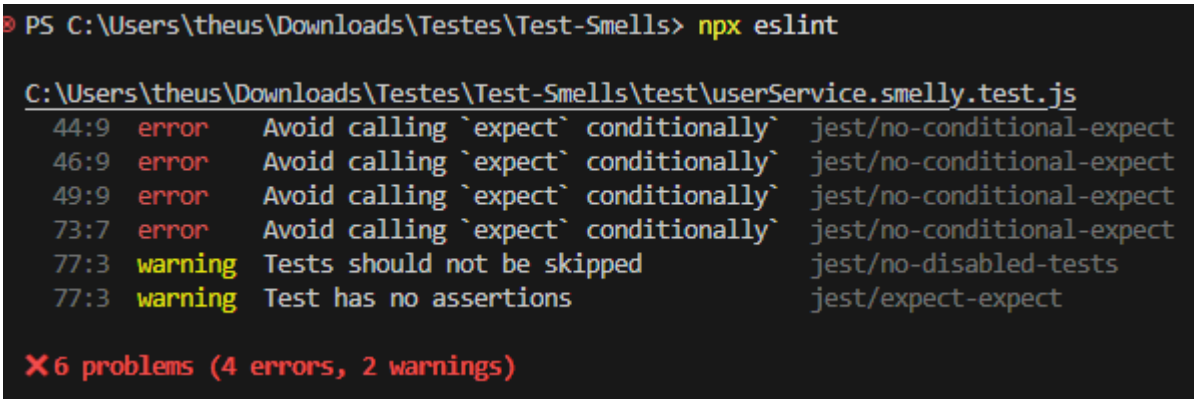
```
test('deve desativar um usuário comum com sucesso', () => {
const usuarioComum = userService.createUser('Comum', 'comum@teste.com',
30);
const resultado = userService.deactivateUser(usuarioComum.id);
expect(resultado).toBe(true);
});

test('deve alterar status para inativo ao desativar usuário comum', ()
=> {
const usuarioComum = userService.createUser('Comum', 'comum@teste.com',
30);
userService.deactivateUser(usuarioComum.id);
const usuarioAtualizado = userService.getUserById(usuarioComum.id);
expect(usuarioAtualizado.status).toBe('inativo');
});

test('deve impedir desativação de usuário administrador', () => {
const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com',
40, true);
const resultado = userService.deactivateUser(usuarioAdmin.id);
expect(resultado).toBe(false);
});
```

Decisões de Refatoração

- Remoção de lógica condicional: Dividi o teste em três, cada um cobrindo um cenário específico.

- Aplicação do padrão AAA: Cada teste segue claramente a sequência *Arrange* → *Act* → *Assert*.
- Nomes descritivos: Agora cada teste descreve exatamente o comportamento esperado.
- Melhoria da legibilidade: Com testes menores e focados, a leitura e manutenção ficaram mais simples.

3. Relatório da Ferramenta



O ESLint, com o plugin eslint-plugin-jest, detectou automaticamente:

- Uso de expect dentro de condicionais (jest/no-conditional-expect)
- Teste desabilitado (jest/no-disabled-tests)
- Teste sem assertions (jest/expect-expect)

Essa detecção automática agilizou a identificação de problemas, confirmando a análise manual e ajudando a priorizar a refatoração.

---

4. Conclusão

A escrita de testes limpos é fundamental para garantir que a suíte de testes seja confiável, fácil de manter e capaz de detectar falhas reais no sistema. A utilização de ferramentas de análise estática, como o ESLint com plugins para Jest, potencializa esse processo ao identificar automaticamente más práticas e inconsistências, permitindo agir preventivamente antes que se tornem problemas maiores.Com a refatoração aplicada, os testes ficaram mais claros, objetivos e robustos, reduzindo a chance de falsos positivos/negativos e aumentando a confiança no código.