Refatoração de Testes e Detecção de Test Smells

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS - PUC MINAS
DISCIPLINA: TESTE DE SOFTWARE
ALUNO: ALFREDO LUIS VIEIRA
MATRÍCULA: 807165

1 - Análise de Smells

Analisando a suíte de testes original, identificamos pelo menos três Test Smells evidentes. O primeiro é a Lógica Condicional no Teste, encontrado no test('deve desativar usuários...'). Este teste utiliza um loop for e uma estrutura if...else para executar expects de forma condicional, o que é uma má prática. O risco é que, se o teste falhar, não é imediatamente óbvio qual dos cenários (o de usuário comum ou o de administrador) causou o erro, dificultando a depuração.

O segundo smell é o Teste Frágil, visível no test('deve gerar um relatório...'). O teste está fortemente acoplado à implementação exata da saída, verificando uma string formatada (ID: ..., Nome: ...). O perigo aqui é que qualquer mudança trivial na formatação, como adicionar um espaço, quebrará o teste, mesmo que a lógica de negócio (gerar os dados corretos) esteja funcional, aumentando o custo de manutenção.

Por fim, o smell mais perigoso encontrado é a Assertiva Evasiva no test('deve falhar ao criar usuário menor de idade'). Ao colocar a asserção (expect) apenas dentro de um bloco catch, o teste só passa se o erro for lançado. Se um bug for introduzido e a função *parar* de lançar o erro, o bloco catch será ignorado e o teste passará silenciosamente, criando um falso positivo e escondendo a regressão.

2 - Processo de Refatoração:

O teste escolhido foi o mais problemático da suíte original é o test('deve desativar usuários se eles não forem administradores'), pois ele acumulava múltiplos "smells": era um Teste Eager (Testa Múltiplas Coisas) e usava Lógica Condicional (for e if/else), tornando-o difícil de ler e depurar.

O código "Antes" da refatoração era o seguinte:

```
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];

  // 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);
    }
  }
});
```

Seguindo as regras de refatoração, esse teste "gigante" foi quebrado em dois testes menores, focados e independentes, resultando no código Depois:

```
test('não deve desativar um usuário administrador', () => {
  const usuarioAdmin = userService.createUser('Admin', 'admin@teste.com', 40, true);

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

  expect(resultado).toBe(false);
  expect(usuarioAtualizado.status).toBe('ativo'); // Garante que o status não mudou
});
```

As decisões de refatoração foram focadas em corrigir os "smells" identificados. Primeiramente, o "Teste Gigante" (Eager Test) foi corrigido seguindo a regra de separar os testes. O teste original violava o Princípio da Responsabilidade Única ao validar duas lógicas de negócio opostas (desativar um admin E um usuário comum). Ao dividi-lo em dois testes, cada um agora possui um único e claro motivo para falhar, facilitando a depuração.

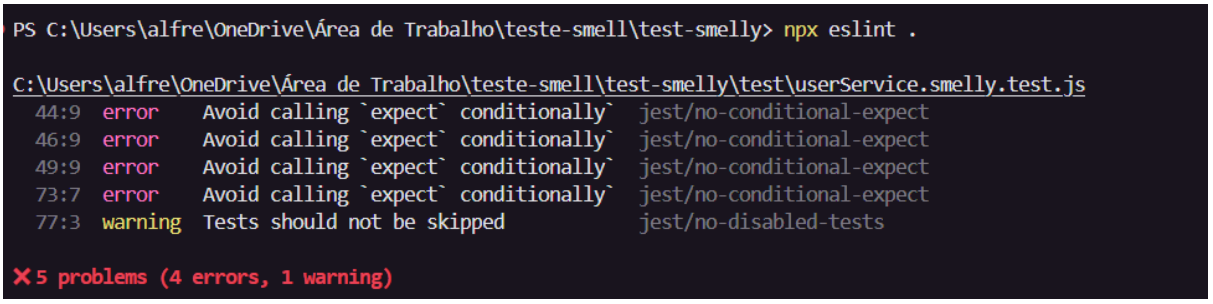
Em segundo lugar, essa divisão eliminou a Lógica Condicional. O for loop e a estrutura if...else não são mais necessários, o que torna o código de cada teste linear e muito mais fácil de ler. Isso também corrige diretamente o erro jest/no-conditional-expect que o ESLint apontou.

Finalmente, cada novo teste agora aplica rigorosamente o padrão Arrange, Act, Assert (AAA). A seção "Arrange" prepara apenas o cenário específico (um usuário comum ou um

admin). A seção "Act" executa a única ação sendo testada. E a seção "Assert" verifica de forma direta todos os resultados esperados para aquela ação. Os nomes dos testes também foram alterados para serem mais descritivos, indicando exatamente o comportamento esperado.

3 - Relatório da Ferramenta:

Para complementar a análise manual, foi executada a ferramenta de análise estática ESLint no arquivo userService.smelly.test.js, como pode ser visto no *screenshot* abaixo. A ferramenta automatizou e validou imediatamente a detecção dos "Test Smells" que havíamos identificado. O ESLint reportou 4 erros graves de jest/no-conditional-expect, apontando exatamente para as asserções (expect) que estavam "escondidas" dentro dos blocos if...else e do bloco try...catch. A ferramenta nos alerta que chamadas condicionais de expect são perigosas, pois podem levar a falsos positivos (testes que passam mesmo com bugs), confirmando os riscos da "Lógica Condicional" e da "Assertiva Evasiva". Além disso, a ferramenta emitiu um aviso (jest/no-disabled-tests) para o test.skip, reforçando que testes pulados são débitos técnicos. Isso demonstra como a análise estática automatizada é crucial para identificar e impor padrões de código limpo de forma instantânea, antes mesmo da execução da suíte.



4 - Conclusão:

Este trabalho demonstrou na prática como a identificação de "Test Smells" é crucial para a saúde de uma suíte de testes. A análise inicial identificou manualmente problemas graves, como Lógica Condicional, Testes Frágeis e Assertivas Evasivas. Essa análise foi posteriormente validada de forma concreta pela ferramenta ESLint, que reportou erros como jest/no-conditional-expect e avisos como jest/no-disabled-tests, provando que "smells" não são apenas teóricos, mas sim falhas programáticas reais. O processo de refatoração, ao focar na separação de responsabilidades e na aplicação rigorosa do padrão AAA, corrigiu diretamente cada um desses problemas, resultando em testes mais limpos, focados e sem erros de lint.

Como resultado, a suíte de testes final é fundamentalmente mais robusta, legível e confiável. Ao eliminar a Lógica Condicional, o risco de falhas de depuração complexa foi mitigado. Mais importante, ao corrigir a "Assertiva Evasiva" (o try...catch), o risco de falsos positivos foi eliminado, garantindo que o teste agora falhe se um bug for introduzido. Este exercício evidencia que o objetivo do teste de software não é apenas fazer os testes

passarem, mas construir uma rede de segurança de alta qualidade, de fácil manutenção e em que a equipe possa confiar.