Pontifícia Universidade Católica de Minas Gerais
Coração Eucarístico

Pedro Henrique Pires Rodrigues
816373

Teste de Software
Test Smells

Belo Horizonte
2025

Análise de Smells:

1) Eager Test: Logo no primeiro teste é possível perceber que a duas funcionalidades sendo testadas, a criação de um usuário e a busca dele na base de dados.

```
test('deve criar e buscar um usuário corretamente', () => {
  // Act 1: Criar
  const usuarioCriado = userService.createUser(
    dadosUsuarioPadrao.nome,
    dadosUsuarioPadrao.email,
    dadosUsuarioPadrao.idade
  );
  expect(usuarioCriado.id).toBeDefined();

  // Act 2: Buscar
  const usuarioBuscado = userService.getUserById(usuarioCriado.id);
  expect(usuarioBuscado.nome).toBe(dadosUsuarioPadrao.nome);
  expect(usuarioBuscado.status).toBe('ativo');
});
```

2) Fragile Test: O teste abaixo verifica, além se os dados batem com o exigido, se a frase gerada segue uma certa estrutura. Esta que pode ser modificada futuramente. Por isso verificar essa estrutura não é interessante, pois ainda que a ordem dos atributos seja diferente, para o teste, deve importar somente se os elementos corretos estão presentes no relatório.

```
test('deve gerar um relatório de usuários formatado', () => {
  const usuario1 = userService.createUser('Alice', 'alice@email.com', 28);
  userService.createUser('Bob', 'bob@email.com', 32);

  const relatorio = userService.generateUserReport();

  // Se a formatação mudar (ex: adicionar um espaço, mudar a ordem), o teste quebra.
  const linhaEsperada = `ID: ${usuario1.id}, Nome: Alice, Status: ativo\n`;
  expect(relatorio).toContain(linhaEsperada);
  expect(relatorio.startsWith('--- Relatório de Usuários ---')).toBe(true);
});

```

3) Developers Not Writing Tests: Test não implemen

```
test.skip('deve retornar uma lista vazia quando não há usuários', () => {
  // TODO: Implementar este teste depois.
});
```

Processo de Refatoração:

ANTES:

```
test('deve falhar ao criar usuário menor de idade', () => {
  // Este teste não falha se a exceção NÃO for lançada.
  // Ele só passa se o `catch` for executado. Se a lógica de validação
  // for removida, o teste passa silenciosamente, escondendo um bug.
  try {
    userService.createUser('Menor', 'menor@email.com', 17);
  } catch (e) {
    expect(e.message).toBe('O usuário deve ser maior de idade.');
  }
});

```

DEPOIS:

```
test('deve falhar ao criar usuário menor de idade', () => {
    expect(() => {
        userService.createUser('Menor', 'menor@email.com', 17);
    }).toThrow('O usuário deve ser maior de idade.');
});
```

Para este caso o problema era uma falha silenciosa. Se por acaso a exceção não fosse lançada, como deveria, o teste não acusaria falha ainda sim. Para corrigilo troquei o *try catch* por uma estrutura apropriada do próprio *Jest*, tornando agora detectável o caso do teste indevidamente não lançar exceção.

Relatório da Ferramenta:

Sem a ferramenta seria necessário que um desenvolvedor experiente olhasse teste por teste para descobrir cada um desses *test smells*, mas com a ajuda da ferramenta qualquer um poderia gerar esse relatório com um simples comando de terminal

Conclusão

A escrita de testes limpos e o uso de análise estática fortalecem a qualidade e a sustentabilidade de um software, ao prevenir erros, facilitar a manutenção e garantir um desenvolvimento mais confiável e duradouro.