PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Atividade Prática

Teste de Software

Refatoração de Testes e Detecção de Test Smells

João Marcos de Aquino Gonçalves

Belo Horizonte
2025

Test Smells identificados

Smell 1: Eager Test (Teste Gigante):
Antes:

```
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');
  });
```

É um smell pois o teste cobre dois comportamentos distintos, criar e buscar um usuário. O seu risco é se caso um dos testes falhar, o outro também falhará, dificultando a identificação do problema.
Smell 2: Conditional Test Logic (if / for dentro do teste):
Antes:

```
  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);
      }
    }
```

```
  });
```

É um smell pois testes não devem ter condições lógicas e loops dentro deles, pois isso pode mascarar falhas devido a nem todas as condições serem atendidas.

Smell 3: Test Fragility (Dependência de Formatação Exata):
Antes:

```
  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);
  });
```

É um smell pois o teste depende de uma linha exata dentro do código, caso a formatação seja alterada o teste para de funcionar.

Refatoração:
Smell 1: Eager Test (Teste Gigante):
Anteriormente, o teste de criar e buscar usuário possuia duas etapas, criar o usuário e buscar o usuário. Em sua refatoração, separei esse método em dois métodos específicos para cada ação. Dessa forma, a lógica é testada devidamente para cada função, com o AAA explícito e menos risco de erro no teste.

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

    expect(usuarioCriado.id).toBeDefined();
    expect(usuarioCriado.nome).toBe(dadosUsuarioPadrao.nome);
    expect(usuarioCriado.status).toBe('ativo');
  });

  test('deve buscar um usuário corretamente', () => {
    const usuarioCriado = userService.createUser(
      dadosUsuarioPadrao.nome,
      dadosUsuarioPadrao.email,
      dadosUsuarioPadrao.idade
    );

    const usuarioBuscado = userService.getUserById(usuarioCriado.id);
    expect(usuarioBuscado).toEqual(usuarioCriado);
  });
```

Relatório da Ferramenta:
Print da primeira execução:

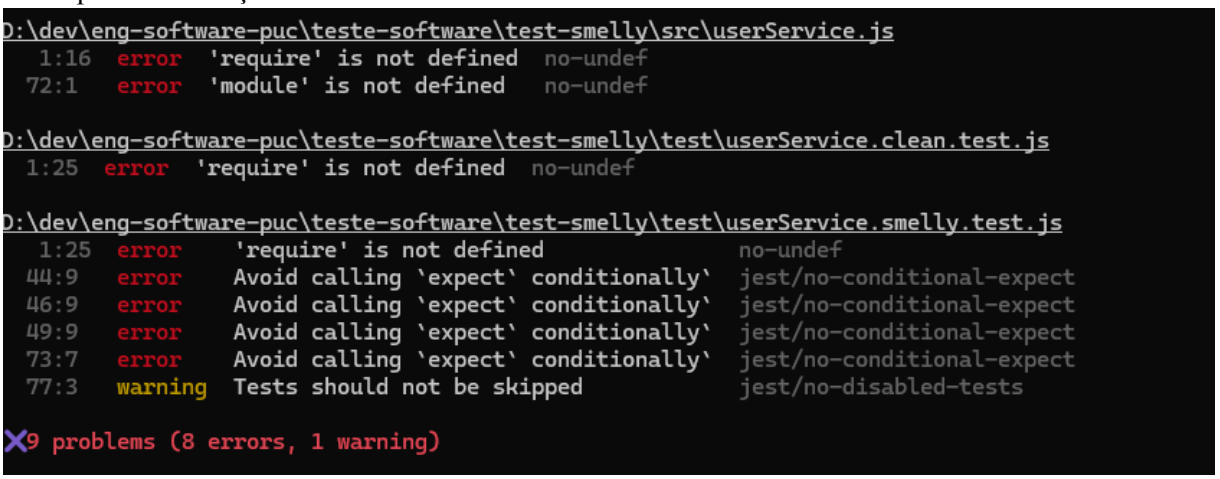
```
D:\dev\eng-software-puc\teste-software\test-smelly\src\userService.js
   1:16  error  'require' is not defined  no-undef
  72:1   error  'module' is not defined   no-undef

D:\dev\eng-software-puc\teste-software\test-smelly\test\userService.clean.test.js
   1:25  error    'require' is not defined              no-undef
  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

D:\dev\eng-software-puc\teste-software\test-smelly\test\userService.smelly.test.js
   1:25  error    'require' is not defined              no-undef
  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

✖14 problems (12 errors, 2 warnings)
```

Print após a refatoração:



Conclusão:
É nítido como um teste bem estruturado pode afetar o processo de desenvolvimento e validação do seu software. A ferramenta apresentada na atividade se mostrou excelente em detectar os smells e, em poucos passos, ajudou a tornar pequenos casos de testes mais eficientes, limpos e fáceis de ler e dar manutenção para garantir a eficácia dos testes.