Testes de Software

Refatoração de Testes e Detecção de Test Smells

Lucas Ferreira Garcia
804803

1. **Análise de Smells**

1.1 Lógica condicional em testes

Trecho:

```
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);
  }
}
```

Motivo do smell:
Testes não devem conter lógica condicional (if, for), pois isso gera múltiplos caminhos de execução e reduz a clareza.
Risco:
Pode gerar falsos positivos, dificultar entendimento e mascarar falhas dependendo do ramo executado.

1.2 Teste frágil:

Trecho:

```
const linhaEsperada = `ID: ${usuario1.id}, Nome: Alice, Status: ativo\n`;
expect(relatorio).toContain(linhaEsperada);
expect(relatorio.startsWith('--- Relatório de Usuários ---')).toBe(true);
```

Motivo do smell:
Esse teste depende da formatação literal da string, quebrando com alterações cosméticas.
Risco:
Quebra facilmente e impede refatorações seguras mesmo quando o comportamento permanece correto.

2.3 Teste de exceção mal estruturado

Trecho:

```
try {
  userService.createUser('Menor', 'menor@email.com', 17);
} catch (e) {
  expect(e.message).toBe('O usuário deve ser maior de idade.');
}
```

Motivo do smell:
Se a exceção não for lançada, o teste passa mesmo assim, o que é incorreto.
Risco:
Confiabilidade comprometida, já que o teste pode indicar sucesso mesmo com erro real no sistema.

2. **Processo de refatoração:**

Trecho escolhido: validação de idade
Antes:

```
try {
  userService.createUser('Menor', 'menor@email.com', 17);
} catch (e) {
  expect(e.message).toBe('O usuário deve ser maior de idade.');
}
```

Depois:

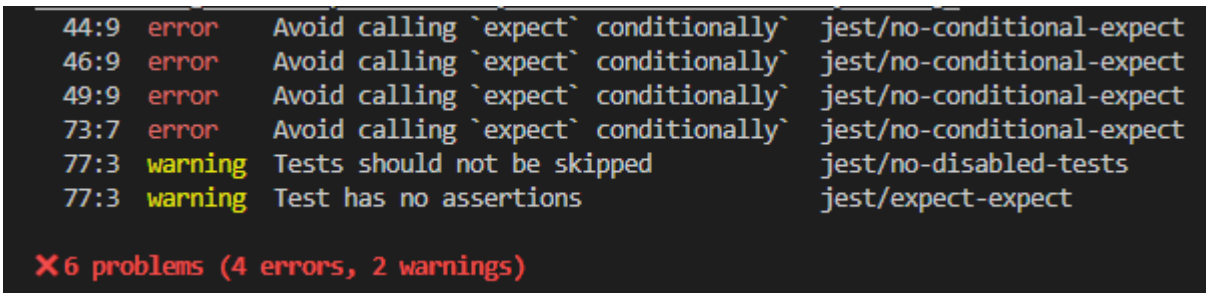
```
// Validação de idade
test('deve lançar erro ao criar usuário menor de idade', () => {
  // Arrange
  const act = () =>
    userService.createUser('Menor', 'menor@email.com', 17);

  // Act + Assert
  expect(act).toThrow('O usuário deve ser maior de idade.');
});
```

Justificativa da Refatoração

O teste foi reestruturado utilizando a função toThrow(), recomendada para validação de exceções no Jest. Essa abordagem elimina a necessidade de try/catch e o risco de o teste passar sem que a exceção seja de fato lançada, tornando o teste mais confiável e seguro.
Ao isolar o comportamento sob teste em uma função (act) e utilizar expect diretamente nela, seguimos o padrão Arrange, Act, Assert (AAA) e aumentamos a clareza da intenção do teste.

3. **Relatório da ferramenta**



A ferramenta ajudou a identificar problemas como:

- Expectativas condicionais
- Testes desabilitados
- Más práticas de estrutura

Isso reforçou a automação na análise de qualidade do código de teste.

4. **Conclusão**

A atividade reforçou a importância de escrever testes limpos e estruturados. Com a refatoração:

- Os testes se tornaram mais legíveis, confiáveis e mantíveis
- A lógica de negócio pôde ser validada sem ruído e sem falsos positivos
- A aplicação do padrão AAA organizou o raciocínio de teste
- O ESLint atuou como ferramenta essencial para detecção automática de smells

Testes bem escritos permitem evolução segura do software, evitam regressões e aumentam a produtividade da equipe. Ferramentas de linting aplicadas aos testes contribuem para garantir disciplina e padronização no desenvolvimento.