Refatoração de Testes e Detecção de Test Smells

Autor: Miguel Vieira
Disciplina: Teste de Software
Data: Novembro de 2025

Relatório Técnico

Este relatório apresenta o processo de identificação e refatoração de Test Smells em um conjunto de testes automatizados do projeto **UserService**, conforme atividade prática da disciplina de Teste de Software.

1. Análise de Smells

Durante a inspeção manual do arquivo `userService.smelly.test.js`, foram identificados os seguintes Test Smells:

Conditional Test Logic
Descrição: O teste que desativa usuários contém um loop e estruturas condicionais (`for` e `if`), o que faz com que comportamentos diferentes sejam testados dentro do mesmo caso. Isso dificulta a rastreabilidade de falhas e compromete a clareza do propósito do teste.
Correção: A refatoração dividiu o teste em dois: um verificando o comportamento para usuários comuns e outro para administradores. Cada caso agora possui sua própria asserção e é executado de forma independente.

Exception Handling Swallowing
Descrição: O teste que valida a criação de usuários menores de idade utiliza `try/catch` sem garantir que o teste falhe caso a exceção não seja lançada. Assim, se a validação for removida, o teste passa silenciosamente, mascarando erros.
Correção: A refatoração substituiu o bloco `try/catch` pelo matcher `toThrow()` do Jest, garantindo que o teste realmente falhe caso a exceção esperada não ocorra.

Fragile Test
Descrição: O teste que valida o relatório de usuários depende da formatação textual exata da string de saída. Pequenas mudanças cosméticas (como espaços ou ordem de linhas) podem quebrar o teste mesmo quando o comportamento funcional permanece correto.
Correção: Na refatoração, o teste passou a verificar apenas a presença dos dados essenciais (nomes e status) usando expressões regulares, tornando-o mais robusto e menos sensível à formatação.

2. Exemplo de Refatoração

O trecho a seguir ilustra uma das refatorações realizadas, substituindo o teste original (com lógica condicional) por duas versões independentes e mais legíveis.

Antes:

```
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:

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

test('não deve desativar 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');
});
```

3. Resultados do ESLint

A ferramenta ESLint foi configurada com o plugin Jest para detectar problemas comuns em testes automatizados. Ao executar `npx eslint .`, foram encontrados avisos no arquivo original (`smelly`), como o uso de lógica condicional e testes desabilitados (`test.skip`). Após a refatoração, o arquivo `userService.clean.test.js` não apresentou nenhum erro ou aviso.

4. Conclusão

O processo de refatoração dos testes trouxe melhorias significativas na legibilidade, isolamento e confiabilidade dos casos de teste. A utilização do ESLint como ferramenta de apoio permitiu identificar automaticamente práticas incorretas e garantir a padronização do código. Essa

atividade reforçou a importância de escrever testes limpos, claros e com responsabilidade única, contribuindo diretamente para a qualidade e a manutenibilidade do software.