RELATÓRIO DE REFATORAÇÃO DE TESTES

Detecção de Test Smells e Análise Estática com ESLint

Aluno: Wanessa Dias Costa
Matrícula: 815234

Disciplina: Teste de Software
Curso: Engenharia de Software
Universidade: Pontifícia Universidade Católica de Minas Gerais

Belo Horizonte, MG – 2 de novembro de 2025

Sumário

1 Análise Inicial e Detecção Automática

O objetivo deste trabalho foi analisar uma suíte de testes deliberadamente "mal cheirosa"(`userService.smelly.`
identificar os *Test Smells* e refatorá-la.

O primeiro passo foi configurar uma ferramenta de análise estática (ESLint) para automatizar a detecção de problemas.

1.1 Configuração do ESLint

Foram instalados o `eslint` e o `eslint-plugin-jest`. O arquivo `.eslintrc.json` foi configurado para estender as regras recomendadas do Jest e adicionar regras específicas para detectar "smells", como `jest/no-conditional-expect`.

1.2 Relatório da Ferramenta (Execução Inicial)

Ao executar o ESLint no projeto (`npx eslint .`), a ferramenta imediatamente reportou uma série de erros e avisos no arquivo `smelly`, como visto na Figura 1.

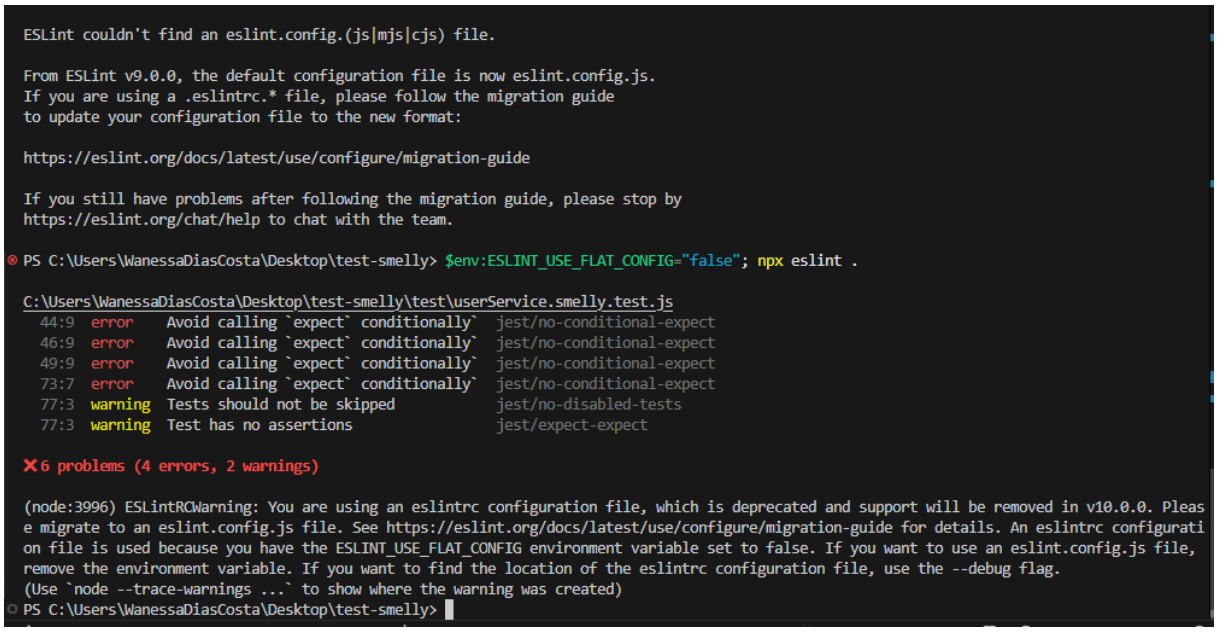

Figura 1: Relatório inicial do ESLint no arquivo `userService.smelly.test.js`.

Os erros reportados confirmaram os principais "smells":

- **jest/no-conditional-expect:** O "smell"de **Lógica Condicional** foi detectado. Testes possuíam `if/else` que executavam `expect`s diferentes, tornando o teste complexo e não determinístico.
- **jest/no-disabled-tests:** O "smell"de **Teste Desabilitado** (`test.skip`) foi encontrado, representando dívida técnica e uma funcionalidade não testada.
- **jest/expect-expect:** Um teste foi encontrado sem nenhuma asserção (`expect`), tornando-o inútil, pois ele sempre passaria (falso positivo).

—

2 Análise Manual dos Test Smells

Além da detecção automática, a análise manual do código `smelly` revelou problemas mais profundos de design, que a ferramenta não pegaria.

2.1 Smell 1: Lógica Condicional (Conditional Logic)

- **Descrição:** O teste `'deve desativar usuários se eles não forem administradores'` usava um `for` e um `if/else` para testar dois cenários (usuário comum e admin) de uma só vez.
- **Risco:** Torna o teste obscuro. Se o teste falhar, é difícil saber qual cenário causou a falha. Viola o princípio de que um teste deve validar uma única coisa.

2.2 Smell 2: Teste Gigante (Eager Test)

- **Descrição:** O teste `'deve criar e buscar um usuário corretamente'` realizava duas ações (Act) distintas: primeiro um `createUser` e depois um `getUserById`.
- **Risco:** Viola o padrão Arrange-Act-Assert (AAA). Se o teste falhar, não sabemos se o problema está na criação ou na busca. Testes devem ter responsabilidade única.

2.3 Smell 3: Teste de Exceção Perigoso (try/catch)

- **Descrição:** O teste `'deve falhar ao criar usuário menor de idade'` usava um bloco `try/catch` para validar a exceção.
- **Risco:** Se a lógica de validação de idade fosse removida (bug), a função não lançaria um erro. O bloco `catch` (onde o `expect` estava) nunca seria executado, e o teste passaria silenciosamente, escondendo a regressão.

—

3 Processo de Refatoração (Padrão AAA)

Para corrigir os "smells", o arquivo `__tests__/userService.clean.test.js` foi criado, aplicando rigorosamente o padrão Arrange, Act, Assert (AAA) e eliminando os problemas detectados.

3.1 Estratégia de Refatoração

A estratégia consistiu em:

1. **Separar Testes:** Dividir "Testes Gigantes"e "Lógica Condicional"em testes menores e focados, cada um com um único "Act".
2. **Remover Lógica:** Eliminar qualquer `if, for, try/catch` de dentro dos testes.
3. **Usar "Matchers"Corretos:** Substituir `try/catch` pelo "matcher"`.toThrow()` do Jest, que é a forma correta de testar exceções.
4. **Implementar Testes Faltantes:** Corrigir os testes desabilitados (`test.skip`).

3.2 Exemplo de Refatoração (Lógica Condicional)

O teste mais problemático foi o de desativação de usuário.

Listing 1: Antes: `userService.smelly.test.js`

```
test('deve desativar usu rios se eles n o 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];

  // SMELL: Loop e L gica Condicional
  for (const user of todosOsUsuarios) {
    const resultado = userService.deactivateUser(user.id);
    if (!user.isAdmin) {
      // Erro do ESLint aqui
      expect(resultado).toBe(true);
      const usuarioAtualizado = userService.getUserById(user.id);
      expect(usuarioAtualizado.status).toBe('inativo');
    } else {
      // Erro do ESLint aqui
      expect(resultado).toBe(false);
    }
  }
});
```

O código foi refatorado em dois testes claros, lineares e independentes:

Listing 2: Depois: `userService.clean.test.js`

```
it('deve desativar um usu rio comum com sucesso', () => {
  // ARRANGE
  const usuarioComum = userService.createUser('Comum',
     'comum@teste.com', 30, false);

  // ACT
  const resultado = userService.deactivateUser(usuarioComum.id);

  // ASSERT
  expect(resultado).toBe(true);
  const usuarioAtualizado = userService.getUserById(usuarioComum.id);
  expect(usuarioAtualizado.status).toBe('inativo');
});

it('n o deve desativar um usu rio administrador', () => {
  // ARRANGE
  const usuarioAdmin = userService.createUser('Admin',
     'admin@teste.com', 40, true);

  // ACT
  const resultado = userService.deactivateUser(usuarioAdmin.id);

  // ASSERT
  expect(resultado).toBe(false);
  const usuarioAtualizado = userService.getUserById(usuarioAdmin.id);
  expect(usuarioAtualizado.status).toBe('ativo');
});
```

—

4 Validação e Resultados Finais

Após a refatoração completa do arquivo `userService.clean.test.js`, as ferramentas de validação foram executadas novamente.

4.1 Validação com ESLint

Ao executar o ESLint especificamente no novo arquivo, nenhum erro ou aviso foi reportado:

```
$ $env:ESLINT_USE_FLAT_CONFIG="false"; npx eslint
   __tests__/userService.clean.test.js
(node:...) ESLintRCWarning: You are using an eslintrc configuration
   file...
$
```

A única saída foi o aviso sobre a versão do ESLint (que era esperado), provando que todos os "smells"de linting foram corrigidos com sucesso.

4.2 Validação da Suíte de Testes

A execução do `npm test` confirmou que todos os testes (tanto do arquivo antigo quanto do novo refatorado) passaram, provando que a refatoração não quebrou a lógica de negócio existente.

4.3 Link do Repositório

O projeto final, contendo o arquivo `userService.clean.test.js` refatorado, foi enviado para o repositório no GitHub:

`https://github.com/diascw/test-smelly`

—

5 Conclusão

Este trabalho ilustrou que 100% de cobertura de código (ou testes que "passam") não garantem a qualidade da suíte de testes. Testes "mal cheirosos"são frágeis, difíceis de manter e podem esconder bugs, como visto no caso do `try/catch`.

A adoção de ferramentas de análise estática como o ESLint é fundamental para criar um "guard rail"automatizado que impede a introdução de "smells"no código.

A refatoração, guiada pelo padrão Arrange-Act-Assert (AAA), transformou testes confusos e frágeis em uma suíte limpa, legível e robusta. Cada teste agora tem uma responsabilidade única, é linear e fácil de entender, o que aumenta a confiança no software e reduz drasticamente o custo de manutenção futura.