

PONTÍFICA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Engenharia de Software – Teste de Software

Ana Luiza Machado Alves (735648)

**REFATORAÇÃO DE TESTES E DETECÇÃO DE TEST SMELLS:
Uma Análise Estática (ESLint) para Detecção de Test Smells e Refatoração de Suítes de Testes
Problemáticas**

Belo Horizonte
2025

1. Introdução

Um “Test Smell” (mau cheiro em teste) é um sintoma no código de teste que indica um problema mais profundo, tornando-o frágil, obscuro, lento e, conseqüentemente, incapaz de detectar bugs. Neste trabalho, será utilizado ferramentas de análise estática (ESLint) para detectar Test Smells e refatorar uma suíte de testes problemática, transformando-a em um código limpo, legível e robusto.

Repositório do projeto: <https://github.com/analuizaalvesm/test-smelly>

2. Análise Manual

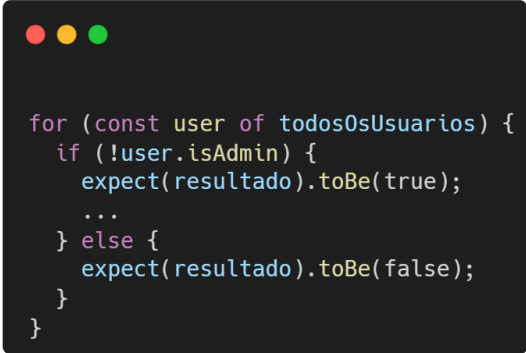
Durante a análise do arquivo original `userService.smelly.test.js`, foram identificados os seguintes Test Smells:

Smell 1 - Eager Test: `test('deve criar e buscar um usuário corretamente');`

O teste 'deve criar e buscar um usuário corretamente' realiza duas ações distintas (criação e busca) dentro do mesmo caso de teste. Isso viola o princípio de foco único e dificulta identificar onde a falha ocorre. A correção envolve dividir o teste em dois, seguindo o padrão AAA.

Smell 2 - Conditional Test Logic: `test('deve desativar usuários se eles não forem administradores');`


Este teste contém estruturas condicionais (if) e loop (for), o que o torna não determinístico. A execução dos asserts fica dependente de condições, o que pode mascarar falhas (alguns `expect()` nem são executados). Isso compromete a previsibilidade e clareza do teste. A correção deve pautar na remoção do for e do if e separar os casos em dois testes separados, um para usuário comum e outro para administrador.



```
for (const user of todosOsUsuarios) {  
  if (!user.isAdmin) {  
    expect(resultado).toBe(true);  
    ...  
  } else {  
    expect(resultado).toBe(false);  
  }  
}
```

Smell 3 - Fragile Test: `test('deve gerar um relatório de usuários formatado');`

O teste 'deve gerar um relatório de usuários formatado' depende da formatação exata da string do relatório, o que o torna frágil a pequenas mudanças. A correção consiste em validar apenas o conteúdo essencial (nomes e título) ao invés da formatação literal.



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

3. Análise dos Resultados da Detecção Automática com ESLint

A ferramenta ESLint com o plugin Jest foi configurada conforme o guia do trabalho. Na primeira execução, foram identificados diversos avisos relacionados a testes condicionais, nomes duplicados e blocos desativados:

```
Ana Luiza@DESKTOP-K0P3GEO MINGW64 ~/Desktop/test-smelly (main)
$ npx eslint

C:\Users\Ana Luiza\Desktop\test-smelly\src\userService.js
  1:16  error  'require' is not defined  no-undef
  72:1  error  'module' is not defined   no-undef

C:\Users\Ana Luiza\Desktop\test-smelly\test\userService.smelly.test.js
  1:25  error  'require' is not defined  no-undef
  9:1   error  'describe' is not defined no-undef
 13:3   error  'beforeEach' is not defined no-undef
 18:3   error  'test' is not defined     no-undef
 25:5   error  'expect' is not defined   no-undef
 29:5   error  'expect' is not defined   no-undef
 30:5   error  'expect' is not defined   no-undef
 33:3   error  'test' is not defined     no-undef
 44:9   error  Avoid calling `expect` conditionally` jest/no-conditional-expect
 44:9   error  'expect' is not defined   no-undef
 46:9   error  Avoid calling `expect` conditionally` jest/no-conditional-expect
 46:9   error  'expect' is not defined   no-undef
 49:9   error  Avoid calling `expect` conditionally` jest/no-conditional-expect
 49:9   error  'expect' is not defined   no-undef
 54:3   error  'test' is not defined     no-undef
 62:5   error  'expect' is not defined   no-undef
 63:5   error  'expect' is not defined   no-undef
 66:3   error  'test' is not defined     no-undef
 73:7   error  Avoid calling `expect` conditionally` jest/no-conditional-expect
 73:7   error  'expect' is not defined   no-undef
 77:3   warning Tests should not be skipped  jest/no-disabled-tests
 77:3   error  'test' is not defined     no-undef

X 24 problems (23 errors, 1 warning)
```

O ESLint reportou 24 problemas (23 erros e 1 aviso). Parte deles está relacionada à configuração do ambiente (variáveis globais do Node e Jest não reconhecidas) e o restante a problemas reais de qualidade nos testes. As regras do plugin eslint-plugin-jest destacaram condições e testes desativados.

A ferramenta identifica Test Smells por meio de análise estática, examinando a estrutura do código antes da execução. Ela percorre a AST para detectar padrões como expect() dentro de condicionais, testes desativados, títulos duplicados e erros de configuração. Essas validações permitem identificar rapidamente problemas estruturais sem executar o código, tornando o processo de revisão mais ágil e eficiente.

Na análise manual, foram identificados três Test Smells principais: Eager Test, Conditional Test Logic, Fragile Test. A comparação mostrou que o ESLint confirmou parcialmente as observações humanas.

A correspondência mais clara foi o Conditional Test Logic: o ESLint detectou o uso de expect() em estruturas condicionais, validando a análise manual. Também identificou testes desativados via test.skip(), algo não observado na revisão humana, mas relevante por indicar código potencialmente obsoleto.

Por outro lado, o ESLint não conseguiu identificar os tipos de smell Eager Test e Fragile Test, que exigem interpretação semântica e compreensão da intenção do teste, evidenciando questões que ainda dependem da análise humana. Assim, embora a ferramenta automatize a detecção de más práticas estruturais, ela complementa, mas não substitui, o julgamento do desenvolvedor.

4. Processo de Refatoração

O teste original que trata da desativação de usuários é um dos exemplos mais evidentes de Test Smells no arquivo `__tests__/userService.smelly.test.js`. Ele concentra múltiplas responsabilidades, utiliza estruturas condicionais e loops dentro do corpo do teste, tornando o comportamento confuso e frágil.

Antes da refatoração (`__tests__/userService.smelly.test.js`):

```
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];

  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);
    }
  }
});
```

O teste antigo utiliza estruturas condicionais (if e for) que tornam os asserts dependentes de situações específicas e misturam comportamentos diferentes em um único caso. Isso compromete a clareza e previsibilidade, dificulta identificar qual cenário falhou e reduz a manutenibilidade do código de teste.

Após a refatoração (`__tests__/userService.clean.test.js`):

```
test('DADO um usuário comum QUANDO desativar o usuário ENTÃO retorna true e status \'inativo\'', () => {
  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('DADO um usuário administrador QUANDO desativar o usuário ENTÃO retorna false', () => {
  const usuarioAdmin = userService.createUser(
    'Admin',
    'admin@teste.com',
    40,
    true
  )

  const resultado = userService.deactivateUser(usuarioAdmin.id)

  expect(resultado).toBe(false)
})
```

As decisões de refatoração adotadas foram motivadas pela necessidade de melhorar a clareza, a manutenibilidade e a confiabilidade dos testes, eliminando padrões considerados Test

Smells. A primeira ação foi a separação de responsabilidades, dividindo o teste original em dois casos independentes: um para usuários comuns e outro para administradores. Essa alteração removeu o Eager Test e permitiu identificar com precisão o cenário de falha, facilitando o diagnóstico.

Em seguida, ocorreu a remoção de lógica condicional dentro dos testes, eliminando estruturas como `for` e `if` que tornavam o comportamento imprevisível. Com isso, todos os `expect()` passaram a ser executados de forma determinística e linear, assegurando cobertura completa e previsível.

Foi aplicado o padrão AAA (Arrange, Act, Assert), que organiza cada teste em três etapas: preparação dos dados e do cenário (Arrange), execução do método `deactivateUser` (Act) e verificação dos resultados (Assert). Essa padronização melhora a legibilidade e facilita futuras manutenções.

Também buscou-se utilizar clareza semântica na nomeação dos testes. Os novos títulos seguem o padrão “Given When Then”, uma abordagem comumente utilizada em testes unitários para organizar o código do teste de forma clara e concisa.

Por fim, o ESLint não reportou mais nenhum erro ou aviso e todos os casos de teste foram executados com sucesso:

```
Ana Luiza@DESKTOP-K0P3GE0 MINGW64 ~/Desktop/test-smelly (main)
• $ npm test test/userService.clean.test.js

> test-smells-lab@1.0.0 test
> jest test/userService.clean.test.js

PASS test/userService.clean.test.js
  UserService - Testes Refatorados (Clean)
    ✓ DADO dados de usuário válidos QUANDO criar o usuário ENTÃO retorna id, nome correto e status 'ativo' (3 ms)
    ✓ DADO um usuário existente QUANDO buscar por ID ENTÃO retorna o usuário com os dados esperados (1 ms)
    ✓ DADO um usuário comum QUANDO desativar o usuário ENTÃO retorna true e status 'inativo'
    ✓ DADO um usuário administrador QUANDO desativar o usuário ENTÃO retorna false (1 ms)
    ✓ DADO vários usuários QUANDO gerar relatório ENTÃO o relatório contém nomes e o cabeçalho (1 ms)
    ✓ DADO dados de usuário menor de idade QUANDO tentar criar ENTÃO lança erro sobre maioridade (8 ms)

Test Suites: 1 passed, 1 total
Tests: 6 passed, 6 total
Snapshots: 0 total
Time: 0.305 s, estimated 1 s
Ran all test suites matching /test\\userService.clean.test.js/i.
```

5. Conclusão

A realização deste trabalho permitiu compreender como os Test Smells comprometem a clareza, a robustez e a eficácia das suítes de teste. A análise inicial revelou problemas estruturais, como condicionais em asserts, testes com múltiplas responsabilidades e dependência de detalhes de implementação, que, embora não impedissem a execução, reduziam a confiabilidade dos testes ao longo do tempo.

A refatoração aplicada, guiada pelo padrão Arrange, Act, Assert (AAA), promoveu uma estrutura mais limpa, organizada e fácil de manter, permitindo isolar comportamentos e garantir que cada caso de teste validasse apenas uma responsabilidade específica. Essa abordagem aumentou a legibilidade e facilitou a identificação de falhas reais.

Além disso, o uso do ESLint com o plugin Jest demonstrou o potencial das ferramentas de análise estática na automação da detecção de más práticas, reduzindo erros humanos e padronizando a qualidade do código de teste.

No entanto, observou-se que a revisão manual ainda é essencial para identificar smells sem correspondência direta nas regras automatizadas, como o Eager Test e o Fragile Test.

Por fim, uma suíte de testes limpa, sem smells e validada tanto manual quanto automaticamente reforça a importância da combinação entre boas práticas de escrita de testes e ferramentas de verificação automatizada para garantir a qualidade, sustentabilidade e manutenibilidade de projetos de software a longo prazo.