Pontifícia Universidade Católica de Minas Gerais

Engenharia de Software
Teste de Software

Refatoração de Testes e Detecção de Test Smells

Aluno: Ana Flávia de Carvalho Santos
Professor: Prof. Cleiton Tavares

Belo Horizonte
2 de novembro de 2025

1 Análise de Smells

1.1 Conditional Test Logic

O smell foi identificado no teste de desativação de usuários, que contém um loop for iterando sobre uma lista de usuários e condicionais if/else para executar diferentes asserções dependendo do tipo de usuário. Isso é considerado um mau cheiro porque os testes unitários não devem ter lógica condicional, a presença dessas estruturas aumenta a complexidade ciclomática do teste, tornando-o difícil de ler, entender e manter. Além disso, o teste viola o princípio de responsabilidade única ao verificar dois comportamentos distintos (desativação de usuário comum e falha na desativação de administrador) em um único caso de teste, caracterizando também um Eager Test.

Os riscos associados a este smell incluem obscuridade na compreensão do que está sendo testado, dificuldade de depuração quando o teste falha, pois não fica claro qual cenário específico causou a falha, possibilidade de cobertura incompleta se uma das condições nunca for executada, alto custo de manutenção ao precisar modificar a estrutura condicional, e uma falsa sensação de segurança ao aparentar cobrir múltiplos cenários em um único teste quando a cobertura real pode ser ruim.

1.2 Fragile Test

O smell foi identificado no teste de geração de relatório que verifica a saída de um relatório comparando-a com strings formatadas exatas, incluindo detalhes de implementação como espaços, vírgulas, quebras de linha e ordem específica dos campos. Este padrão é considerado um mau cheiro porque o teste está fortemente acoplado aos detalhes de implementação da formatação, e não ao comportamento funcional do sistema. Qualquer alteração mínima na formatação, como adicionar um espaço, reordenar campos, alterar a pontuação ou incluir novos dados, causará a falha do teste, mesmo que a funcionalidade de geração correta dos dados do relatório esteja funcionando como deveria.

Os riscos deste tipo de teste incluem alto custo de manutenção, pois mudanças cosméticas ou de formatação exigem atualizações constantes, falsos negativos que reportam falhas para mudanças que não representam bugs reais e desconfiança na suíte de testes por parte dos desenvolvedores quando ela quebra frequentemente por motivos que não representam risco. Em suma, testes devem verificar o que o sistema faz, não como ele faz.

1.3 Silent Catcher

Este smell foi identificado no teste de não permitir usuários menores de idade. O teste utiliza um bloco try-catch para verificar se uma exceção é lançada ao tentar criar um usuário menor de 18 anos, executando uma asserção apenas dentro do bloco catch. Esse é considerado um mau cheiro crítico porque, se a função createUser não lançar a exceção esperada, por exemplo, devido à remoção acidental da validação de idade ou alteração no comportamento do método, o bloco try será executado sem erros, o bloco catch não será acionado, e o teste passará silenciosamente sem executar nenhuma asserção. Isso resulta em um falso positivo no qual o teste indica sucesso apesar de um bug crítico existir no código de produção.

Os riscos associados são críticos: ocultação de bugs críticos, onde o teste pode passar mesmo quando a validação de segurança é completamente removida ou está quebrada,

falsos positivos, potencial violação de requisitos de negócio ou regulatórios se a validação de idade for obrigatória por lei e falta de garantia de que alguma asserção será executada durante o teste

2 Processo de Refatoração

Para demonstrar o processo de refatoração aplicado, foi selecionado o teste mais crítico identificado na análise: o teste de validação de idade que apresentava o smell Silent Catcher. Este teste foi escolhido por representar o maior risco à segurança da suíte, uma vez que poderia passar silenciosamente mesmo na presença de bugs graves relacionados a regras de negócio obrigatórias.

```
  test('deve falhar ao criar usuário menor de idade', () => {
    // Este teste não falha se a exceção NÃO for lançada.
    // Ele só passa se o `catch` for executado. Se a lógica de validação
    // for removida, o teste passa silenciosamente, escondendo um bug.
    try {
      userService.createUser('Menor', 'menor@email.com', 17);
    } catch (e) {
      expect(e.message).toBe('O usuário deve ser maior de idade.');
    }
  });

  test.skip('deve retornar uma lista vazia quando não há usuários', () => {
    // TODO: Implementar este teste depois.
  });
});
```

Figura 1: Teste antes da refatoração.

O teste original utilizava um bloco try-catch para capturar a exceção esperada quando um usuário menor de idade é criado. O problema fundamental dessa abordagem é que, se a exceção não for lançada, o bloco catch simplesmente não será executado e o teste terminará sem executar nenhuma asserção, passando silenciosamente e gerando um falso positivo perigoso.

```
test('deve falhar ao criar usuário menor de idade', () => {
// Arrange
const nome = 'Menor';
const email = 'menor@email.com';
const idade = 17;

// Act e Assert
expect(() => userService.createUser(nome, email, idade)).toThrow('O usuário deve ser maior de idade.');
});
});
```

Figura 2: Teste depois da refatoração.

A refatoração aplicada substituiu completamente o bloco try-catch pelo uso do matcher toThrow() do Jest. Esta abordagem é a forma idiomática e recomendada pelo framework para testar exceções, garantindo que o teste falhará explicitamente se a exceção não for lançada. A nova implementação utiliza uma função arrow como argumento para o expect(), permitindo que o Jest execute a função em um contexto controlado e verifique se a exceção esperada foi de fato lançada. Além disso, o matcher verifica não apenas que uma exceção foi lançada, mas também que a mensagem da exceção corresponde exatamente ao texto esperado, garantindo que a exceção correta está sendo lançada pelo motivo correto.

Esta refatoração corrigiu o smell identificado ao tornar o teste robusto e explícito em sua intenção. Agora, se a validação de idade for removida ou modificada de forma que a exceção não seja mais lançada, o teste falhará imediatamente com uma mensagem clara indicando que a exceção esperada não ocorreu. O padrão AAA (Arrange-Act-Assert) também ficou mais claro, com a fase de Arrange preparando os dados de entrada, e as fases Act e Assert combinadas na chamada do matcher.

3 Relatório da Ferramenta

Após a configuração do ESLint com o plugin eslint-plugin-jest, a ferramenta foi executada sobre o arquivo userService.smelly.test.js original, gerando o relatório apresentado na Figura 3. A análise estática identificou automaticamente 6 problemas na suíte de testes, 4 erros e 2 avisos. Os erros corresponderam a violações da regra jest/no-conditional-expect nas linhas 44, 46, 49 e 73, detectando as chamadas de expect dentro de estruturas condicionais e loops, confirmando o smell de Lógica Condicional identificado manualmente. Os avisos corresponderam às regras jest/no-disabled-tests e jest/expect-expect na linha 77, sinalizando a presença de um teste desabilitado com test.skip que não possuía asserções implementadas.

A ferramenta de análise estática automatizou o processo de detecção de Test Smells, tornando-o mais rápido, consistente e objetivo. Enquanto a análise manual requer conhecimento profundo de boas práticas e está sujeita a falhas humanas, o ESLint aplica regras predefinidas de forma sistemática e instantânea, identificando problemas que poderiam passar despercebidos em revisões de código. Importante destacar que, após a refatoração completa e execução do ESLint sobre o arquivo userService.clean.test.js, nenhum erro ou aviso foi reportado, confirmando a eliminação bem-sucedida de todos os smells identificados.

```
C:\Users\dtiDigital\Desktop\Faculdade\test-smelly\test\userService.smelly.test.js
  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
  77:3  warning  Test has no assertions                 jest/expect-expect

✖ 6 problems (4 errors, 2 warnings)
```

Figura 3: Resultado da execução do ESLint no arquivo original, mostrando 4 erros e 2 avisos

4 Conclusão

Este trabalho demonstrou na prática como test smells comprometem a eficácia de uma suíte de testes, mesmo quando ela apresenta alta cobertura de código. A presença de lógica condicional, testes frágeis e captura silenciosa de exceções não apenas dificulta a manutenção e compreensão dos testes, mas também cria uma falsa sensação de segurança, permitindo que bugs críticos passem despercebidos. A refatoração aplicada, seguindo o padrão AAA (Arrange-Act-Assert) e o princípio de responsabilidade única, transformou testes com problemas em casos de teste claros, focados e confiáveis. Cada teste refatorado passou a verificar um único comportamento de forma explícita, eliminando ambiguidades e facilitando tanto a identificação de falhas quanto futuras manutenções.

A integração de ferramentas de análise estática, como o ESLint com plugins especializados garante a qualidade contínua do código de teste, automatizando a detecção de smells e más práticas, estabelecendo um padrão consistente e objetivo que complementa a revisão humana. A adoção dessas ferramentas melhora a confiabilidade e manutenibilidade da suíte de testes e também contribui para a sustentabilidade a longo prazo do projeto de software, permitindo refatorações seguras, desenvolvimento ágil e entrega contínua com maior confiança na detecção de regressões.