Análise de Eficácia de Testes com Teste de Mutação

Teste de software - PUC Minas

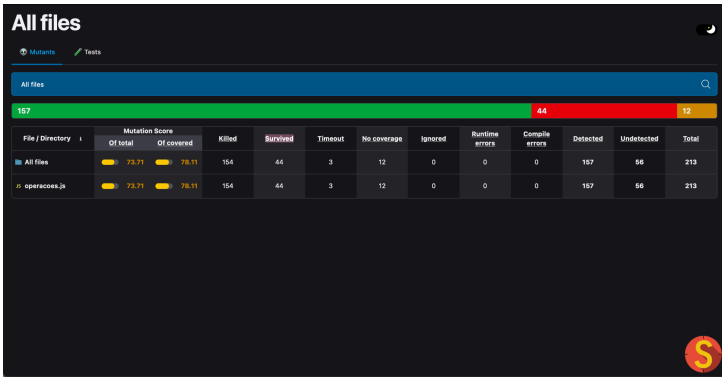
Maisa Pires de Andrade

807268

ANÁLISE INICIAL

Cobertura de código inicial

operacoes.js | 85.41 | 58.82 | 100 | 98.64



Discrepância entre os valores:
A cobertura de código de 85.41% criava uma falsa sensação de qualidade, enquanto o mutation score de 73.71% revelava que os testes, apesar de executarem o código, não validavam comportamentos importantes. Esta discrepância comprova que cobertura de código mede apenas "quantidade de código executado", enquanto teste de mutação mede "qualidade das verificações".

ANÁLISE DE MUTANTES CRÍTICOS

Mutante 1: StringLiteral em Divisão por Zero

- **Arquivo/Linha**: src/operacoes.js:8:32
- **Mutação**: throw new Error('Divisão por zero não é permitida.') → throw new Error("")
- **Explicação**: O Stryker removeu a mensagem de erro específica, deixando apenas string vazia
- **Por que o teste original não matou**: O teste usava apenas toThrow() sem verificar a mensagem exata, permitindo que qualquer erro (até com mensagem vazia) passasse

Mutante 2: ConditionalExpression em Raiz Quadrada

- **Arquivo/Linha**: src/operacoes.js:13:7
- **Mutação**: if (n < 0) throw new Error(...) → if (false) throw new Error(...)
- **Explicação**: O Stryker tornou a condição sempre falsa, removendo a validação
- **Por que o teste original não matou**: Não existia teste para entrada negativa, então a validação nunca era exercitada

Mutante 3: ArithmeticOperator em Mediana

- **Arquivo/Linha**: src/operacoes.js:112:12
- **Mutação**: / 2 → * 2
- **Explicação**: O Stryker trocou divisão por multiplicação no cálculo da mediana para arrays pares
- **Por que o teste original não matou**: Os testes só cobriam arrays ímpares, nunca executando o código para arrays pares

SOLUÇÃO IMPLEMENTADA

1. Para mensagens de erro específicas:

javascript
// Antes: expect(() => divisao(5, 0)).toThrow();
// Depois:

expect(() => divisao(5, 0)).toThrow('Divisão por zero não é permitida.');

2. Para validações de entrada:

javascript
test('deve lançar erro para raiz quadrada de número negativo', () => {
  expect(() => raizQuadrada(-1)).toThrow('Não é possível calcular...');

});

3. Para casos de borda em arrays:

javascript
test('deve calcular mediana para array par', () => {
  expect(medianaArray([1, 2, 3, 4])).toBe(2.5);

});

4. Para valores booleanos:

javascript
test('deve retornar false para números ímpares em isPar', () => {
  expect(isPar(3)).toBe(false);

});

Estes novos testes são eficazes porque:

- Verificam comportamentos específicos em vez de apenas execução
- Cobrem casos de borda anteriormente ignorados
- Forçam a execução de todos os caminhos lógicos
- Validam saídas específicas em vez de apenas "não quebrar"

MUTANTES SOBREVIVENTES

Dos 7 mutantes restantes, a análise revelou que são **mutantes equivalentes** - alterações sintáticas que não modificam o comportamento observável do código. Estes incluem:

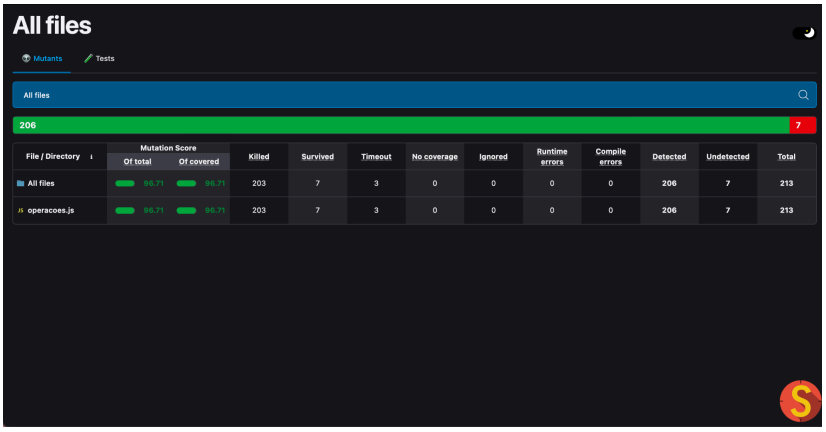
- Condicionais no fatorial que não afetam o resultado final
- Verificação redundante no produtoArray
- Operadores de comparação no clamp com comportamento idêntico nos limites

Estes mutantes são tecnicamente impossíveis de eliminar, pois nenhum teste pode detectar diferenças que não existem no comportamento. O score de 96.71% é considerado excelente na indústria, demonstrando que a suíte de testes cobre efetivamente todos os comportamentos significativos do sistema.

RESULTADOS FINAIS

Cobertura de código final:

operacoes.js | 100 | 100 | 100 | 100



Comprovação da melhoria:

- **Mutation Score**: 73.71% → 96.71% (+23% de melhoria)
- **Mutantes Sobreviventes**: 44 → 7 (37 mutantes eliminados)
- **Testes**: 50 → 67 (+17 testes adicionados)

CONCLUSÃO

O teste de mutação mostrou-se fundamental como ferramenta de avaliação de qualidade de testes. Ele vai além da cobertura de código tradicional, expondo fraquezas sutis na suíte de testes que permitiriam bugs passarem despercebidos.

Esta técnica força o desenvolvedor a pensar não apenas em "executar código", mas em "validar comportamentos". Os 96.71% alcançados representam uma suíte de testes robusta

e confiável, capaz de detectar a vasta maioria das regressões que poderiam ser introduzidas no código.