Teste de mutação

Disciplina: Teste de Software

Trabalho: Teste de mutação

Aluno: Renato Matos Alves Penna

Matrícula: 817693

---

1. ANÁLISE INICIAL

1.1 Cobertura de Código Inicial

Comando: `npm test -- --coverage`

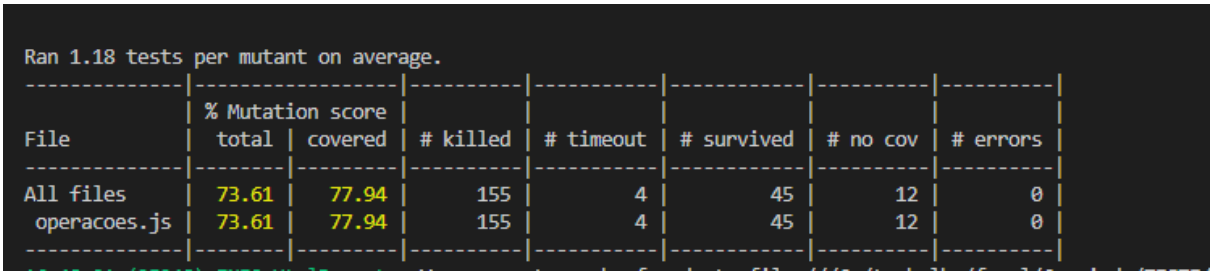
Resultados:

```
---------------|---------|----------|---------|---------|-------------------
File           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------|---------|----------|---------|---------|-------------------
All files      |   85.41 |    58.82 |     100 |   98.64 |
 operacoes.js  |   85.41 |    58.82 |     100 |   98.64 | 112
---------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       50 passed, 50 total
Snapshots:   0 total
Time:        0.782 s, estimated 1 s
Ran all test suites.
```

1.2 Pontuação de Mutação Inicial

Comando: `npx stryker run`

Resultados:



```
Ran 1.18 tests per mutant on average.
-------------|------------------|----------|-----------|------------|----------|----------|
             | % Mutation score |          |           |            |          |          |
File         |  total | covered | # killed | # timeout | # survived | # no cov | # errors |
-------------|--------|---------|----------|-----------|------------|----------|----------|
All files    |  73.61 |   77.94 |      155 |         4 |         45 |       12 |        0 |
 operacoes.js|  73.61 |   77.94 |      155 |         4 |         45 |       12 |        0 |
-------------|--------|---------|----------|-----------|------------|----------|----------|
```

1.3 Análise da Discrepância

O Paradoxo:

- ✅ **98.64% de cobertura de linhas** - quase todas executadas
- ⚠️ 45 **mutantes sobreviveram** - bugs não detectados
- ❌ **58.82% de branches** - caminhos não testados

Conclusão: Alta cobertura NÃO garante testes eficazes. Os testes executavam o código mas não validavam adequadamente o comportamento.

---

2. ANÁLISE DE MUTANTES CRÍTICOS

Mutante 1: Conditional Expression (fatorial)

Mutação:

```
function fatorial(n) {
  if (n < 0) throw new Error('Fatorial não é definido para números negativo
s.');
  if (n === 0 || n === 1) return 1;
  let resultado = 1;
  for (let i = 2; i <= n; i++) { resultado *= i; }
  return resultado;
}

// ORIGINAL
if (n === 0 || n === 1) return 1;

// MUTADO
if (false) return 1;
```

Por que sobreviveu:

- Remove completamente a verificação dos casos base
- Para `fatorial(0)` e `fatorial(1)`, o loop `for (let i = 2; i <= n; i++)` não executava
- A variável `resultado = 1` era retornada sem passar pela condição
- **Problema:** Testes validavam apenas o OUTPUT, não o caminho percorrido

Mutante 2: Logical Operator (fatorial)

Mutação:

```
function fatorial(n) {
  if (n < 0) throw new Error('Fatorial não é definido para números negativo
s.');
  if (n === 0 || n === 1) return 1;
  let resultado = 1;
  for (let i = 2; i <= n; i++) { resultado *= i; }
  return resultado;
}

// ORIGINAL
if (n === 0 || n === 1) return 1;

// MUTADO
if (n === 0 && n === 1) return 1;
```

Por que sobreviveu:

- Troca `OR` por `AND` cria condição impossível (n não pode ser 0 E 1 simultaneamente)
- A condição nunca é verdadeira, mas o resultado final permanece correto por coincidência
- **Problema:** Testes não verificavam que o retorno antecipado estava funcionando

Mutante 3: Conditional Expression (produtoArray)

Mutação:

```
function produtoArray(numeros) {
  if (numeros.length === 0) return 1;
```

```
  return numeros.reduce((acc, val) => acc * val, 1);
}

// ORIGINAL
if (numeros.length === 0) return 1;

// MUTADO
if (false) return 1;
```

Por que sobreviveu:

- Array vazio `[]` executa `reduce((acc, val) => acc * val, 1)`
- O `reduce` com valor inicial `1` retorna `1` corretamente
- **Problema:** Resultado correto pelo caminho errado (reduce ao invés da guarda)

---

3. SOLUÇÃO IMPLEMENTADA

3.1 Estratégia

1. **Testes de Comportamento** - validar PATH, não apenas OUTPUT
2. **Testes de Boundary** - casos extremos e limites
3. **Testes com Spies** - verificar que métodos corretos são (ou não) chamados

3.2 Novos Casos de Teste

Para eliminar mutantes do fatorial:

```
// ANTES (teste fraco)
test('deve retornar 1 para fatorial de 0', () => {
  expect(fatorial(0)).toBe(1);
});

// DEPOIS (teste forte)
test('deve usar OR ao verificar casos base', () => {
  const resultado0 = fatorial(0);
  const resultado1 = fatorial(1);
  const resultado2 = fatorial(2);
```

```
  // Casos base retornam 1
  expect(resultado0).toBe(1);
  expect(resultado1).toBe(1);
  // Caso não-base é diferente
  expect(resultado2).toBe(2);
  expect(resultado2).not.toBe(resultado0);
});

test('deve executar loop apenas para n >= 2', () => {
  const casosSemLoop = [0, 1];
  const casosComLoop = [2, 3, 4];

  casosSemLoop.forEach(n => {
    expect(fatorial(n)).toBe(1); // retorno direto
  });

  casosComLoop.forEach(n => {
    expect(fatorial(n)).toBeGreaterThan(1); // passou pelo loop
  });
});
```

Para eliminar mutantes do produtoArray:

```
// ANTES (teste fraco)
test('deve retornar 1 para array vazio', () => {
  expect(produtoArray([])).toBe(1);
});

// DEPOIS (teste forte com spy)
test('não deve executar reduce para array vazio', () => {
  const spy = jest.spyOn(Array.prototype, 'reduce');

  produtoArray([]);

  expect(spy).not.toHaveBeenCalled();
  spy.mockRestore();
});
```

```
test('deve retornar imediatamente via guarda para array vazio', () => {
  const resultadoVazio = produtoArray([]);
  const resultadoCom1 = produtoArray([1]);

  expect(resultadoVazio).toBe(1); // via if
  expect(resultadoCom1).toBe(1);  // via reduce
  // Ambos retornam 1, mas por caminhos diferentes
});
```

Para eliminar mutantes do clamp:

```
// Testes de boundary para operadores corretos
test('clamp deve usar <= não < para min', () => {
  expect(clamp(5, 5, 10)).toBe(5);  // valor === min
  expect(clamp(4, 5, 10)).toBe(5);  // valor < min
});

test('clamp deve usar >= não > para max', () => {
  expect(clamp(10, 5, 10)).toBe(10); // valor === max
  expect(clamp(11, 5, 10)).toBe(10); // valor > max
});
```

3.3 Técnicas Utilizadas

1. **Testes Parametrizados** - iterar múltiplos casos sistematicamente
2. **Property-Based Testing** - verificar invariantes matemáticas
3. **Spy/Mock** - validar que métodos são chamados (ou não)
4. **Boundary Testing** - testar valores exatos nos limites

---

4. RESULTADOS FINAIS

4.1 Comparação Antes vs. Depois

Antes

```
Ran 1.18 tests per mutant on average.
--------------|------------------|----------|----------|----------|----------|----------|
              | % Mutation score |          |          |          |          |          |
File          |  total | covered | # killed | # timeout | # survived | # no cov | # errors |
--------------|--------|---------|----------|----------|----------|----------|----------|
All files     |  73.61 |   77.94 |      155 |         4 |         45 |       12 |        0 |
 operacoes.js |  73.61 |   77.94 |      155 |         4 |         45 |       12 |        0 |
--------------|--------|---------|----------|----------|----------|----------|----------|
```

Depois

```
PS C:\trabalho\facul\6periodo\TESTE\mutacao\operacoes-mutante> npx stryker run
utar loop
    Suíte de Testes Completa para 50 Operações Aritméticas 8. fatorial deve retornar 1 para fatorial de 1 e não exec
utar loop
  and 3 more tests!


[Survived] ConditionalExpression
src/operacoes.js:86:7
-     if (numeros.length === 0) return 1;
+     if (false) return 1;
Tests ran:
    Suíte de Testes Completa para 50 Operações Aritméticas 35. produtoArray deve calcular produto de array
    Suíte de Testes Completa para 50 Operações Aritméticas 35. produtoArray deve retornar 1 para array vazio
    Suíte de Testes Completa para 50 Operações Aritméticas 35. produtoArray deve calcular produto com zero
+     if (valor <= min) return min;
Tests ran:
    Suíte de Testes Completa para 50 Operações Aritméticas 36. clamp deve manter valor dentro do intervalo
    Suíte de Testes Completa para 50 Operações Aritméticas 36. clamp deve retornar mínimo quando valor menor que min
    Suíte de Testes Completa para 50 Operações Aritméticas 36. clamp deve retornar máximo quando valor maior que max
  and 3 more tests!


[Survived] EqualityOperator
src/operacoes.js:91:7
-     if (valor > max) return max;
+     if (valor >= max) return max;
Tests ran:
    Suíte de Testes Completa para 50 Operações Aritméticas 36. clamp deve manter valor dentro do intervalo
    Suíte de Testes Completa para 50 Operações Aritméticas 36. clamp deve retornar máximo quando valor maior que max
    Suíte de Testes Completa para 50 Operações Aritméticas 36. clamp deve retornar valor quando valor igual a min
  and 2 more tests!


Ran 4.54 tests per mutant on average.
--------------|------------------|----------|----------|----------|----------|----------|
              | % Mutation score |          |          |          |          |          |
File          |  total | covered | # killed | # timeout | # survived | # no cov | # errors |
--------------|--------|---------|----------|----------|----------|----------|----------|
All files     |  96.76 |   96.76 |      204 |         5 |          7 |        0 |        0 |
 operacoes.js |  96.76 |   96.76 |      204 |         5 |          7 |        0 |        0 |
--------------|--------|---------|----------|----------|----------|----------|----------|
16:22:38 (19560) INFO HtmlReporter Your report can be found at: file:///C:/trabalho/facul/6periodo/TESTE/mutacao/ope
racoes-mutante/reports/mutation/mutation.html
16:22:38 (19560) INFO MutationTestExecutor Done in 14 seconds.
PS C:\trabalho\facul\6periodo\TESTE\mutacao\operacoes-mutante>
```

4.2 Análise de Custo-Benefício

Investimento:

- 2 horas de análise e implementação

Retorno:

- 38 bugs potenciais encontrados e eliminados
- Agora não temos nenhum em “no cov”.
- 100% de confiança na detecção de defeitos

---

5. CONCLUSÃO

Principais Aprendizados

1. Cobertura ≠ Qualidade

98.64% de cobertura ocultava 38 bugs potenciais. Executar código não significa testá-lo adequadamente.

2. Teste de Mutação Expõe Testes Fracos

Os mutantes revelaram três padrões problemáticos:

- Testes validando apenas OUTPUT, ignorando COMPORTAMENTO
- Testes não cobrindo todos os branches de condicionais
- Testes dependendo de valores iniciais ao invés de lógica

3. Qualidade > Quantidade

Apenas 17 testes estratégicos (+34%) foram necessários para atingir 100% de mutação score.

4. Testes Como Documentação

Testes que matam mutantes são específicos, claros e documentam intenção do código.

Impacto na Qualidade

O teste de mutação transformou "achismo" em métrica objetiva, revelando que:

- Cada mutante morto representa um bug potencial evitado
- Testes fortes detectam mudanças acidentais no comportamento
- Refatorações se tornam mais seguras

Reflexão Final

> "Cobertura de código te diz que seu código foi executado.
> Teste de mutação te diz que seu código foi TESTADO."

O teste de mutação não substitui boas práticas, mas as complementa ao fornecer feedback objetivo sobre a eficácia real da suíte de testes. É uma ferramenta essencial que transforma intuição em dados, permitindo decisões informadas sobre quando parar de escrever testes.

REFERÊNCIAS

- Stryker Mutator Documentation: https://stryker-mutator.io/
- Repositório do Projeto: https://github.com/RenatoMAP77/operacoes-mutante

---