RELATÓRIO DE TESTE DE MUTAÇÃO

Análise de Eficácia de Testes com StrykerJS

Disciplina: Teste de Software
Trabalho: Teste de Mutação
Aluno: Pedro Franco

---

1. ANÁLISE INICIAL

1.1 Cobertura de Código Inicial

A suíte de testes original apresentou uma **cobertura de código de aproximadamente 95-100%**, indicando que praticamente todas as linhas de código foram executadas durante os testes. À primeira vista, este resultado sugere uma suíte de testes robusta e confiável.

1.2 Pontuação de Mutação Inicial

Contudo, ao executar a análise de teste de mutação com o StrykerJS, obtivemos uma **pontuação de mutação inicial de apenas 73%**, revelando que aproximadamente 27% dos mutantes gerados conseguiram sobreviver à suíte de testes.

1.3 Discrepância entre Cobertura e Eficácia

Esta discrepância significativa entre a alta cobertura de código e a baixa pontuação de mutação evidencia uma limitação crítica da métrica de cobertura: **executar uma linha de código não garante que o comportamento dessa linha está sendo adequadamente verificado**.

A cobertura mede apenas se o código foi executado, mas não avalia:

- Se as **asserções** estão verificando os valores corretos
- Se **casos de borda** estão sendo testados
- Se **condições alternativas** estão sendo validadas
- Se **erros** estão sendo apropriadamente capturados

Esta constatação confirma que a cobertura de código, embora útil, é uma métrica insuficiente para avaliar a qualidade real de uma suíte de testes.

---

2. ANÁLISE DE MUTANTES CRÍTICOS

2.1 Mutante 1: produtoArray - Condição de Array Vazio

Localização: Função `produtoArray()`, linha da verificação de array vazio

Mutação Aplicada:

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

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

Por que o mutante sobreviveu: O teste original apenas verificava o caso de array com múltiplos elementos:

```
test('deve calcular o produto de um array', () => {
  expect(produtoArray([2, 3, 4])).toBe(24);
});
```

Este teste **nunca passou um array vazio** para a função. Portanto, quando o Stryker substituiu a condição `numeros.length === 0` por `false`, o teste continuou passando, pois:

- A linha mutada nunca foi alcançada durante a execução do teste
- O comportamento para arrays não-vazios permaneceu inalterado
- Não havia asserção verificando o retorno para array vazio

Impacto: Este mutante sobrevivente indica que o comportamento da função quando recebe um **array vazio não está sendo validado**, deixando uma vulnerabilidade significativa no código.

2.2 Mutante 2: isDivisivel - Operador de Igualdade

Localização: Função `isDivisivel()`, linha do retorno

Mutação Aplicada:

```
// ORIGINAL
function isDivisivel(dividendo, divisor) {
  return dividendo % divisor === 0;
}
```

```
// MUTADO
function isDivisivel(dividendo, divisor) {
  return true;
}
```

Por que o mutante sobreviveu: O teste original apenas verificava o caso positivo (quando é divisível):

```
test('deve verificar se um número é divisível por outro', () => {
  expect(isDivisivel(10, 2)).toBe(true);
});
```

Este teste possui uma falha grave: ele **apenas verifica casos onde o resultado é `true`**. Quando o Stryker substituiu toda a lógica por `return true`, o teste continuou passando porque:

- O teste esperava `true` e recebeu `true`
- Não havia teste para verificar o caso negativo (quando não é divisível)
- A lógica do operador `===` nunca foi realmente validada

Impacto: Este é um exemplo clássico de **teste tautológico**, onde o teste sempre passa independente da implementação estar correta ou não.

---

3. SOLUÇÃO IMPLEMENTADA

3.1 Estratégia Geral

Para eliminar os mutantes sobreviventes, implementei uma estratégia abrangente baseada em três pilares:

1. **Testes de Casos de Borda** (Boundary Value Testing)
2. **Testes de Casos Negativos** (Negative Testing)
3. **Validação de Exceções** (Exception Testing)

3.2 Solução para Mutante 1 (produtoArray)

Novos testes adicionados:

```
test('35. produtoArray - array vazio', () => {
  expect(produtoArray([])).toBe(1);
});

test('35. produtoArray - com zero', () => {
```

```
  expect(produtoArray([5, 0, 3])).toBe(0);
});

test('35. produtoArray - um elemento', () => {
  expect(produtoArray([7])).toBe(7);
});
```

Justificativa:

- O teste com **array vazio** garante que a condição `numeros.length === 0` seja executada e validada
- O teste com **zero** verifica o comportamento matemático correto (qualquer produto com zero resulta em zero)
- O teste com **um elemento** valida o caso base do reduce
- Estes testes cobrem todos os caminhos lógicos possíveis da função

Eficácia: Com estes testes, o mutante que substitui a condição por `false` será detectado, pois o teste espera retorno `1` para array vazio, mas a função executaria o reduce e retornaria outro valor.

3.3 Solução para Mutante 2 (isDivisivel)

Novos testes adicionados:

```
test('37. isDivisivel - divisível', () => {
  expect(isDivisivel(10, 2)).toBe(true);
});

test('37. isDivisivel - não divisível', () => {
  expect(isDivisivel(10, 3)).toBe(false);
});

test('37. isDivisivel - por 1', () => {
  expect(isDivisivel(7, 1)).toBe(true);
});

test('37. isDivisivel - zero por número', () => {
  expect(isDivisivel(0, 5)).toBe(true);
});
```

Justificativa:

- O teste **"não divisível"** é crucial - ele verifica que a função retorna `false` quando deveria

- Este teste impede que mutantes que substituem a lógica por `return true` ou `return false` sobrevivam
- Os casos especiais (divisão por 1, zero) garantem robustez adicional
- A cobertura bidirecional (true e false) é essencial para funções booleanas

Eficácia: O mutante que retorna sempre `true` falhará no teste `isDivisivel(10, 3)`, que espera `false`.

---

4. RESULTADOS FINAIS

4.1 Pontuação de Mutação Final

Após a implementação dos novos testes, a análise com StrykerJS apresentou:

Pontuação de Mutação: 95%

5. CONCLUSÃO

5.1 Importância do Teste de Mutação

Este trabalho demonstrou de forma prática e tangível que o **Teste de Mutação é uma ferramenta indispensável para avaliar a qualidade real de uma suíte de testes**. Enquanto métricas tradicionais como cobertura de código fornecem uma falsa sensação de segurança, o teste de mutação expõe as fragilidades reais dos testes.