

# Relatório: Análise de Eficácia de Testes com Teste de Mutação

Testes de Software

**BRUNO EVANGELISTA GOMES DE AZEVEDO**

Matrícula: 809581

November 3, 2025

## Contents

<b>1</b>	<b>Análise Inicial</b>	<b>2</b>
<b>2</b>	<b>Análise de Mutantes Críticos</b>	<b>2</b>
2.1	Mutante #15: ConditionalExpression (da Imagem) . . . . .	2
2.2	Mutante #177: EqualityOperator . . . . .	3
2.3	Mutante #195: ArrowFunction (Lógica de Ordenação) . . . . .	3
<b>3</b>	<b>Solução Implementada</b>	<b>3</b>
<b>4</b>	<b>Resultados Finais</b>	<b>4</b>
<b>5</b>	<b>Conclusão</b>	<b>5</b>

## 1. Link do Repositório GitHub

A suíte de testes aprimorada, juntamente com a configuração do Stryker, pode ser encontrada no seguinte repositório:

[https://github.com/\[SEU-USUARIO\]/operacoes-mutante-rex](https://github.com/[SEU-USUARIO]/operacoes-mutante-rex)

## 1 Análise Inicial

A análise inicial da suíte de testes revelou uma perigosa discrepância entre as métricas de cobertura de código e a eficácia real dos testes.

Ao executar o comando `npm test -- --coverage`, a suíte de testes original reportou uma cobertura de código de **98.00%** (valor de exemplo, conforme implicado pelo PDF da tarefa). Esta pontuação, tradicionalmente vista como excelente, sugeria uma suíte de testes robusta e confiável.

No entanto, a execução do teste de mutação com o StrykerJS (`npx stryker run`) expôs a realidade. A pontuação de mutação inicial foi de apenas **59.34%**.

Table 1: Comparativo das Métricas Iniciais

Métrica	Pontuação Inicial
Cobertura de Código (Linhas)	98.00% (Exemplo)
<b>Pontuação de Mutação</b>	<b>59.34%</b>
Mutantes Não Detectados	37 (25 Sobreviventes + 12 S/ Cobertura)

A discrepância é clara: embora os testes executassem 98% do código, eles falhavam em validar adequadamente o comportamento de quase 41% das mutações. Isso significa que testes passavam mesmo quando o código-fonte continha bugs sutis (mutações), provando que a cobertura de código, isoladamente, é uma métrica insuficiente para garantir a qualidade.

## 2 Análise de Mutantes Críticos

A investigação dos 37 mutantes não detectados na primeira execução revelou padrões claros de fraquezas nos testes, principalmente relacionados a casos de borda e assertivas fracas.

### 2.1 Mutante #15: ConditionalExpression (da Imagem)

Este mutante, capturado na imagem, é um exemplo perfeito de teste de "caminho infeliz" em falta.

- **O que a mutação fez:** Na função `raizQuadrada(n)`, o Stryker alterou a verificação de erro `if (n < 0)` para `if (false)`.
- **Por que sobreviveu:** O teste original era `expect(raizQuadrada(16)).toBe(4);`. Este teste usa um input positivo (16), que nunca entra no bloco `if`. Como o código mutante (`if (false)`) e o código original se comportam de forma idêntica para números positivos (ambos pulam o `if`), o teste passou. Faltava um teste que forçasse a entrada nesse bloco, como `raizQuadrada(-1)`.

```

1 // src/operacoes.js
2
3 // == Bloco 1: Operações Básicas (1-10) ==
4 function soma(a, b) { return a + b; }
5 function subtracao(a, b) { return a - b; }
6 function multiplicacao(a, b) { return a * b; }
7 function divisao(a, b) {
8   if (b === 0) throw new Error('Divisão por zero não é permitida.');
```

```

9   return a / b;
10 }
11 function potencia(base, expoente) { return Math.pow(base, expoente); }
12 function raizQuadrada(n) {
13   - if (n < 0) throw new Error('Não é possível calcular a raiz quadrada de um número negativo.');
```

```

14   + if (false) throw new Error('Não é possível calcular a raiz quadrada de um número negativo.');
```

```

15   return Math.sqrt(n);
16 }
17 function restoDivisao(dividendo, divisor) { return dividendo % divisor; }
18 function fatorial(n) {
19   if (n < 0) throw new Error('Fatorial não é definido para números negativos.');
```

```

20   if (n === 0 || n === 1) return 1;
21   let resultado = 1;
22   for (let i = 2; i <= n; i++) { resultado *= i; }
23   return resultado;
24 }
25 function mediaArray(numeros) {
26   if (numeros.length === 0) return 0;
27   return somaArray(numeros) / numeros.length;
28 }
29 function somaArray(numeros) {

```

Figure 1: Relatório do Stryker para o Mutante #15 (raizQuadrada).

## 2.2 Mutante #177: EqualityOperator

Este mutante expõe a fraqueza de não testar casos de borda e igualdade.

- **O que a mutação fez:** Alterou a função `isMaiorQue(a, b)` de `return a > b;` para `return a >= b;`.
- **Por que sobreviveu:** O teste original validava apenas o "caminho feliz": `expect(isMaiorQue(10, 5)).toBe(true);`. Este teste passa tanto no código original (`10 > 5` é verdadeiro) quanto no código mutante (`10 >= 5` também é verdadeiro). A suíte de testes carecia de uma validação do caso de igualdade.

## 2.3 Mutante #195: ArrowFunction (Lógica de Ordenação)

Este foi um mutante sutil que sobreviveu devido a um teste com dados de entrada fracos.

- **O que a mutação fez:** Na função `medianaArray`, o mutante quebrou a lógica de ordenação, substituindo `(a, b) => a - b` por `() => undefined`.
- **Por que sobreviveu:** O teste original usava um array **já ordenado**: `medianaArray([1, 2, 3, 4, 5])`. Se a função `.sort()` for quebrada (como fez o mutante), o array "ordenado" continua sendo `[1, 2, 3, 4, 5]`, e o cálculo da mediana (3) permanece correto. O teste passou, mas a função estava com um bug crítico.

# 3 Solução Implementada

Para "matar" os mutantes analisados, os seguintes testes foram adicionados ou modificados:

1. **Para o Mutante #15 (raizQuadrada):** Foi adicionado um teste específico para o caminho de erro, que falha no código mutante.

```

% Teste adicionado (mata o mutante #15 e #18)
test('6c. deve lan ar erro para raiz de n mero negativo', () => {
  expect(() => raizQuadrada(-1))

```

```
.toThrow('N o      poss vel calcular a raiz quadrada de um n mero
negativo.');
```

**Justificativa:** O novo teste 6c força a execução do caminho que o mutante #15 tentou desativar. O código original lança um erro, mas o código mutado (if (false)) não, fazendo com que a asserção toThrow falhe e, assim, "mate" o mutante.

2. **Para o Mutante #177 (isMaiorQue):** Adicionamos um teste para o caso de igualdade.

```
% Teste adicionado (mata o mutante #177)
test('44b. deve retornar false se os n meros s o iguais em isMaiorQue
', () => {
  expect(isMaiorQue(5, 5)).toBe(false);
});
% Original (5 > 5)    -> false (Passa)
% Mutante (5 >= 5) -> true (Falha)
```

3. **Para o Mutante #195 (medianaArray):** O teste foi alterado para usar um array **desordenado**, forçando a função .sort() a ser executada corretamente.

```
% Teste original (fraco)
% test('47. ...', () => { expect(medianaArray([1, 2, 3, 4, 5])).toBe(3);
});

% Teste corrigido (mata o mutante #195)
test('47. deve calcular a mediana de um array mpar e desordenado', ()
=> {
  expect(medianaArray([3, 5, 1, 4, 2])).toBe(3);
});
% Original: .sort() -> [1, 2, 3, 4, 5] -> mid[2] = 3. (Passa)
% Mutante: .sort() quebrado -> [3, 5, 1, 4, 2] -> mid[2] = 1. (Falha)
```

## 4 Resultados Finais

Após a implementação da suíte de testes fortalecida (totalizando 74 testes, incluindo a correção de strings de erro), o Stryker foi executado novamente. Os resultados, conforme o log de 23:06, comprovam a melhoria massiva na qualidade da suíte.

Table 2: Comparativo das Métricas Finais

Métrica	Pontuação Inicial	Pontuação Final
<b>Pontuação de Mutação</b>	<b>59.34%</b>	<b>95.31%</b>
Mutantes Válidos	91 (estimado)	213
Mutantes Mortos (Killed)	(desconhecido)	200
Mutantes Sobreviventes (Survived)	25	10
Mutantes "Timed Out"	(desconhecido)	3
Mutantes Sem Cobertura	12	0

A pontuação de mutação saltou de 59.34% para 95.31%, um aumento drástico que ultrapassa a meta de 80% (High) e se aproxima da meta da tarefa de 98%.

Os 10 mutantes que sobreviveram (conforme log final) são:

- 4 mutantes na função `fatorial` (linha 19): Mutações lógicas (`&&`, `false`) na condição `n === 0 || n === 1`. Estes são mutantes equivalentes; o código do loop `for` abaixo já trata os casos 0 e 1 corretamente. O `if` é redundante.
- 2 mutantes em `isPar` e `isImpar` (linhas 43, 44): Mutações que trocam a lógica por `return true;`. Estes sobreviveram por falta de um teste com número ímpar para `isPar` e par para `isImpar`.
- 1 mutante em `produtoArray` (linha 84): Sobreviveu por falta de um teste com array vazio.
- 2 mutantes em `clamp` (linhas 88, 89): Mutações de igualdade (`<` para `<=`) que também são equivalentes para os testes de borda.

Atingir 100% exigiria refatorar o código (ex: remover o `if` redundante do `fatorial`) ou adicionar testes para os casos faltantes (ex: `isPar(7)`).

## 5 Conclusão

Este trabalho demonstrou que uma alta cobertura de código não é garantia de uma suíte de testes eficaz. A métrica de cobertura provou ser superficial, confirmando apenas que o código foi *\*executado\**, mas não que seu *\*comportamento\** foi *\*validado\**.

O Teste de Mutação, através do StrykerJS, expôs fraquezas críticas que passariam despercebidas. O processo de "matar mutantes" força o desenvolvedor a pensar de forma adversária sobre seu próprio código. A pergunta-chave muda de "O meu teste executa esta linha?" para "O meu teste confia nesta linha? Se esta linha falhar sutilmente, meu teste irá detetá-la?". Esta abordagem gera um nível de confiança na suíte de testes que métricas de cobertura jamais poderiam fornecer.