

# Teste de Mutação

**Thiago Andrade Ramalho**

808904

01.11.2025

Teste de Software

## Análise Inicial

A cobertura de código inicial ficou alta porque os testes executam a maior parte das linhas e ramos, mas a pontuação de mutação começou bem mais baixa, os casos eram eficientes para as entradas ideais do método, com poucas asserções fortes e quase nenhuma entrada que fosse contrária ao objetivo do método como, valores zero, limites, erros, igualdade, mensagens. Assim, muitos mutantes alteravam operadores `>`, `>=`, `&&`, `||`), removiam verificações ou trocavam constantes sem que os testes falhassem, mostrando que os testes não eram suficientes e sim fracos.. A discrepância mostra que a execução sem verificação robusta não encontra defeitos, por isso incluir casos negativos, limites e checagens específicas é essencial para aproximar a pontuação de mutação da cobertura.

```
> operacoes-mutante-lab@1.0.0 test
  ✓ 42. deve calcular a área de um retângulo (1 ms)
  ✓ 43. deve calcular o perímetro de um retângulo
  ✓ 44. deve verificar se um número é maior que outro
  ✓ 45. deve verificar se um número é menor que outro
  ✓ 46. deve verificar se dois números são iguais (1 ms)
  ✓ 47. deve calcular a mediana de um array ímpar e ordenado
  ✓ 48. deve calcular o dobro de um número
  ✓ 49. deve calcular o triplo de um número
  ✓ 50. deve calcular a metade de um número (1 ms)

-----|-----|-----|-----|-----|-----
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:        1.652 s, estimated 3 s
Ran all test suites.
❖ PS C:\Users\Thiago Andrade\Documents\GitHub\operacoes-mutante>
```

```
PS C:\Users\Thiago Andrade\Documents\Github\operacoes-mutante> npm test
✓ 33. deve verificar que um número é primo (1 ms)
✓ 34. deve calcular o 10º termo de Fibonacci (1 ms)
✓ 35. deve calcular o produto de um array (1 ms)
✓ 36. deve manter um valor dentro de um intervalo (clamp) (3 ms)
✓ 37. deve verificar se um número é divisível por outro (1 ms)
✓ 38. deve converter Celsius para Fahrenheit (2 ms)
✓ 39. deve converter Fahrenheit para Celsius (1 ms)
✓ 40. deve calcular o inverso de um número (1 ms)
✓ 41. deve calcular a área de um círculo (1 ms)
✓ 42. deve calcular a área de um retângulo (2 ms)
✓ 43. deve calcular o perímetro de um retângulo (2 ms)
✓ 44. deve verificar se um número é maior que outro (2 ms)
✓ 45. deve verificar se um número é menor que outro (1 ms)
✓ 46. deve verificar se dois números são iguais (1 ms)
✓ 47. deve calcular a mediana de um array ímpar e ordenado (1 ms)
✓ 48. deve calcular o dobro de um número (2 ms)
✓ 49. deve calcular o triplo de um número (1 ms)
✓ 50. deve calcular a metade de um número (1 ms)

Test Suites: 1 passed, 1 total
Tests: 50 passed, 50 total
Snapshots: 0 total
```

## Análise de Mutantes Críticos

No fatorial, a mutação trocou o caso-base de `n === 0 || n === 1` para `n === 0 && n === 1`, eliminando a condição verdadeira para 0 ou 1. Sobreviveu porque o teste não validava ambos os casos-base, e é morto ao acrescentar asserções para `fatorial(0)===1` e `fatorial(1)===1`.

```
20 function fatorial(n) {
21   if (n < 0) throw new Error('Fatorial não é definido para números negativos.');
```

●●●

```
22 - if (n === 0 || n === 1) return 1; ●▼●●
23 + if (n === 0 && n === 1) return 1;
24   let resultado = 1;
25   for (let i = 2; i <= n; i++) { resultado *= i; } ●
26   return resultado;
27 }
```

Em `medianaArray`, a checagem de paridade foi corrompida (`length % 2 === 0` → `length * 2 === 0`), o que passou despercebido porque só havia cenário com tamanho ímpar (ou sem garantir o ramo de tamanho par); resolve-se incluindo testes com arrays de tamanho par e também casos desordenados para verificar a ordenação interna antes do cálculo.

```
106 function medianaArray(numeros) {
107   if (numeros.length === 0) throw new Error('Array vazio ne possui mediana.');
```

●●●

```
108   const sorted = [...numeros].sort((a, b) => a - b);
109   const mid = Math.floor(sorted.length / 2);
110 - if (sorted.length % 2 === 0) { ●▼●
111 + if (sorted.length * 2 === 0) {
112     return (sorted[mid - 1] + sorted[mid]) / 2; ●●●
113   }
114   return sorted[mid];
115 }
```

Os mutantes do método clamp são equivalentes, ou seja, o comportamento observável pelos testes continua igual. Nesses casos, não há teste que mate o mutante sem mudar o próprio código-fonte, como passando a validar a mensagem exata do erro ou refatorando a lógica para expor diferenças reais, dessa forma esses mutantes permanecem vivos sem indicar falha prática na suíte.

```

87  function clamp(valor, min, max) {
88  -  if (valor < min) return min; ▼
89  +  if (valor <= min) return min;
90  if (valor > max) return max; ●
    return valor;

```

## Solução Implementada

Os novos testes utilizam três formas para derrubar mutantes, exceções e mensagens, pois além de esperar o throw, validam o texto exato da mensagem, pegando mutações que trocam string ou removem o erro. Casos-limite e fronteiras, eles cobrem vazio, negativo, zero, min e max, e igualdade, diferenciando > de >=, < de <= e || de &&. Por último as propriedades funcionais, elas checam passos essenciais, como ordenar antes da mediana, identidades neutras, como produto de array vazio. Com isso, deixa-se de testar apenas o resultado esperado e passa a cobrir bordas e invariantes, e capturando desvios sutis. Dessa forma os únicos sobreviventes agora são mutantes de fato equivalentes, que só morrem com mudanças no código-fonte.

```

Ran 2.12 tests per mutant on average.
-----|-----|-----|-----|-----|-----|
File    | % Mutation score | # killed | # timeout | # survived | # no cov | # errors |
-----|-----|-----|-----|-----|-----|
All files | 98.54 | 98.54 | 198 | 5 | 3 | 0 | 0 |
operacoes.js | 98.54 | 98.54 | 198 | 5 | 3 | 0 | 0 |
-----|-----|-----|-----|-----|-----|
17:47:36 (10004) INFO HtmlReporter Your report can be found at: file:///C:/Users/Thiago%20And
on.html
17:47:36 (10004) INFO MutationTestExecutor Done in 37 seconds.

```

## Conclusão

O teste de mutação é importante porque o foco sai da quantidade de cobertura para a qualidade das asserções. Ao acrescentar algumas alterações no código e verificar se os testes as detectam, ele revela pontos cegos que a cobertura tradicional não mostra, como ramos pouco verificados, mensagens de erro não acertadas e casos de fronteira ausentes. Isso faz

com que a escrita de testes seja mais precisa, reduz a probabilidade de regressões discretas e fortalece a confiança no software. Assim a cobertura diz onde os testes passam e a mutação diz quão bem eles protegem o que importa.