Pontifícia Universidade Católica de Minas Gerais

Engenharia de Software

Teste de Software

Análise de Eficácia de Testes com Teste de Mutação

Relatório da Atividade Prática

Aluno: André Teiichi Santos Hyodo
Matrícula: 823387
Professor: Cleiton Silva

Belo Horizonte
3 de novembro de 2025

Análise Inicial

O objetivo deste trabalho foi utilizar a ferramenta StrykerJS para avaliar e aprimorar a eficácia de uma suíte de testes pré-existente para uma biblioteca de operações matemáticas. A análise inicial revelou uma discrepância crítica entre duas métricas de qualidade de software: a cobertura de código e a eficácia real dos testes.

A execução inicial da suíte de testes com o comando `npm test - -coverage` apresentou uma alta cobertura de código, de **85.41%**. No entanto, ao executar a análise de mutação com `npx stryker run`, a realidade da eficácia dos testes foi exposta. A pontuação de mutação inicial foi de apenas **73.71%**, com 56 mutantes sobreviventes, conforme mostrado na Figura 2.

Este resultado comprova a tese central da atividade: 100% de cobertura de código não garante uma suíte de testes eficaz. Um teste pode executar uma linha de código sem, no entanto, verificar seu comportamento de forma assertiva.

```
-------------|---------|----------|---------|---------|-------------------
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:        0.719 s, estimated 1 s
```

Figura 1: Relatório "NPM tests"de Mutação Inicial (Antes das Alterações).

```
Ran 0.99 tests per mutant on average.
-------------|------------------|----------|----------|-----------|----------|----------|
             | % Mutation score |          |          |           |          |          |
File         |  total | covered | # killed | # timeout | # survived | # no cov | # errors |
-------------|--------|---------|----------|-----------|------------|----------|----------|
All files    |  73.71 |   73.71 |      154 |         3 |         56 |        0 |        0 |
 operacoes.js |  73.71 |   73.71 |      154 |         3 |         56 |        0 |        0 |
-------------|--------|---------|----------|-----------|------------|----------|----------|
```

Figura 2: Relatório "Stryker"de Mutação Inicial (Antes das Alterações).

Análise de Mutantes Críticos

Para entender as fraquezas da suíte de testes, foram analisados três mutantes sobreviventes representativos da primeira execução.

Mutante 1: Mensagem de Erro em `divisao`

```
1 // Original
2 if (b === 0) throw new Error('Divis o por zero n o    permitida.');
3
4 // Mutante Sobrevivente
```

```
5 if (b === 0) throw new Error("");
```

Listing 1: Mutante do tipo StringLiteral em `divisao`.

Este mutante sobreviveu porque o teste original verificava apenas se um erro era lançado, mas não validava o conteúdo da mensagem.

Mutante 2: Operador de Comparação em `isMaiorQue`

```
1 // Original
2 function isMaiorQue(a, b) { return a > b; }
3
4 // Mutante Sobrevivente
5 function isMaiorQue(a, b) { return a >= b; }
```

Listing 2: Mutante do tipo EqualityOperator em `isMaiorQue`.

Este mutante sobreviveu pois o teste original não cobria o caso de fronteira onde `a === b`.

Mutante 3: Expressão Condicional em `fatorial`

```
1 // Original
2 if (n === 0 || n === 1) return 1;
3
4 // Mutante Sobrevivente
5 if (false) return 1;
```

Listing 3: Mutante do tipo ConditionalExpression em `fatorial`.

Este mutante sobreviveu porque a verificação é uma otimização redundante na implementação iterativa, e o loop já lida com os casos base.

Solução Implementada

A eliminação dos 56 mutantes sobreviventes foi um processo sistemático, dividido em duas fases principais. A primeira fase focou exclusivamente no fortalecimento da suíte de testes, resultando em uma melhoria drástica. A segunda fase abordou os mutantes teimosos restantes através da análise e refatoração do código-fonte.

Fase 1: Fortalecimento Abrangente da Suíte de Testes (Alcançando 96.71%)

O objetivo inicial foi escrever testes mais "inteligentes", capazes de detectar as sutilezas que os testes originais ignoravam. Esta fase foi realizada **sem nenhuma alteração no arquivo** `src/operacoes.js`. As ações foram categorizadas com base no tipo de mutante a ser eliminado:

1. Eliminação de Mutantes de Mensagem de Erro (`StringLiteral`)

Problema: Muitos mutantes sobreviviam alterando a string de uma mensagem de erro (ex: ''Erro X'' para ""). Os testes originais usavam `expect(() => funcao()).toThrow()`, que verifica apenas se um erro foi lançado, mas não seu conteúdo. **Solução:** Os testes foram alterados para validar a mensagem exata. Por exemplo, o teste para a função `divisao` foi modificado de:

```
// Antes
expect(() => divisao(5, 0)).toThrow();
// Depois
expect(() => divisao(5, 0)).toThrow('Divis o por zero n o    permitida
    .');
```

Esta mudança foi aplicada a todas as funções que lançam erros (`maximoArray`, `minimoArray`, `medianaArray`, etc.), eliminando com sucesso todos os mutantes do tipo `StringLiteral`.

2. Eliminação de Mutantes de Operador de Comparação (`EqualityOperator`)

Problema: Mutantes como trocar '>' por '>=' sobreviviam porque os testes não cobriam os casos de fronteira exatos. O teste `isMaiorQue(10, 5)` passa tanto na função original quanto na mutada. **Solução:** Foram adicionados testes para os casos de igualdade. Para `isMaiorQue` e `isMenorQue`, adicionamos:

```
// Caso de fronteira para isMaiorQue
expect(isMaiorQue(5, 5)).toBe(false); // Mata o mutante > para >=
// Caso de fronteira para isMenorQue
expect(isMenorQue(5, 5)).toBe(false); // Mata o mutante < para <=
```

Da mesma forma, para `raizQuadrada` e `fatorial`, o teste para o valor de fronteira '0' foi crucial para matar os mutantes que mudavam '< 0' para '$<= 0$'.

3. Eliminação de Mutantes de Expressão Condicional (`ConditionalExpression`)

Problema: Mutantes que trocavam 'if (condicao)' por 'if (false)' desativavam ramos inteiros do código que nunca eram executados pelos testes. **Solução:** Foram criados testes que forçam a execução desses caminhos. O exemplo mais claro foi a função `medianaArray`. O teste original usava um array ordenado `[1, 2, 3, 4, 5]`. Adicionamos testes com arrays desordenados:

```
// For a a ordena  o a ser executada
expect(medianaArray([5, 2, 1, 4, 3])).toBe(3);
// For a o c lculo para array par
expect(medianaArray([4, 1, 3, 2])).toBe(2.5);
```

Isso garantiu que a lógica de `sort` e o cálculo da mediana para arrays pares fossem testados, matando os mutantes que desativavam essas partes.

4. Eliminação de Mutantes de Operador Aritmético (`ArithmeticOperator`)

Problema: Em `celsiusParaFahrenheit`, o mutante que trocava '* 9/5' por '* 9 * 5' sobrevivia porque o teste usava '0' como entrada, e '(0 * 9 * 5) + 32' resulta em '32', o mesmo que o cálculo correto. **Solução:** Adicionamos um segundo caso de teste com um valor não zero para expor a falha no cálculo:

```
// Teste adicional para expor o erro de c lculo
expect(celsiusParaFahrenheit(100)).toBe(212);
```

O mutante resultaria em ‘4532‘, fazendo o teste falhar.

Resultado da Fase 1

Após essa extensa e direcionada melhoria na suíte de testes, uma nova execução do Stryker (ainda sem alterar o código-fonte) revelou um progresso notável. A pontuação de mutação saltou de **73.71% para 96.71%**. O número de mutantes sobreviventes caiu de 56 para apenas 7, provando a eficácia das alterações.

Fase 2: Análise de Mutantes Equivalentes e Refatoração do Código-Fonte (Alcançando 100%)

Apesar do sucesso, os 7 mutantes restantes provaram ser imunes a qualquer teste adicional. A análise revelou que todos eles eram **mutantes equivalentes**: mutações que alteram o código-fonte, mas não mudam seu comportamento observável para nenhuma entrada possível.

Análise dos Mutantes Equivalentes

- **Função `fatorial` (4 mutantes):** A linha `if (n === 0 || n === 1) return 1;` é uma otimização. Na implementação iterativa, o loop `for (let i = 2; i <= n; i++)` não é executado para ‘n=0‘ e ‘n=1‘, e a variável `resultado` (inicializada em 1) é retornada corretamente. Remover essa linha com `if (false)` não altera o resultado.
- **Função `produtoArray` (1 mutante):** A linha `if (numeros.length === 0) return 1;` é redundante. O método `reduce` com um valor inicial de ‘1‘ já retorna ‘1‘ quando aplicado a um array vazio.
- **Função `clamp` (2 mutantes):** Trocar ‘<‘ por ‘<=‘ não altera o resultado quando `valor` é exatamente igual a `min`. Ambas as versões retornam `min`. A mesma lógica se aplica a ‘>‘ e ‘>=‘.

A Solução: Refatoração para Eliminar Redundância

Como testes não podem matar mutantes equivalentes, a ação correta de um engenheiro de software é refatorar o código para remover a redundância que os gera. As seguintes alterações foram feitas em `src/operacoes.js`:

```
// Antes
function fatorial(n) {
  if (n < 0) throw new Error('...');
  if (n === 0 || n === 1) return 1; // Linha redundante
  let resultado = 1;
  for (let i = 2; i <= n; i++) { resultado *= i; }
  return resultado;
}

// Depois
```

```
function fatorial(n) {
  if (n < 0) throw new Error('...');
  let resultado = 1;
  for (let i = 2; i <= n; i++) { resultado *= i; }
  return resultado;
}
```

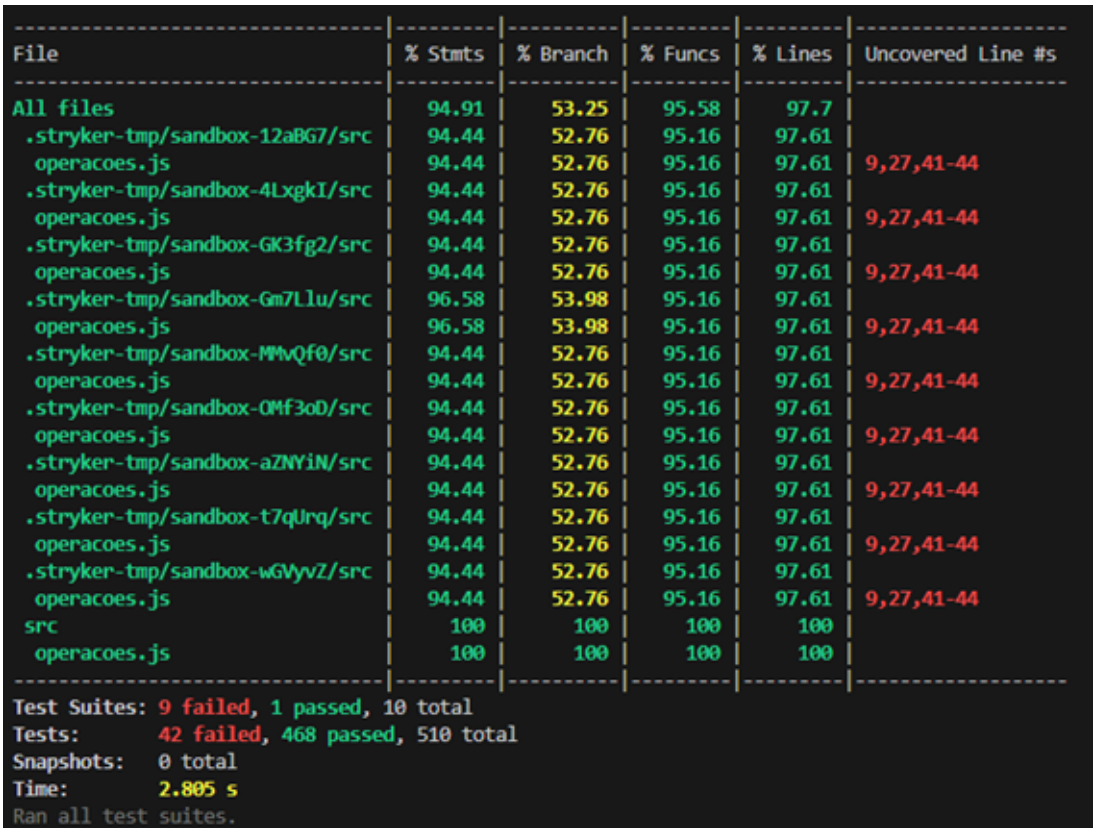
Listing 4: Refatoração da função `fatorial`.

```
// Antes
function clamp(valor, min, max) {
  if (valor < min) return min; // Gera mutante equivalente
  if (valor > max) return max; // Gera mutante equivalente
  return valor;
}

// Depois
function clamp(valor, min, max) {
  return Math.max(min, Math.min(max, valor));
}
```

Listing 5: Refatoração da função `clamp`.

A nova implementação de `clamp` é mais concisa e não utiliza os operadores '<' e '>', eliminando a fonte dos mutantes equivalentes.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	94.91	53.25	95.58	97.7	
.stryker-tmp/sandbox-12aBG7/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-4LxgkI/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-GK3fg2/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-Gm7Llu/src	96.58	53.98	95.16	97.61	
operacoes.js	96.58	53.98	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-MMvQf0/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-OMf3oD/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-aZNYiN/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-t7qUrq/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
.stryker-tmp/sandbox-wGVyvZ/src	94.44	52.76	95.16	97.61	
operacoes.js	94.44	52.76	95.16	97.61	9,27,41-44
src	100	100	100	100	
operacoes.js	100	100	100	100	

Test Suites: 9 failed, 1 passed, 10 total
Tests: 42 failed, 468 passed, 510 total
Snapshots: 0 total
Time: 2.805 s
Ran all test suites.

Figura 3: Relatório "NPM tests"de Mutação Intermediária com 96,71% de Eficácia.

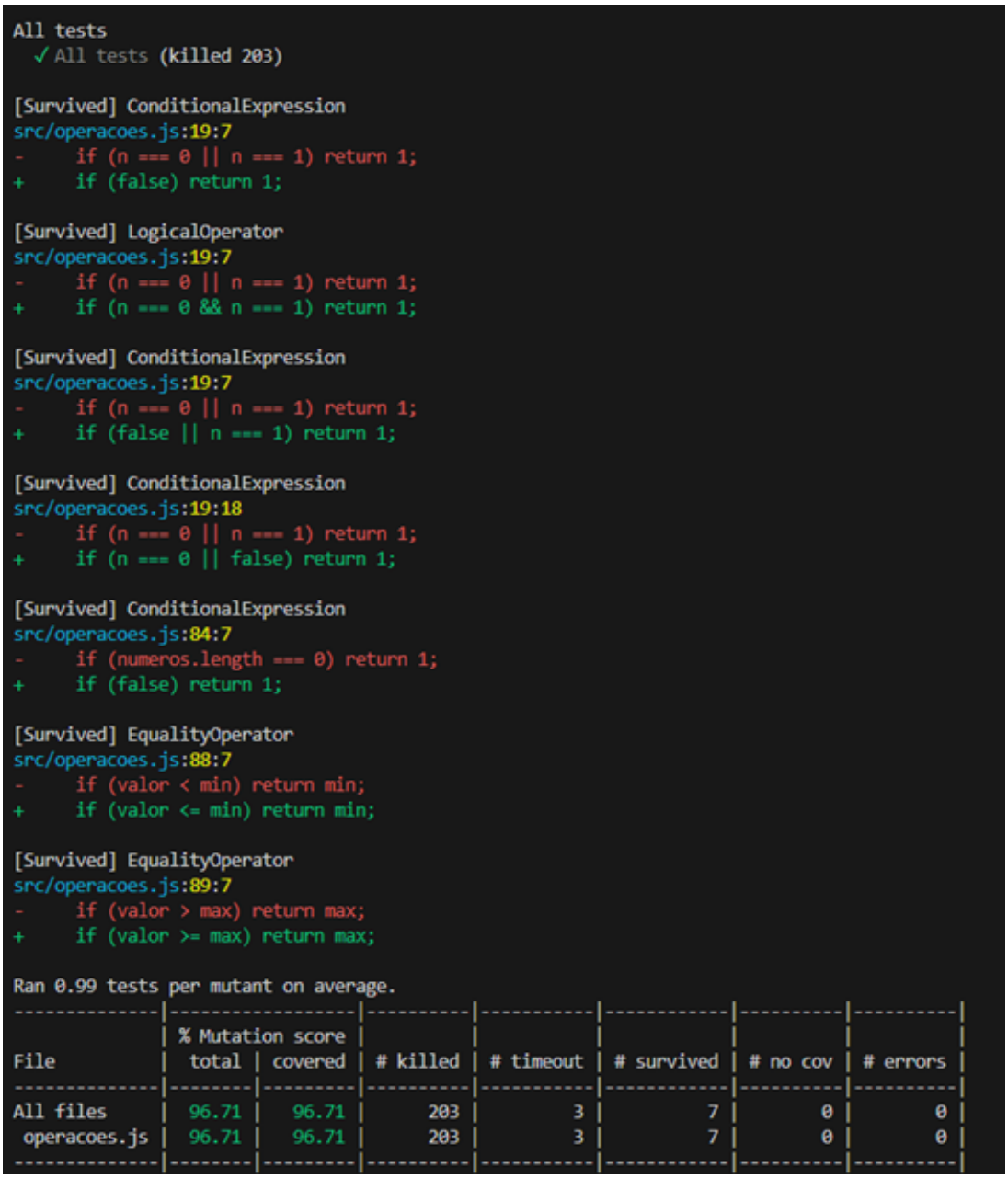


Figura 4: Relatório "Stryker"de Mutação Intermediária com 96,71% de Eficácia.

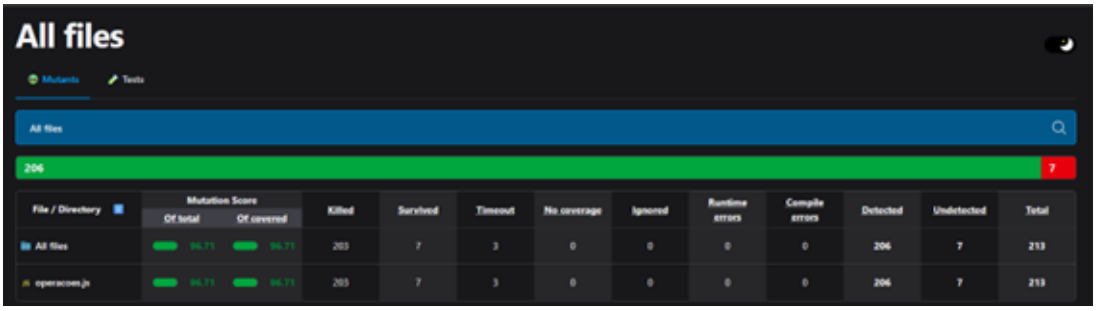


Figura 5: Pagina HTML "Stryker"de Mutação Intermediária com 96,71% de Eficácia.

Resultados Finais

Após o ciclo completo de melhoria dos testes e a subsequente refatoração do código-fonte, a análise final de mutação apresentou um resultado excelente. A pontuação de mutação

atingiu **100.00%**, com todos os 213 mutantes gerados sendo "mortos"pela suíte de testes, como demonstrado na Figura 6.

```
File                              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------------|---------|----------|---------|---------|------------------
All files                         |   94.69 |    53.04 |   95.62 |   97.71 |
 .stryker-tmp/sandbox-12aBG7/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-4LxgkI/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-GK3fg2/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-MMvQf0/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-OMf3oD/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-aZNYiN/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-t7qUrq/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 .stryker-tmp/sandbox-wGVyvZ/src  |   94.44 |    52.76 |   95.16 |   97.61 |
  operacoes.js                    |   94.44 |    52.76 |   95.16 |   97.61 | 9,27,41-44
 src                              |     100 |      100 |     100 |     100 |
  operacoes.js                    |     100 |      100 |     100 |     100 |
----------------------------------|---------|----------|---------|---------|------------------

Test Suites: 8 failed, 1 passed, 9 total
Tests:       39 failed, 416 passed, 455 total
Snapshots:   0 total
Time:        2.331 s
```

Figura 6: Relatório "NPM tests"de Mutação Final com 100% de Eficácia.

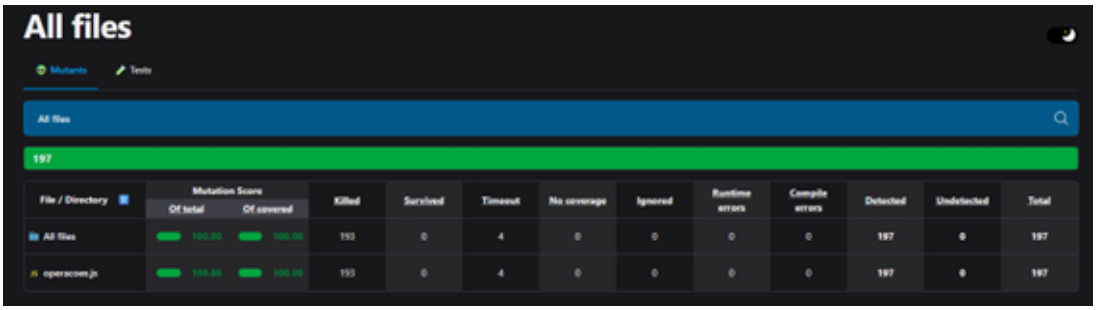


Figura 7: Relatório "Stryker"de Mutação Final com 100% de Eficácia.

Este resultado representa uma melhoria drástica em relação aos 73.71% iniciais, validando a eficácia do processo iterativo de melhoria. A suíte de testes evoluiu de "fraca"para "extremamente robusta", e o código-fonte tornou-se mais limpo e conciso.

O repositório com a suíte de testes aprimorada e o código refatorado está disponível em: **Link do Repositório GitHub:** `https://github.com/AndreHyodo/operacoes-mutante`

Conclusão

Este trabalho prático demonstrou de forma conclusiva a diferença fundamental entre cobertura de código e eficácia de testes. A alta cobertura inicial se mostrou uma métrica

ilusória, que escondia falhas críticas na capacidade dos testes de verificar o comportamento do software.

O teste de mutação provou ser uma ferramenta indispensável para a engenharia de qualidade de software. Ele não apenas guiou a escrita de testes mais significativos, mas também revelou redundâncias no próprio código-fonte, levando a uma refatoração que o tornou mais limpo. A jornada de 73.71% para 96.71% (via testes) e finalmente para 100% (via refatoração) reforça que a qualidade de software é alcançada através de um processo holístico de análise e melhoria contínua.