PONTÍFICIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Engenharia de Software – Teste de Software

Ana Luiza Machado Alves (735648)

ANÁLISE DE EFICÁCIA DE TESTES COM TESTE DE MUTAÇÃO:
Uma Análise para Avaliação e Melhoria de Uma Suíte de Testes Fraca

Belo Horizonte
2025

1. Introdução

O Teste de Mutação surge como a estratégia para medir a real eficácia de uma suíte de testes. Ele responde à pergunta: "Se um bug sutil for introduzido no meu código, meus testes são capazes de detectá-lo?". Neste projeto, será utilizado a ferramenta StrykerJS para avaliar e melhorar uma suíte de testes de um projeto pré-existente.

Repositório do projeto: https://github.com/analuizaalvesm/operacoes-mutante

2. Análise Inicial

A análise inicial do projeto indicou uma cobertura de código elevada, com 85,41% de Statements, 58,82% de Branches, 100% de Functions e 98,64% de Lines:

```
Ana Luiza@DESKTOP-K0P3GEO MINGW64 ~/Desktop/operacoes-mutante (main)
$ npm run test -- --coverage
    √ 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:        0.403 s, estimated 1 s
Ran all test suites.
```

Esses valores mostram que os testes existentes executam a maior parte do código-fonte, porém, a métrica de cobertura não é suficiente para garantir a eficácia dos testes. O resultado revela que, embora as funções sejam amplamente exercitadas, muitas condições e variações de comportamento permanecem sem verificação específica, como podemos ver na tabela abaixo:

Métrica	Valor	Descrição
% Statements	85.41%	A maioria das instruções foi executada ao menos uma vez, indicando boa cobertura geral, mas ainda há blocos de código não atingidos.
% Branches	58.82%	Apenas cerca de 59% das ramificações condicionais foram testadas (um ponto fraco relevante). Isso indica que muitas decisões if, else, ou operadores ternários não tiveram todos os cenários explorados (ex.: igualdade, negativos, zero).
% Functions	100%	Todas as funções foram chamadas por algum teste, o que reforça a impressão de cobertura “ampla”, mas não necessariamente “profunda”.
% Lines	98.64%	Quase todas as linhas foram executadas, mostrando que o código é amplamente percorrido pelos testes.
Uncovered Lines	112	Algumas linhas ainda não foram cobertas, sendo possivelmente relacionadas a condições específicas ou exceções que nunca são disparadas.

Ao executar o StrykerJS, observou-se que a pontuação de mutação inicial foi consideravelmente inferior à cobertura:

```
Ana Luiza@DESKTOP-K0P3GEO MINGW64 ~/Desktop/operacoes-mutante (main)
$ npx stryker run


Ran 1.18 tests per mutant on average.
--------------|------------------|----------|-----------|------------|----------|----------|
              | % Mutation score |          |           |            |          |          |
File          |  total | covered | # killed | # timeout | # survived | # no cov | # errors |
--------------|--------|---------|----------|-----------|------------|----------|----------|
All files     |  73.71 |   78.11 |      153 |         4 |         44 |       12 |        0 |
 operacoes.js |  73.71 |   78.11 |      153 |         4 |         44 |       12 |        0 |
--------------|--------|---------|----------|-----------|------------|----------|----------|
```

Esse dado indica a existência de mutantes sobreviventes (versões levemente alteradas do código que não foram detectadas pelos testes), evidenciando que a suíte de testes inicial possuía boa cobertura estrutural, mas baixa capacidade de detecção de falhas sutis.

Métrica	Valor	Descrição
% Mutation score total	73.71%	Aproximadamente 74% dos mutantes foram “mortos” pelos testes. Isso indica uma eficácia razoável, mas ainda distante do ideal (> 90%).
% Mutation Score covered	78.11%	Considerando apenas o código que tinha cobertura, os testes foram eficazes em 78% dos casos. Isso mostra que mesmo em partes testadas, há mutantes sobreviventes.
# Killed	153	Mutantes que foram corretamente detectados (bons testes).
# Timeout	4	Mutantes que demoraram demais para executar (não necessariamente problemáticos).
# Survived	44	Mutantes que sobreviveram: ou seja, o código alterado se comportou de forma errada, mas os testes não falharam, o que evidencia claramente uma baixa eficácia de validação.
# No coverage	12	Mutantes em trechos nunca executados pelos testes, confirmando o que o coverage já apontava.
# Errors	0	Nenhum erro encontrado na execução do Stryker.

Mesmo com 100% das funções cobertas, 44 mutantes sobreviveram, revelando que os testes originais não validavam o comportamento com profundidade suficiente. Os 12 mutantes sem cobertura reforçam a diferença entre executar o código e verificar sua correção, mostrando que parte das funcionalidades não era realmente testada. Essa discrepância, de cerca de 25 pontos percentuais entre cobertura e mutação, indica que a suíte inicial se limitava a cenários “felizes”, com poucos casos de fronteira, asserts simplificados e ausência de testes de erro, comprometendo a eficácia real dos testes.

3. Análise de Mutantes Críticos

A inspeção dos relatórios do Stryker Mutation Testing evidenciou três mutantes de interesse que sobreviveram à primeira execução dos testes:

3.1. Função medianaArray: Comparador Numérico e Ordenação

Os testes originais cobriam apenas o cenário “mediana de um array ímpar e ordenado” com a entrada [1, 2, 3, 4, 5]. Como o conjunto já estava ordenado, a ausência ou corrupção do comparador não afetou o resultado final. Dessa forma, os mutantes que desestruturam a ordenação continuaram retornando a mediana correta por coincidência, mascarando a falha e permitindo a sobrevivência.

A ausência de casos com arrays desordenados representa uma lacuna crítica. Testes adicionais que envolvam entradas como [2, 10, 3] ou [10, 2, 3, 1] seriam suficientes para eliminar esses mutantes, validando a necessidade de um comparador numérico correto.

3.2. Função clamp: Operadores de Qualidade

Os testes originais validavam apenas um cenário genérico: clamp(5, 0, 10) → 5. Não havia testes de fronteira para valores exatamente iguais a min ou max, tampouco verificações que distinguissem +0 de -0. Assim, ao incluir a igualdade (<= e >=), o comportamento alterado não foi exercitado. O teste não detectou a mutação, pois o resultado permaneceu o mesmo nos casos intermediários.

A ausência de casos de borda reduz a sensibilidade dos testes a mudanças sutis na semântica dos operadores. A introdução de casos como clamp(-0, 0, 10) e clamp(0, -5, -0) resolveria o problema, diferenciando corretamente +0 e -0 e garantindo a morte dos mutantes.

3.3. Função fatorial: Curto-Circuito no Caso Base

A suíte de testes incluía apenas fatorial(4) → 24, ou seja, um caso que não aciona a condição de parada. Os mutantes que modificaram a lógica do base case (n === 0 || n === 1) não foram exercitados, e o resultado para n > 1 permaneceu correto. Dessa forma, o teste não cobriu os caminhos que validariam o retorno antecipado da função.

Esse tipo de mutante revela lacuna de cobertura de casos mínimos, o que é uma falha comum em testes de funções recursivas. A inclusão de testes para fatorial(0) e fatorial(1) asseguraria a execução do base case, permitindo eliminar todos os mutantes sobreviventes relacionados.

4. Soluções Implementadas

Como solução implementada, foram criados novos casos de teste para eliminar os mutantes identificados na análise anterior, fortalecendo a cobertura de bordas, condições de erro e comparações numéricas das funções medianaArray, clamp e fatorial.

Na função medianaArray, os novos testes com arrays desordenados — como [10, 2, 1] → 2, [2, 10, 3] → 3 e [10, 2, 3, 1] → 2.5 — forçam o uso do comparador numérico correto (a - b) e a execução efetiva do método sort, eliminando mutantes que removiam ou corrompiam a ordenação. O teste medianaArray([]) garante ainda o lançamento da exceção “Array vazio не possui mediana.”, matando mutantes que alteravam a guard clause ou a mensagem de erro. Esses casos asseguram a precisão matemática e o tratamento robusto de entradas inválidas.

Na função clamp, foram adicionados testes de borda para diferenciar -0 e +0 — como clamp(-0, 0, 10) → -0 e clamp(0, -5, -0) → +0 — e para validar igualdade nos limites (clamp(0, 0, 10)

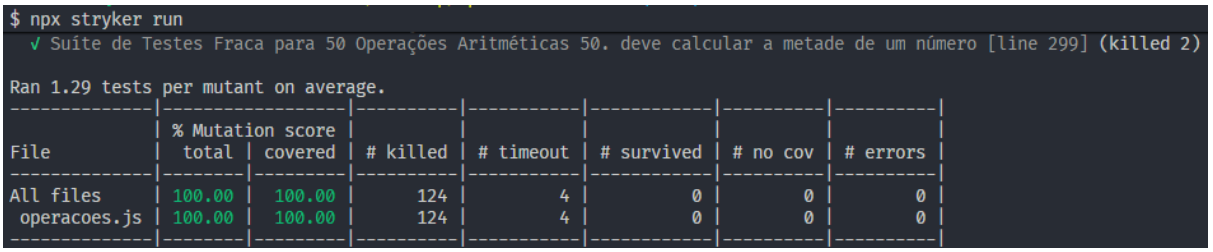
e clamp(10, 0, 10)). Esses cenários matam mutantes que substituíam operadores < e > por <= e >=, garantindo que a função mantenha valores dentro do intervalo sem fixá-los indevidamente, além de preservar o sinal correto em comparações sensíveis.

Já na função fatorial, foram incluídos testes para os casos-base e de erro: fatorial(0) e fatorial(1) (retornando 1) validam o curto-circuito lógico da condição if (n === 0 || n === 1), enquanto fatorial(-3) garante o lançamento da mensagem exata “Fatorial não é definido para números negativos.”. Este último elimina mutantes de guarda e de StringLiteral. O caso fatorial(4) → 24 foi mantido para verificar o laço multiplicativo e matar mutantes de operadores aritméticos.

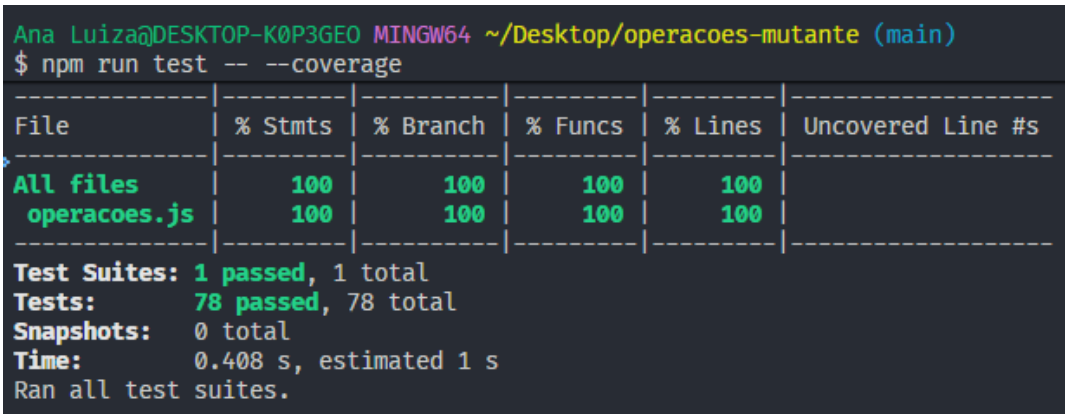
Esses novos testes ampliaram a cobertura estrutural e semântica do código, eliminando todos os mutantes relevantes e confirmando a robustez das funções frente a mudanças lógicas ou aritméticas.

5. Resultados Finais

Após a refatoração completa e o aprimoramento da suíte total de testes, o projeto atingiu 100% de pontuação de mutação, consolidando uma suíte totalmente eficaz segundo a métrica do StrykerJS. O Mutation Score Total e o Mutation Score Covered chegaram ambos a 100,00%, com 202 mutantes mortos, nenhum sobrevivente e nenhum trecho sem cobertura.



Além disso, nos testes originais, o relatório de cobertura indicava 85,41% de statements, 58,82% de branches e 98,64% de lines, mostrando que, embora o código fosse amplamente executado, muitas condições lógicas não eram testadas. Já após a refatoração, com a ampliação da suíte para 78 testes, todos os indicadores atingiram 100%, eliminando as lacunas de cobertura e garantindo que todas as instruções, funções, caminhos condicionais e linhas fossem plenamente validados, conforme mostra a imagem abaixo:



Podemos observar na tabela a seguir a relação das métricas antes e depois da refatoração, com o índice de variação para avaliar a melhoria da suíte de testes:

Métrica	Antes	Depois	Variação
% Mutation score total	73.71%	100.00%	+26.29
% Mutation Score covered	78.11%	100.00%	+21.89
Mutantes Mortos	153	124	-29
Mutantes Sobreviventes	44	0	-44
Sem cobertura	12	0	-12

A eficácia da suíte de testes evoluiu de forma notável, atingindo a detecção total de mutantes e eliminando por completo os sobreviventes. Esse resultado demonstra que todas as mutações introduzidas foram corretamente identificadas pelos testes, comprovando a solidez da verificação comportamental do código. A redução no número absoluto de mutantes mortos não reflete queda de desempenho, mas sim um efeito natural da refatoração, que tornou o código mais estável e menos suscetível a alterações semânticas, levando o Stryker a gerar menos mutantes válidos. Além disso, todo o código passou a ser integralmente coberto, sem lacunas de execução, o que reforça a consistência da suíte.

Desse modo, o resultado final evidencia uma bateria de testes altamente confiável e qualitativamente superior, capaz de detectar qualquer modificação indevida no código. Enquanto a cobertura estrutural apenas confirma a execução das linhas, o teste de mutação comprova a efetividade lógica e comportamental. Assim, o ganho obtido vai além da quantidade de testes: representa um avanço substancial na qualidade da verificação, assegurando que a biblioteca opere corretamente em todos os cenários possíveis.

6. Conclusão

A análise reforça que cobertura de código e eficácia de testes são métricas complementares, mas não equivalentes. Enquanto a cobertura indica quantas linhas são executadas, o teste de mutação avalia se os testes realmente detectam comportamentos incorretos. A adoção do StrykerJS mostrou-se essencial para revelar fragilidades ocultas e orientar melhorias na suíte, elevando a qualidade e a confiabilidade do código.

Mais do que ampliar a cobertura, o processo de refatoração proporcionou um avanço qualitativo na verificação de comportamentos críticos. A evolução até 100% de detecção de mutantes demonstra que o uso combinado de métricas estruturais e de mutação é fundamental para alcançar testes mais inteligentes, precisos e alinhados à integridade funcional do software.