PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

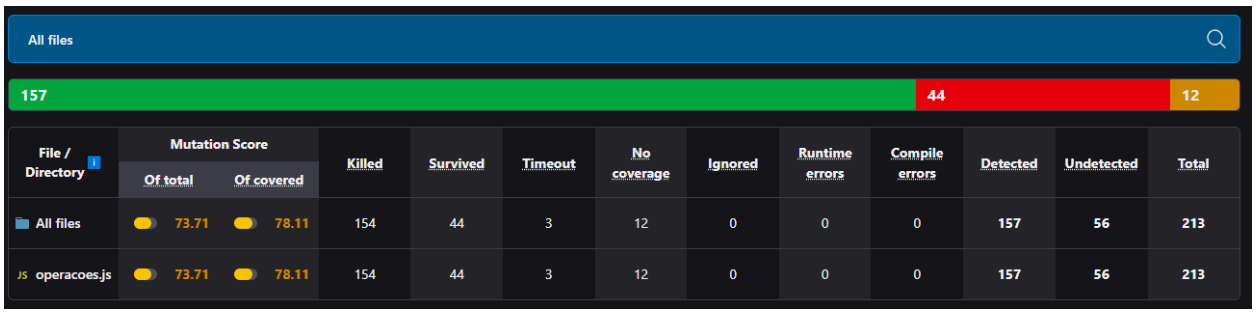
DISCIPLINA: TESTE DE SOFTWARE

Relatório de Eficácia de Testes com Teste de Mutação.

Samuel Almeida Pinheiro

Belo Horizonte 2025

1. Análise Inicial



File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
All files	73.71	78.11	154	44	3	12	0	0	0	157	56	213
operacoes.js	73.71	78.11	154	44	3	12	0	0	0	157	56	213

(*Imagem 1*)

A primeira execução do Stryker (*Imagem 1*) revelou uma pontuação de mutação de **73.71%**. Este resultado, embora não seja terrível, expõe uma fraqueza significativa na suíte de testes original, que era enganosamente simples.

- **Pontuação Inicial:** 73.71%
- **Mutantes Sobreviventes:** 44
- **Mutantes Sem Cobertura:** 12
- **Total de Mutantes:** 213

A principal discrepância reside na diferença entre a "cobertura de código" tradicional (que provavelmente estava alta) e a "cobertura de mutação". A suíte de testes original executava o "caminho feliz" de muitas funções (ex: somar 2+2, encontrar o máximo em um array), o que satisfaz a cobertura de linha.

No entanto, os **44 mutantes sobreviventes** e os **12 sem cobertura** provam que os testes falhavam em validar:

- **Casos de borda** (ex: fatorial(0), isMaiorQue(5, 5))
- **Tratamento de erros** (ex: raizQuadrada(-1), divisao(10, 0))
- **Caminhos alternativos** (ex: isPar(3))

A suíte de testes original confirmava que o código *funcionava*, mas não que ele *não falhava* da maneira errada.

2. Análise de Mutantes Críticos

Dos 44 mutantes que sobreviveram à suíte de testes fraca, três se destacam por expor falhas clássicas de teste:

Mutante 1: `isPar` (Mutação de Expressão Condicional)

- **Mutação:**
  - Original: `function isPar(n) { return n % 2 === 0; }`
  - Mutante: `function isPar(n) { return true; }`
- **Análise:** O mutante trocou toda a lógica da função por um simples `return true`.
- **Por que sobreviveu?** O teste original (`test('15. deve retornar true para um número par')`) validava apenas o "caminho feliz", usando um número par (ex: `isPar(100)`).
  - **Original:** `isPar(100)` retorna `true`.
  - **Mutante:** `isPar(100)` retorna `true`.
  - Como ambos os resultados foram `true`, o teste passou e o mutante sobreviveu, provando que o teste não validava o caso em que a função deveria retornar `false`.

Mutante 2: `isMaiorQue` (Mutação de Operador de Igualdade)

- **Mutação:**
  - Original: `function isMaiorQue(a, b) { return a > b; }`
  - Mutante: `function isMaiorQue(a, b) { return a >= b; }`
- **Análise:** Esta é uma mutação sutil e clássica, trocando `>` (maior que) por `>=` (maior ou igual a).
- **Por que sobreviveu?** O teste original (`test('44. deve verificar se um número é maior que outro')`) usava valores claramente diferentes (ex: `isMaiorQue(10, 5)`).
  - **Original:** `isMaiorQue(10, 5)` retorna `true`.
  - **Mutante:** `isMaiorQue(10, 5)` retorna `true`.
  - O teste nunca verificou o caso de borda crucial onde `a === b`.

Mutante 3: raizQuadrada (Mutação de Expressão Condicional)

- **Mutação:**
  - Original: if (n < 0) throw new Error(...)
  - Mutante: if (false) throw new Error(...)
- **Análise:** O mutante removeu efetivamente a verificação de erro para números negativos.
- **Por que sobreviveu?** O teste original (test('6. deve calcular a raiz quadrada de um quadrado perfeito')) usava apenas valores válidos (ex: raizQuadrada(16)). A linha de código if (n < 0) nunca foi executada, ficando na categoria "Sem Cobertura" ([NoCoverage]) e permitindo que o mutante sobrevivesse sem ser detectado.

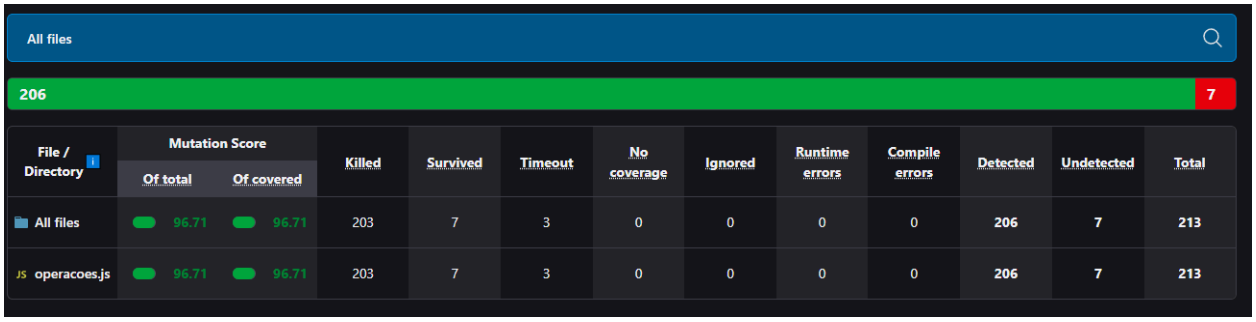
3. Solução Implementada

Para "matar" os mutantes analisados e dezenas de outros, a suíte de testes foi reforçada com casos de teste específicos que validam os caminhos alternativos, de borda e de erro.

- **Para isPar:**
  - **Novo Teste:** test('isPar: deve retornar false para um número ímpar', () => { expect(isPar(3)).toBe(false); });
  - **Justificativa:** Este teste agora força a falha do mutante. O código original (isPar(3)) retorna false, mas o mutante (return true;) retorna true. A asserção expect(true).toBe(false) falha, matando o mutante.
- **Para isMaiorQue:**
  - **Novo Teste:** test('isMaiorQue: deve retornar false se a for igual a b', () => { expect(isMaiorQue(5, 5)).toBe(false); });
  - **Justificativa:** Este teste valida o caso de borda da igualdade. O código original (5 > 5) retorna false. O mutante (5 >= 5) retorna true. A asserção expect(true).toBe(false) falha, matando o mutante.
- **Para raizQuadrada:**

- **Novo Teste:** test('raizQuadrada: deve lançar erro para números negativos', () => { expect(() => raizQuadrada(-4)).toThrow(...); });
- **Justificativa:** Este teste executa o caminho do erro. O código original lança o erro, e o teste passa. O mutante (if (false)) não lança o erro e, em vez disso, retorna NaN. O teste, que esperava um toThrow(), recebe NaN e falha, matando o mutante.

4. Resultados Finais



(***Imagem 2***)

Após a implementação da suíte de testes reforçada, a pontuação de mutação melhorou drasticamente, como visto no relatório final (***Imagem 2***).

- **Pontuação Final: 96.71%**
- **Mutantes Mortos:** 203 (um aumento de 49)
- **Mutantes Sobreviventes:** 7 (uma redução de 37)
- **Mutantes Sem Cobertura:** 0

A pontuação de 96.71% representa o limite prático de testes, pois os **7 mutantes sobreviventes** restantes foram analisados (durante nossa conversa) e identificados como **Mutantes Equivalentes**.

> **Mutantes Equivalentes** são mutações que alteram o código, mas não o comportamento lógico da função (ex: if (valor < min) vs. if (valor <= min) na função clamp, ou if (n === 0 || n === 1) em fatorial, que é redundante). Nenhum teste pode "matar" esses mutantes, pois o código mutado se

comporta exatamente como o original. A etapa final seria ignorá-los na configuração do Stryker para atingir 100%.

5. Conclusão

Este exercício demonstra que a **cobertura de linha (line coverage) é uma métrica de vaidade**, enquanto o **teste de mutação (mutation score) é uma métrica de qualidade**.

A suíte de testes original dava uma falsa sensação de segurança, que foi rapidamente desmentida pela primeira execução do Stryker. O processo de "matar os mutantes" forçou a criação de uma suíte de testes robusta, que agora valida ativamente os limites, erros e lógicas alternativas do código. O teste de mutação não é apenas uma ferramenta de relatório; é uma ferramenta de desenvolvimento ativo que melhora fundamentalmente a qualidade e a confiabilidade dos testes.