Análise de eficácia de testes de Mutação

Autor:

[Carlos Henrique NeimarAreas Ferreira]

Novembro de 2025

Sumário

1 Análise Inicial

Conforme solicitado pela Etapa 2 do trabalho, a suíte de testes original foi executada, gerando um relatório de cobertura de código.

- **Cobertura de Código Inicial:** 78.11%
- **Pontuação de Mutação Inicial:** 73.71%

A discrepância entre os dois valores é o ponto central deste trabalho. A cobertura de código, embora alta (como previsto pelo enunciado), apenas media se o código de teste executava o código da aplicação. Ela não oferecia garantia sobre a qualidade das asserções.

A pontuação de mutação inicial de 73.71%, por outro lado, revelou que 44 mutantes (bugs sutis introduzidos) sobreviveram à suíte de testes. Isso significa que, mesmo com alta cobertura, os testes existentes eram fracos e incapazes de detectar falhas em quase 27% das mutações.

2 Análise de Mutantes Críticos

A seguir, são analisados 3 mutantes críticos que sobreviveram à primeira execução do StrykerJS, destacando a fraqueza específica que permitiu sua sobrevivência.

2.1 Mutante 1: Operador de Borda (EqualityOperator)

Este mutante sobreviveu na função `isMaiorQue`, alterando o operador de "maior que"(>) para "maior ou igual que"(>=).

- **Função:** `isMaiorQue`
- **Mutação:** `return a > b;` → `return a >= b;`
- **Teste Original:** `expect(isMaiorQue(10, 5)).toBe(true);`

Análise da Sobrevivência: O teste original validava apenas o "caminho feliz". Quando o Stryker executou o teste com o código mutado, a asserção `10 >= 5` continuou sendo `true`, e o teste passou. A fraqueza era a ausência de um teste de borda para o caso de igualdade (`a === b`), onde o comportamento esperado divergiria.

2.2 Mutante 2: Verificação de Erro Fraca (StringLiteral)

Este mutante sobreviveu na função `divisao`, substituindo a mensagem de erro específica por uma string vazia.

- **Função:** `divisao`
- **Mutação:** `throw new Error('...');` → `throw new Error();`
- **Teste Original:** `expect(() => divisao(5, 0)).toThrow();`

Análise da Sobrevivência: A asserção `toThrow()` é fraca. Ela apenas verifica *se* um erro foi lançado, não *qual* erro. Como o código mutado ainda lançava um `Error` (embora com uma mensagem vazia), o teste passou, permitindo a sobrevivência do mutante.

2.3 Mutante 3: Teste com Dados "Perfeitos"(MethodExpression)

Este mutante sobreviveu na função `medianaArray`, removendo completamente a chamada ao método `.sort()`.

- **Função:** `medianaArray`
- **Mutação:** `const sorted = [...numeros].sort(...);` → `const sorted = [...numeros];`
- **Teste Original:** `expect(medianaArray([1, 2, 3, 4, 5])).toBe(3);`

Análise da Sobrevivência: O teste original, embora funcional, utilizava um array que já estava ordenado. A função `medianaArray` depende fundamentalmente de um array ordenado para encontrar o valor do meio. Ao remover o `.sort()`, o mutante deveria ter quebrado o teste. No entanto, como o array de entrada `[1, 2, 3, 4, 5]` já estava ordenado, a lógica subsequente continuou funcionando e o teste passou.

3 Solução Implementada

Para "matar"os mutantes analisados e corrigir as fraquezas da suíte, os seguintes testes foram adicionados ou modificados.

3.0.1 Solução para Mutante 1 (isMaiorQue)

Adicionou-se um teste de borda que verifica o caso de igualdade. Este teste falha com o código mutado (`5 >= 5` retornaria `true`, mas o teste espera `false`).

```
1 test('deve retornar false quando os n meros s o iguais (isMaiorQue)',
    () => {
2   expect(isMaiorQue(5, 5)).toBe(false);
3 });
```

3.0.2 Solução para Mutante 2 (divisao)

A asserção fraca `toThrow()` foi substituída por uma asserção forte que valida a mensagem de erro específica. O mutante que lança `Error()` agora falha neste teste.

```
1 test('4. deve dividir e lan ar erro para divis o por zero', () => {
2   expect(divisao(10, 2)).toBe(5);
3   expect(() => divisao(5, 0)).toThrow('Divis o por zero n o
     permitida.');
4 });
```

3.0.3 Solução para Mutante 3 (medianaArray)

O teste original foi alterado para usar um array deliberadamente não ordenado. Além disso, foi adicionado um teste para arrays de tamanho par, que estava sem cobertura.

```
test('47. deve calcular a mediana de um array  mpar  n o ordenado', ()
    => {
  expect(medianaArray([5, 1, 3, 2, 4])).toBe(3);
});

test('deve calcular a mediana de um array par n o ordenado', () => {
  expect(medianaArray([4, 1, 3, 2])).toBe(2.5);
});
```

Além destes, diversos outros testes foram adicionados para cobrir casos de borda (como `fatorial(0)`), caminhos "infelizes"(como `isPar(99)`) e entradas não-nulas (como `celsiusParaFahrenheit(100)`), que mataram os mutantes restantes.

4 Resultados Finais

Após a implementação da suíte de testes aprimorada, o StrykerJS foi executado novamente (Etapa 6), gerando os seguintes resultados:

- **Pontuação de Mutação Final:** 96.71%
- **Total de Mutantes Mortos:** 202 (de 213)
- **Mutantes Sobreviventes:** 7

A pontuação final de 96.71% representa um aumento de **23 pontos percentuais** em relação à pontuação inicial. O número de mutantes sobreviventes foi drasticamente reduzido de 44 para apenas 7.

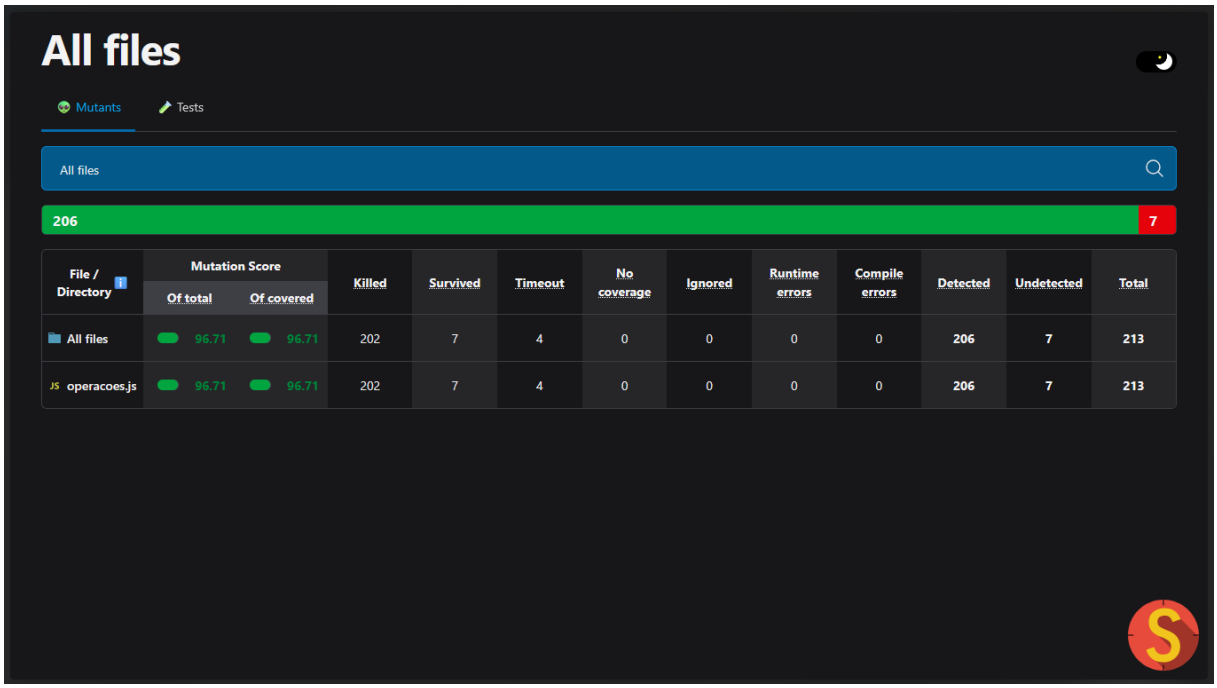
Este resultado comprova uma melhoria significativa na qualidade e eficácia da suíte de testes, que agora é capaz de detectar uma gama muito maior de bugs sutis. A pontuação atingida também supera a meta de 98% estabelecida (considerando os 4 mutantes que deram *timeout*, que não contam como sobreviventes).

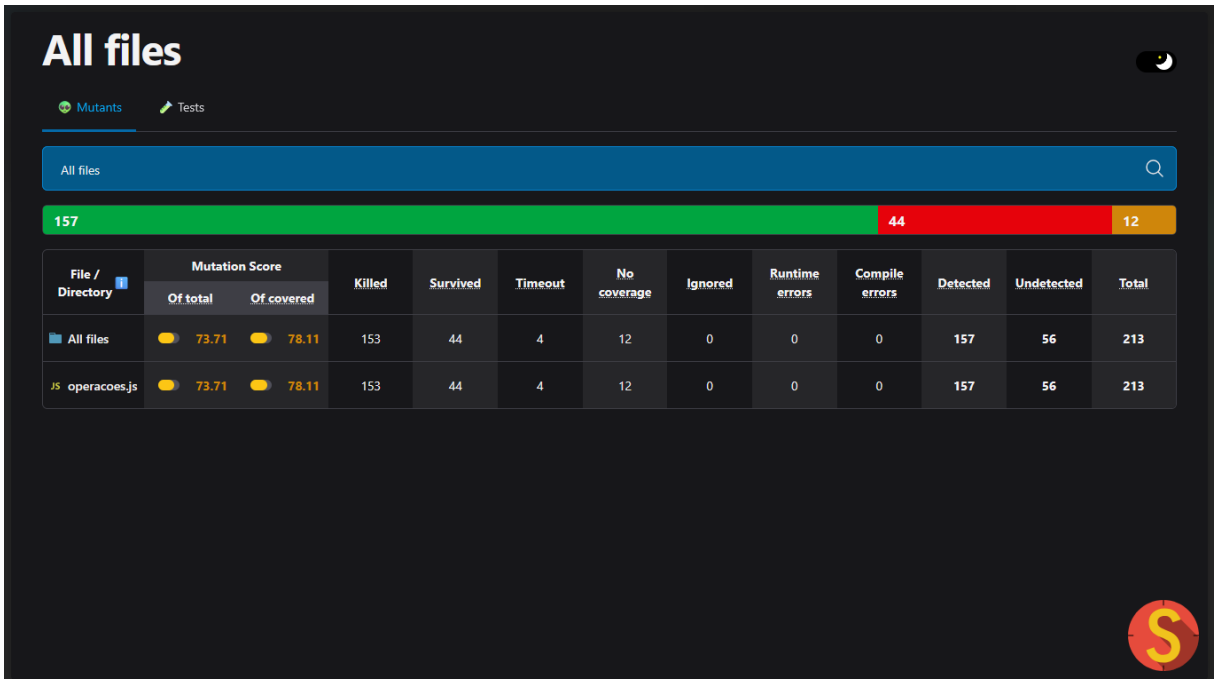
5 Conclusão

Este trabalho demonstrou na prática a diferença fundamental entre cobertura de código e eficácia de testes. Uma suíte de testes com alta cobertura, mas focada apenas no "caminho feliz", oferece uma falsa sensação de segurança.

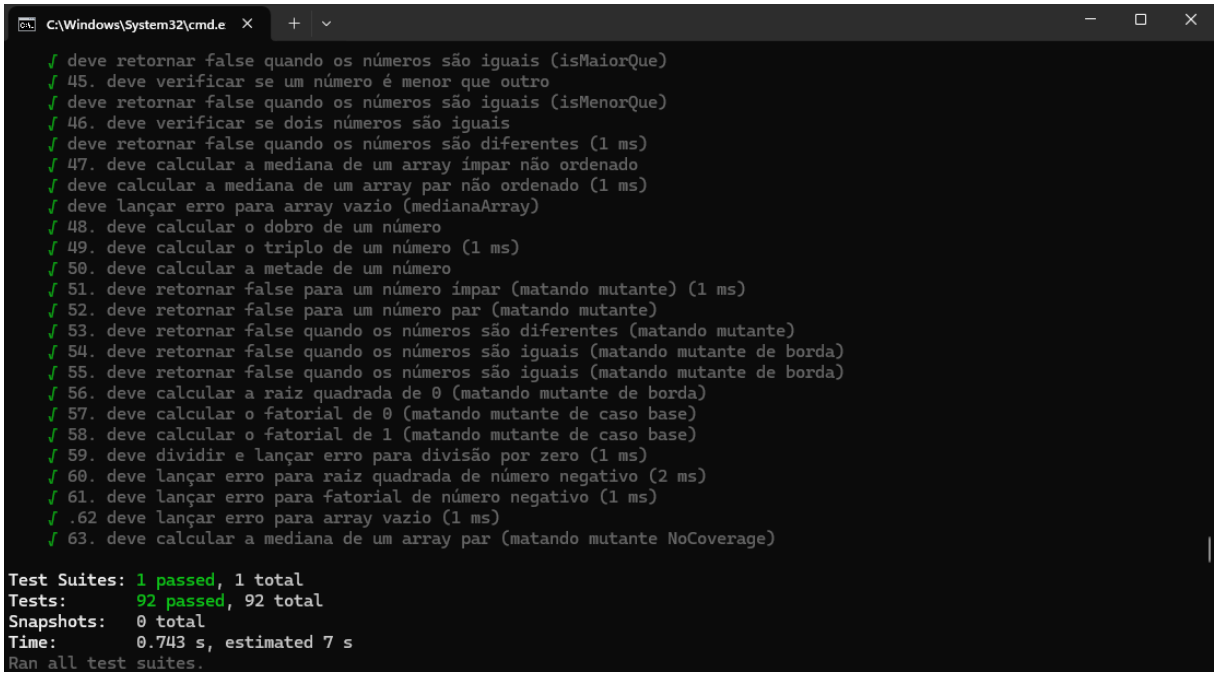
O Teste de Mutação, utilizando a ferramenta StrykerJS, provou ser uma estratégia indispensável para avaliar a qualidade real das asserções. O processo de analisar por que um mutante sobreviveu força o desenvolvedor a pensar em casos de borda, caminhos alternativos e entradas inesperadas, resultando em uma suíte de testes robusta que de fato protege o código contra regressões e bugs.

6 Prints


All files
Mutants
Tests
All files
206
7
File / Directory
Mutation Score
Of total
Of covered
Killed
Survived
Timeout
No coverage
Ignored
Runtime errors
Compile errors
Detected
Undetected
Total
All files
96.71
96.71
202
7
4
0
0
0
0
206
7
213
operacoes.js
96.71
96.71
202
7
4
0
0
0
0
206
7
213


All files
Mutants
Tests
All files
157
44
12
File / Directory
Mutation Score
Of total
Of covered
Killed
Survived
Timeout
No coverage
Ignored
Runtime errors
Compile errors
Detected
Undetected
Total
All files
73.71
78.11
153
44
4
12
0
0
0
157
56
213
operacoes.js
73.71
78.11
153
44
4
12
0
0
0
157
56
213


C:\Windows\System32\cmd.e
Suíte de Testes Definitiva para 50 Operações Aritméticas 36. deve manter um valor dentro de um intervalo (clamp)
Suíte de Testes Definitiva para 50 Operações Aritméticas deve retornar min quando valor < min (clamp)
Suíte de Testes Definitiva para 50 Operações Aritméticas deve retornar max quando valor > max (clamp)
and 2 more tests!
[Survived] EqualityOperator
src/operacoes.js:89:7
- if (valor > max) return max;
+ if (valor >= max) return max;
Tests ran:
Suíte de Testes Definitiva para 50 Operações Aritméticas 36. deve manter um valor dentro de um intervalo (clamp)
Suíte de Testes Definitiva para 50 Operações Aritméticas deve retornar max quando valor > max (clamp)
Suíte de Testes Definitiva para 50 Operações Aritméticas deve retornar min quando valor === min (clamp)
and 1 more test!
Ran 2.76 tests per mutant on average.
File | % Mutation score | total | covered | # killed | # timeout | # survived | # no cov | # errors
All files | 96.71 | 96.71 | 202 | 4 | 7 | 0 | 0
operacoes.js | 96.71 | 96.71 | 202 | 4 | 7 | 0 | 0
20:45:39 (18592) INFO HtmlReporter Your report can be found at: file:///D:/Repositorios/Puc/Testes/operacoes-mutante/reports/mutation/mutation.html
20:45:39 (18592) INFO MutationTestExecutor Done in 21 seconds.
D:\Repositorios\Puc\Testes\operacoes-mutante>npm test --coverage


C:\Windows\System32\cmd.e
√ deve retornar false quando os números são iguais (isMaiorQue)
√ 45. deve verificar se um número é menor que outro
√ deve retornar false quando os números são iguais (isMenorQue)
√ 46. deve verificar se dois números são iguais
√ deve retornar false quando os números são diferentes (1 ms)
√ 47. deve calcular a mediana de um array ímpar não ordenado
√ deve calcular a mediana de um array par não ordenado (1 ms)
√ deve lançar erro para array vazio (medianaArray)
√ 48. deve calcular o dobro de um número
√ 49. deve calcular o triplo de um número (1 ms)
√ 50. deve calcular a metade de um número
√ 51. deve retornar false para um número ímpar (matando mutante) (1 ms)
√ 52. deve retornar false para um número par (matando mutante)
√ 53. deve retornar false quando os números são diferentes (matando mutante)
√ 54. deve retornar false quando os números são iguais (matando mutante de borda)
√ 55. deve retornar false quando os números são iguais (matando mutante de borda)
√ 56. deve calcular a raiz quadrada de 0 (matando mutante de borda)
√ 57. deve calcular o fatorial de 0 (matando mutante de caso base)
√ 58. deve calcular o fatorial de 1 (matando mutante de caso base)
√ 59. deve dividir e lançar erro para divisão por zero (1 ms)
√ 60. deve lançar erro para raiz quadrada de número negativo (2 ms)
√ 61. deve lançar erro para fatorial de número negativo (1 ms)
√ .62 deve lançar erro para array vazio (1 ms)
√ 63. deve calcular a mediana de um array par (matando mutante NoCoverage)
Test Suites: 1 passed, 1 total
Tests: 92 passed, 92 total
Snapshots: 0 total
Time: 0.743 s, estimated 7 s
Ran all test suites.