



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Graduação em Engenharia de Software

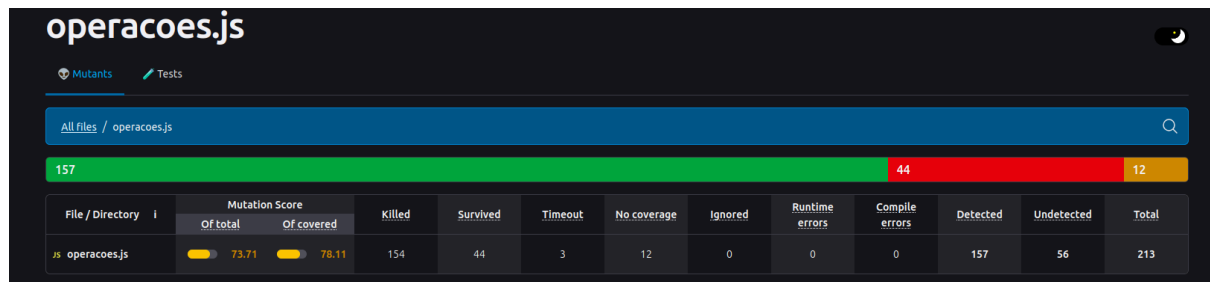
Pedro Henrique Dias Camara

Trabalho de Mutação

Belo Horizonte
2025

1. Análise Inicial

Ao analisar o código e seus casos de teste, a cobertura dos testes foi de 85.41%. Além disso, a pontuação de mutantes foi 73.71. A discrepância entre os valores pode ser explicada pela existência de mutações possíveis em partes cobertas pelos testes presentes que levam os testes à falhar.



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

2. Análise de mutantes críticos

```
42 function arredondar(n) { return Math.round(n); }
43 - function isPar(n) { return n % 2 === 0; } ▼
+ function isPar(n) { return true; }
44 function isImpar(n) { return n % 2 !== 0; } ●●
```

O presente mutante ocorreu na função `isPar`, esse trocou a lógica interna da função por um `'return true'`. Esse mutante foi possível devido à falta de testes que cobrem essa função, assim, se a lógica interna da função estiver errada, os testes presentes não seriam capazes de detectar o erro.

```
7 function clamp(valor, min, max) {
8 - if (valor < min) return min; ▼●
+ if (valor <= min) return min;
9 if (valor > max) return max; ●●
10 return valor;
```

Esse mutante é em relação à função `clamp`, ele alterou para que a lógica interna verificasse se o valor era menor ou igual à `min`, em vez de apenas igual. Assim, era possível que valores iguais para `valor` e `min` retornassem um valor maior que `max`. Não havia um teste para esse caso, logo o mutante foi possível.

```

function isMaiorQue(a, b) { return a > b; } ●●
- function isMenorQue(a, b) { return a < b; } ▼●
+ function isMenorQue(a, b) { return a <= b; }
function isEqual(a, b) { return a === b; } ●
function medianaArray(numeros) {

```

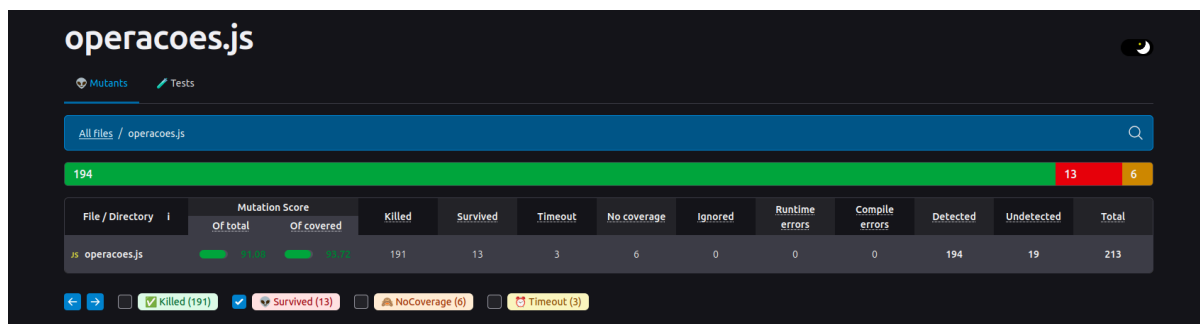
O mutante em questão acontece na função `isMenorQue`, e ele altera a comparação de menor para menor ou igual. Esse mutante foi possível durante a falta de testes que verificam o retorno da função se `a` e `b` forem o mesmo valor.

3. Soluções Implementadas

Para o primeiro mutante, a solução foi simplesmente adicionar um teste que verifica o funcionamento da lógica da função. Assim, não é mais possível que uma lógica interna quebrada não seja detectada pelos testes. Em seguida, o segundo mutante foi resolvido por meio de um teste que verifica o retorno de uma entrada em que o valor 'valor' e min sejam o mesmo e maior que o max, retornando um valor maior que max. Finalmente, para resolver o mutante da função `isMenorQue`, um teste em que os valores são iguais foi implementado. Assim garantindo o funcionamento da função no caso descrito.

4. Resultados finais

Após as alterações nos testes, a pontuação dos mutantes foi para 91.08, uma melhoria de mais de 17 pontos. Essa melhoria incluiu a implementação de outros testes não mencionados nas seções acima, cobrindo os casos de outros mutantes.



5. Conclusão

Tendo em vista as melhorias aos testes do código possibilitados pela ferramenta Stryker, é evidente o impacto do uso de mutantes para aprimorar casos de teste. Embora alguns testes implementados aparentem redundantes, a cobertura incompleta das funções pode causar problemas caso a lógica das funções sem testes for alterada, introduzindo novos comportamentos indesejados.