PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
BACHARELADO EM ENGENHARIA DE SOFTWARE

MATHEUS NOLASCO MIRANDA SOARES

MATRÍCULA: 688826

Análise de Eficácia de Testes com Teste de Mutação

Testes de Software

Análise Inicial

Cobertura de Código Inicial

A execução inicial da suíte de testes apresentou os seguintes resultados de cobertura de código: Statements: 85.41%, Branch: 58.82%, Funcs: 100%, Lines: 98.64%

Pontuação de Mutação Inicial

Ao executar o Stryker pela primeira vez, foram gerados 213 mutantes, com os seguintes resultados: Mutation Score (total): 73.71%, Mutation Score (covered): 78.11%, Mutantes mortos: 154, Mutantes sobreviventes: 44, Mutantes sem cobertura: 12, Timeout: 3, Erros: 0

Discrepância entre Cobertura e Qualidade

A diferença de 25 pontos percentuais entre cobertura de linhas e pontuação de mutação revela que executar código não garante testá-lo adequadamente. A cobertura mede execução, enquanto o teste de mutação mede validação. Os 44 mutantes sobreviventes indicam lacunas em: validação de condições, casos extremos, valores de retorno e tratamento de erros.

Análise de Mutantes Críticos

Mutante 1: Remoção da Ordenação na Função medianaArray

Mutação: Remoção do .sort((a, b) => a - b)

Teste original: medianaArray([1, 2, 3, 4, 5]) → esperado 3

Por que sobreviveu: O array de teste já estava ordenado, tornando a ordenação invisível ao teste.

Problema: O teste não validava o algoritmo completo, apenas um caso específico.

Mutante 2: Alteração da Condição de Validação em fatorial

Mutação: Condição de validação alterada para incluir zero

Teste original: `fatorial(4)` → esperado `24`

Por que sobreviveu: Não testava o caso base `n = 0`, onde 0! = 1.

Problema: Ausência de testes para casos extremos e valores de fronteira.

Mutante 3: Retorno Constante true na Função isPar

Mutação: Substituição da lógica por `return true`

Teste original: `isPar(10)` → esperado `true`

Por que sobreviveu: Apenas validava o caso positivo.

Problema: Teste unilateral não verifica o comportamento completo de funções booleanas.

Identificação dos Testes que Mataram os Mutantes

Mutante 1: Remoção da Ordenação em medianaArray

Testes que mataram este mutante:

```
test('NOVO - 47.3 deve calcular a mediana de um array ímpar não ordenado', () => { expect(medianaArray([5, 1, 3, 2, 4])).toBe(3);});

test('NOVO - 47.4 deve calcular a mediana de um array par não ordenado', () => { expect(medianaArray([4, 1, 3, 2])).toBe(2.5);});

test('NOVO - 47.5 deve calcular a mediana corretamente com números negativos e positivos não ordenados', () => { expect(medianaArray([-2, 4, 1, 3])).toBe(2); });
```

Por que esses testes mataram o mutante:
Os testes 47.3, 47.4 e 47.5 utilizam arrays não ordenados. Quando o mutante remove o `.sort()`, esses arrays permanecem desordenados e o cálculo da mediana retorna valores incorretos, fazendo os testes falharem.

Mutante 2: Alteração de n < 0 para n <= 0 em fatorial

Teste que matou este mutante:

```
test('NOVO - 8.2 deve retornar 1 para n = 0', () => {
expect(fatorial(0)).toBe(1); });
```

Por que esse teste matou o mutante:
O teste passa n = 0 e espera que a função retorne 1 (pois 0! = 1 por definição matemática).

- Código original: n < 0 é false para n = 0 → não lança erro → continua e retorna 1
- Mutante: n <= 0 é true para n = 0 → lança erro

Como o teste espera o valor `1` mas o mutante lança uma exceção, o teste falha e mata o mutante.

Mutante 3: Retorno Constante true em isPar

Teste que matou este mutante:

```
test('NOVO - 15.1 deve retornar false para um número ímpar', () => {
expect(isPar(3)).toBe(false); });
```

Por que esse teste matou o mutante:
O teste passa um número ímpar (3) e espera false como retorno. Quando o mutante substitui a lógica por return true, o teste recebe true em vez de false, falhando e matando o mutante.

Resultados Finais

Pontuação de Mutação Final

- Mutation Score: 73.71% → 97.65%
- Mutantes Mortos: 154 → 205
- Mutantes Sobreviventes: 44 → 5
- Sem Cobertura: 12 → 0

A suíte de testes passou de 73.71% para 97.65% de pontuação de mutação, representando uma melhoria de 23.94 pontos percentuais. Isso significa que a qualidade da suíte de testes foi significativamente aprimorada, eliminando 39 dos 44 mutantes sobreviventes originais e cobrindo todos os 12 mutantes que estavam sem cobertura.
A eliminação de 39 mutantes sobreviventes comprova a melhoria significativa da qualidade da suíte de testes.

Por Que Não Foi Possível Atingir 98%?

Os 5 mutantes restantes são mutantes equivalentes, ou seja, não alteram o comportamento do código:

- Mutantes em fatorial: O loop for (let i = 2; i <= n; i++) nunca executa para n = 0 ou n = 1, tornando if (n === 0 || n === 1) return 1 redundante.
- Mutante em produtoArray: O reduce com valor inicial 1 já retorna 1 para arrays vazios, tornando a verificação explícita desnecessária.

Esses mutantes indicam redundâncias no código. Para eliminá-los, seria necessário refatorar a implementação, o que estava fora do escopo do trabalho.

Conclusão dos Resultados

O teste de mutação revelou ser uma ferramenta indispensável para avaliar a verdadeira qualidade de testes. Este trabalho demonstrou que métricas tradicionais de cobertura contam apenas parte da história. A cobertura mede execução; o teste de mutação mede validação.

A discrepância entre 98.64% de cobertura e 73.71% de pontuação de mutação deixou uma lição clara: executar código não é testá-lo. É possível ter alta cobertura e testes frágeis que não detectam defeitos reais. O teste de mutação expõe essas fragilidades ao simular erros e verificar se os testes os identificam.

Durante este trabalho, ficou evidente que o teste de mutação funciona como um guia objetivo, apontando lacunas específicas como ausência de testes para casos negativos, extremos ou validações de algoritmos. Essa precisão elimina o aspecto subjetivo da criação de testes, fornecendo feedback concreto sobre onde a suíte falha.

Vale lembrar que o teste de mutação tem limitações. Mutantes equivalentes são inevitáveis em códigos com redundâncias, transformando a ferramenta em indicador de qualidade do próprio código de produção.

A melhoria de 73.71% para 97.65% comprova o valor prático da abordagem. Em ambientes reais, onde bugs têm custos elevados, o teste de mutação oferece confiança de que a suíte é uma proteção real contra regressões. Não substitui outras práticas, mas as complementa de forma crucial, garantindo que os testes façam o que prometem: proteger o software.