

# Teste e Inspeção de Software

Alyson Matheus Maruyama Nascimento - 8532269

Cristiano Di Maio Chiaramelli - 9293053

Felipe Tiago De Carli - 10525686

Rafael Gongora Bariccatti - 10892273

## **Projeto 2 - Teste de Mutação**



**Universidade de São Paulo - São Carlos**

# Introdução

Este documento tem como objetivo relatar o desenvolvimento do segundo projeto da disciplina SSC0721 - Teste e Inspeção de Software, ministrada pela professora Simone Senger de Souza.

O sistema a ser testado utilizando técnicas de testes de mutação corresponde ao Jogo da Vida, que foi implementado em linguagem Java e utilizado como objeto de estudo durante o primeiro projeto da disciplina.

O principal objetivo deste projeto é aplicar os conceitos de teste de mutação, utilizando a ferramenta de teste de mutação PIT (<https://pitest.org/>)

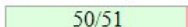

## Avaliação da qualidade do conjunto de casos de teste

Após executar a ferramenta PIT aplicando todos os operadores de mutação no código utilizado no Projeto 1, temos os seguintes resultados retornados pela ferramenta:

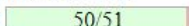
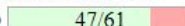
### Cobertura de Código:

#### Pit Test Coverage Report

##### Project Summary

| Number of Classes | Line Coverage                                                                           | Mutation Coverage                                                                        |
|-------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| 2                 | 98%  | 77%  |

##### Breakdown by Package

| Name                    | Number of Classes | Line Coverage                                                                           | Mutation Coverage                                                                         |
|-------------------------|-------------------|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <a href="#">default</a> | 2                 | 98%  | 77%  |

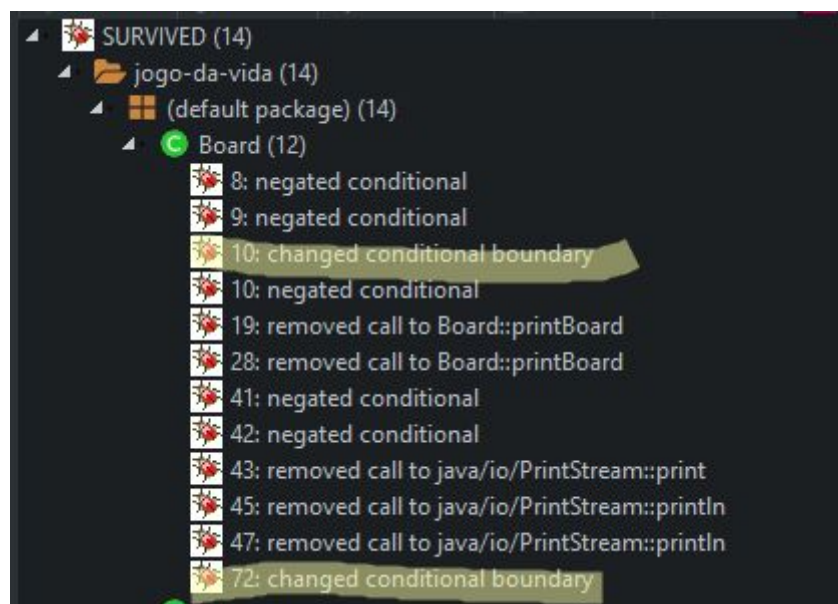
### Mutantes Vivos:

Tivemos um total de 14 mutantes vivos, como pode ser visto na figura:



## Aplicação do teste de mutação

Dentre os mutantes vivos acima, a grande maioria está relacionada a trechos de código utilizados apenas para imprimir as matrizes no terminal. Sendo assim, temos que dentre esses 14 mutantes vivos, somente os dois mutantes grifados na figura abaixo representam mutações no código relacionadas à lógica do jogo da vida:



O primeiro mutante vivo (linha 10 da classe Board) representa a mutação no construtor da classe utilizado para sortear os valores de cada célula do tabuleiro. Como este código é um gerador aleatório de números entre 0 e 1, este tipo de mutação pode ser classificada como mutação equivalente.

Podemos também citar como mutantes equivalentes os da linha 28 e 43, visto que são relacionados à métodos de somente impressão no terminal (respectivamente *printBoard()* e *println()*), e consequentemente a remoção ou

alteração dessas chamadas não altera a lógica do programa.

```
// Board.java
public Board() {
    for(int i = 0; i < this.size; i++) {
        for(int j = 0; j < this.size; j++) {
            // A linha abaixo foi alterada
            this.board[i][j] = Math.random() < 0.5 ? 0 : 1;
        }
    }
}
```

O último mutante, por sua vez, indicado na linha 72, representa o código responsável por processar cada uma das células do tabuleiro e retornar seu novo valor para a próxima iteração. Este mutante não pôde ser morto pela forma que o código estava estruturado, porém isso não representou uma falha de lógica no programa.

O programa foi reestruturado a fim de matar este mutante, e as diferenças da versão antiga para a nova estão apontadas logo abaixo:

#### Versão Antiga:

```
// Board.java
// .processCell() method
// ...
1 if(numberOfNeighbours == 2) return currentValue;
2
3 if(currentValue == 1) { // Alive Cells
4     if(numberOfNeighbours < 2 || numberOfNeighbours > 3) {
5         return 0;
6     }
7 }
// ...
```

#### Versão Refatorada:

```
// Board.java
// .processCell() method
```

```
// ...
1 if(currentValue == 1) { // Alive Cells
2   if(numberOfNeighbours < 2 || numberOfNeighbours > 3) {
3     return 0;
4   }
5 } else { // Dead Cells
6   if(numberOfNeighbours == 3) return 1;
7 }
```

Podemos notar que a primeira condicional *if* (linha 1) do código antigo foi removido. Isso foi necessário porque o teste de mutação alterava a condição `if(numberOfNeighbours < 2)`, da linha 4, para `if(numberOfNeighbours <= 2)`, e consequentemente a condicional que foi removida não permitia que a execução do código alcançasse o mutante.

## Escores de mutação

O primeiro teste de mutação houveram 61 mutantes, em que 4 são mutantes equivalentes e 47 foram mortos com sucesso.

O escore do teste de mutação é dado pela fórmula:

$$Escore = \frac{mutantesMortos}{totalDeMutantes - mutantesEquivalentes}$$

Logo, o escore do primeiro teste foi de **0.825**.

No segundo teste houve a correção do mutante sobrevivente da linha 72. O escore do segundo teste foi de **0.842**.

## Análise do teste de mutação

Apesar do teste de mutação não ter detectado nenhum erro lógico no programa, é visível sua utilidade em grandes aplicações, onde não há certeza se todos os casos de teste estão abrangendo corretamente o código escrito.

O sistema PIT também apresentou uma grande eficiência e rapidez no uso. A simplicidade de aplicar o sistema é um grande incentivo para adotá-lo em larga escala.

## Questionamento

- **Se os mutantes que têm comportamento diferente do programa original são mortos, como eu sei que o mutante que morreu não é a versão correta do programa? Como resolver isso?**

Se o mutante morreu significa que existe um ou mais caso de teste que falhou por conta do código alterado. Então, se um mutante morto for uma versão correta do programa, existe um erro nos testes escritos, pois se os testes estivessem certos, o mutante com a versão correta do programa deveria sobreviver.