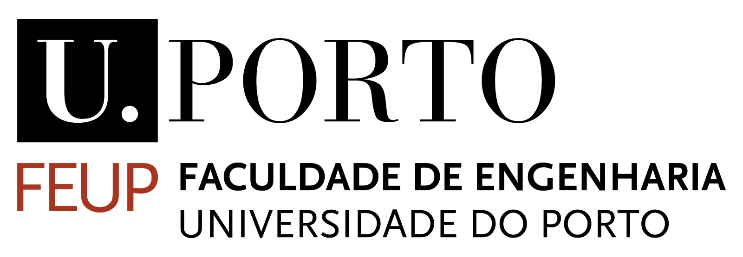
****

Teste, Verificação e Validação de Software



**Correção do Exercício Prático**

1 GDC

a) Através da ferramenta *Pitclipse* identifique os mutantes presentes na função ***GDC*** da classe ***Algorithm*.**

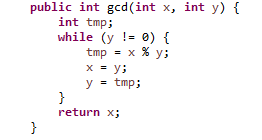
****

Figura 1 - Função gdc

b) Encontre a melhor solução com o método de eliminação de mutantes.

c) Teste a sua solução com a ferramenta *Pitclipse* e verifique se é possível eliminar os mutantes.

2 Min

a) Através da ferramenta *Pitclipse* identifique os mutantes presentes na função ***Min*** da classe ***Algorithm*.**

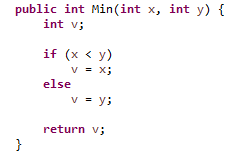


Figura 2 - Função min

b) Encontre a melhor solução com o método de eliminação de mutantes e verifique com a ferramenta *Pitclipse* se é possível eliminar os mutantes.

c) O que conclui dos mutantes da função ***Min*** da classe ***Algorithm*?**

3 NumZero e NegateArray

a) Através da ferramenta *Pitclipse* identifique os mutantes presentes na função ***NumZero*** e ***NegateArray*** da classe ***Algorithm*** e encontre possíveis soluções de teste capazes de os eliminar.

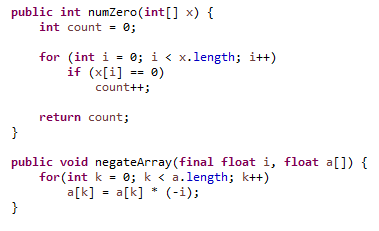


Figura 3 - Função numZero e negateArray

4 IsLessThanThree

a) A função ***IsLessThanThree*** já tem um teste associado, porém mesmo com um ***assert*** aplicado e estando correto, os mutantes não morreram.

Por favor corrija a situação.



Figura 4 - Função isLessThanThree

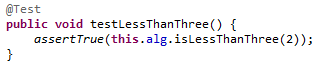


Figura 5 - Teste da função

Mutantes Vivos

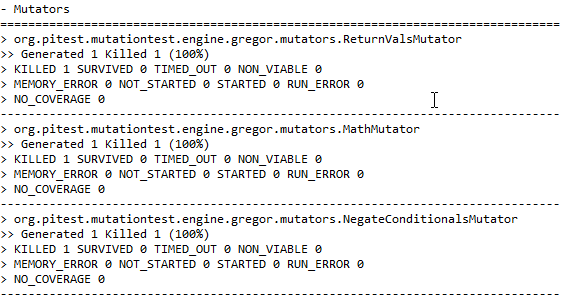


Bom trabalho!

1 GDC - correção

a)

* Negate Conditionals Mutator, na linha 6 (!= 🡪 ==)
* Math Mutator, na linha 7 (% 🡪 \*)
* Return Values Mutator, na linha 11 (return value is 0)



b)

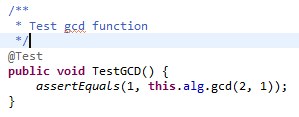
O mutante Return Values Mutator pode ser eliminado obrigando ao *coverage* daquela linha. Resta apenas analisar os outros mutantes.

Possível solução:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **x** | **y** | **gcd** | **!= --> ==** | **% 🡪 \*** |
| **T1** | 0 | 0 | 0 | loop | 0 |
| **T2** | 0 | 1 | 1 | 0 | 1 |
| **T3** | 1 | 0 | 1 | - | 1 |
| **T3** | 2 | 1 | 1 | 2 | 2 |

A solução x=2 e y=1 é um bom caso de teste.

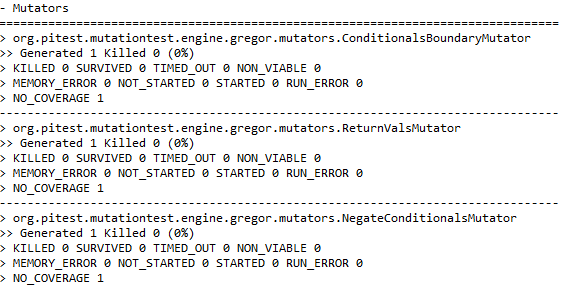
c)



2 Min - correção

a)

* Conditional Boundary Mutator, linha 17 ( < 🡪 <=)
* Negate Conditionals Mutator, linha 17 (< 🡪 >=)
* Return Values Mutator, na linha 22 (return value is 0)



b)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **x** | **y** | **min** | **< 🡪 <=** | **< 🡪 >=** |
| **T1** | 0 | 0 | 0 | 0 | 0 |
| **T2** | 0 | 1 | 0 | 0 | 1 |
| **T3** | 1 | 0 | 0 | 0 | 1 |

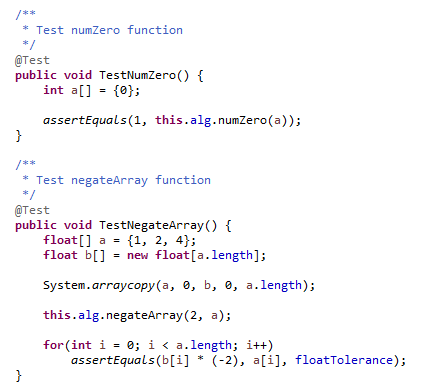
Neste caso não existe nenhum teste que consiga matar o mutante <= visto que o resultado deste teste mutado é igual ao resultado do teste original.

c)

O mutante que não se consegue matar na função é um **mutante funcionalmente equivalente**.

3 NumZero e NegateArray - correção

Neste exercício não é pedido que se faça a resolução pelo método de eliminação de mutantes pelo que bastaria apresentar testes unitários que matassem os mutantes.



4 IsLessThanThree - correção

Neste exercício bastaria acrescentar um teste que consiga passar no teste mutante mas não no teste original. Por exemplo:

