Pedro William Andrade Carvalho

Teste de Software

Atividade 2

1. Introdução

Este documento descreve a implementação e os testes de mutantes no código da classe MutLoop, que inclui métodos que manipulam listas através de loops for e while. O objetivo é garantir que os testes sejam robustos o suficiente para detectar mutações (modificações maliciosas ou incorretas) no código, uma prática conhecida como "teste de mutação".

2. Classe MutLoop

A classe MutLoop contém três métodos principais que operam sobre intervalos de números inteiros:

- **simpleFor(self, x)**: Cria uma lista contendo os números de 0 até x-1 usando um loop for.
- **inlineFor(self, x)**: Cria uma lista contendo os números de 1 até x, gerados através de uma list comprehension com um loop for embutido.
- **simpleWhile(self, x)**: Cria uma lista contendo os números de 0 até x-1 usando um loop while.

3. Mutantes Gerados

Vários mutantes foram introduzidos no código da classe MutLoop para testar a eficácia dos testes unitários. Cada mutante representa uma modificação potencialmente errônea do código, e a meta é que os testes identifiquem essas mudanças.

Report antes de efetuar medidas para prevenir mutações.

3.1. Mutante 1

Local: Método simpleFor

```
- input_list = range(0, x)
+ input_list = range(1, x)
```

```
def simpleFor(self, x):
    input_list = range(1, x)
    output_list = []
    for i in input_list:
        output_list.append(i)
    return output_list
```

Descrição: Este mutante altera o intervalo do loop for para começar de 1 em vez de 0. Isso significa que o número 0 será excluído da lista gerada, o que altera o comportamento esperado, especialmente quando x = 1.

3.2. Mutante 4

Local: Método inlineFor

```
- input_list = range(0, x)
+ input_list = range(1, x)
```

```
def inlineFor(self, x):
    input_list = range(1, x)
    output_list = [y + 1 for y in input_list]
    return output_list
```

Descrição: Similar ao Mutante 1, mas aplicado ao método inlineFor. Isso faz com que a lista gerada exclua 0 e comece de 1, o que muda o resultado final da list comprehension.

3.3. Mutante 6

Local: Método inlineFor

```
- output_list = [y + 1 for y in input_list]
+ output_list = [y - 1 for y in input_list]
```

```
def inlineFor(self, x):
    input_list = range(0, x)
    output_list = [y - 1 for y in input_list]
    return output_list
```

Descrição: Este mutante altera a operação dentro da list comprehension, diminuindo cada valor em 1 em vez de incrementá-lo. Isso significa que, em vez de gerar [1, 2, 3,...], o método agora gera [-1, 0, 1,...].

3.4. Mutante 7

Local: Método inlineFor

```
- output_list = [y + 1 for y in input_list]
```

```
+ output_list = [y + 2 for y in input_list]
```

```
def inlineFor(self, x):
    input_list = range(0, x)
    output_list = [y + 2 for y in input_list]
    return output_list
```

Descrição: Este mutante altera a operação dentro da list comprehension para incrementar cada valor em 2 em vez de 1. Como resultado, a lista gerada é [2, 3, 4,...] em vez de [1, 2, 3,...].

3.5. Mutante 9

Local: Método simpleWhile

```
- i = 0
+ i = 1
```

```
def simpleWhile(self, x):
    i = 1
    output_list = []
    while i < x:
        output_list.append(i)
        i += 1
    return output_list
```

Descrição: Esse mutante modifica o valor inicial de i no loop while para 1 em vez de 0. Isso significa que a lista gerada começará de 1, omitindo 0, o que afeta a sequência de números esperada.

3.6. Mutante 13

Local: Método simpleWhile

```
- i = 1
+ i = 2
```

```
def simpleWhile(self, x):
    i = 0
    output_list = []
    while i < x:
        output_list.append(i)
        i += 2
    return output_list
```

Descrição: Esse mutante altera o valor de i para 2 dentro do loop while. A lista gerada por este método incluirá apenas números pares, o que distorce a sequência linear esperada.

3.7. Mutante 14

Local: Método simpleWhile

```
- i = 1
+ i = None
```

```
def simpleWhile(self, x):
    i = 0
    output_list = []
    while i < x:
        output_list.append(i)
        i = None
    return output_list
```

Descrição: Esse mutante define i como None dentro do loop while. Isso causará um erro de tipo (TypeError) ao tentar comparar None com um número em while i < x:. Este mutante testa se os testes conseguem detectar o erro de execução.

4. Testes Implementados

Para garantir a detecção dos mutantes, uma série de testes foram implementados. Estes testes verificam o comportamento correto dos métodos da classe MutLoop.

4.1. Teste de simpleFor

- **Teste 1:** Verifica se simpleFor(0) retorna uma lista vazia ([]).
- **Teste 2:** Verifica se simpleFor(1) retorna [0], matando o mutante que começa o range em 1.
- **Teste 3:** Verifica se simpleFor(2) retorna [0, 1], garantindo que todos os valores até x-1 sejam incluídos.

4.2. Teste de inlineFor

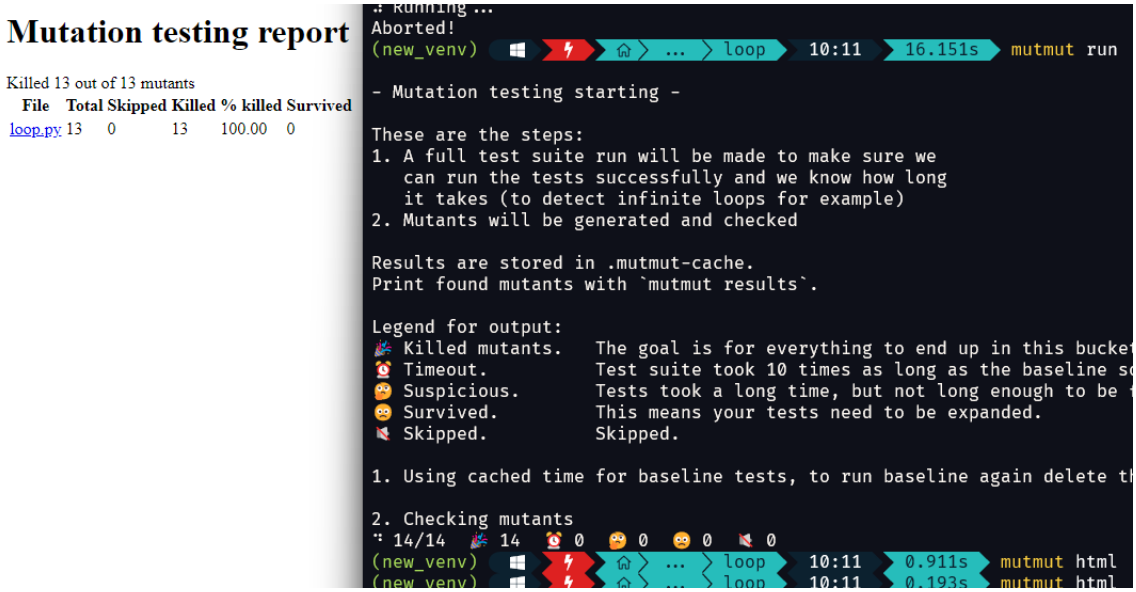
- **Teste 4:** Verifica se inlineFor(0) retorna uma lista vazia ([]).
- **Teste 5:** Verifica se inlineFor(1) retorna [1], matando o mutante que começa o range em 1.
- **Teste 6:** Verifica se inlineFor(2) retorna [1, 2], garantindo que a modificação do incremento seja detectada.

4.3. Teste de simpleWhile

- **Teste 7:** Verifica se simpleWhile(0) retorna uma lista vazia ([]).
- **Teste 8:** Verifica se simpleWhile(1) retorna [0], matando o mutante que começa o loop em 1.

- **Teste 9:** Verifica se simpleWhile(2) retorna [0, 1], detectando a modificação que incrementa i em 2.
- **Teste 10:** Verifica se uma exceção TypeError é levantada se i for definido como None.

5. Conclusão



Mutation testing report

Killed 13 out of 13 mutants

File	Total	Skipped	Killed	% killed	Survived
loop.py	13	0	13	100.00	0

```
Aborted!
(new_venv)  ...  loop  10:11  16.151s  mutmut run

- Mutation testing starting -

These are the steps:
1. A full test suite run will be made to make sure we
   can run the tests successfully and we know how long
   it takes (to detect infinite loops for example)
2. Mutants will be generated and checked

Results are stored in .mutmut-cache.
Print found mutants with `mutmut results`.

Legend for output:
🎉 Killed mutants.   The goal is for everything to end up in this bucke
⏰ Timeout.          Test suite took 10 times as long as the baseline s
🤔 Suspicious.       Tests took a long time, but not long enough to be
🙁 Survived.         This means your tests need to be expanded.
🔇 Skipped.          Skipped.

1. Using cached time for baseline tests, to run baseline again delete t

2. Checking mutants
⠹ 14/14  🎉 14  ⏰ 0  🤔 0  🙁 0  🔇 0
(new_venv)  ...  loop  10:11  0.911s  mutmut html
(new_venv)  ...  loop  10:11  0.193s  mutmut html
```

Os testes de mutação são uma técnica eficaz para garantir que os testes unitários cubram adequadamente as possíveis falhas ou modificações indesejadas no código. Neste exercício, introduzimos mutantes nos métodos da classe MutLoop e verificamos a capacidade dos testes de identificá-los. A ativação das flags KILL_ZERO_FOR, KILL_ONE_FOR, KILL_INLINE, KILL_ZERO_WHILE, e KILL_ONE_WHILE foi utilizada para garantir que os mutantes fossem detectados e mortos, confirmando a robustez dos testes implementados.