

SWE30009: Software Testing and Reliability

Project Report

Name: SWH00420 Tran Quoc Dung

Student ID: 103803891

Assignment Solutions:

Task 1: Understanding of the *Metamorphic testing* methodology

- **Metamorphic testing** is a software-testing technique, used for generating test cases, and alleviating the test oracle problem by evaluating relations between inputs & outputs.
- **Test oracle** is a mechanism or procedure, which appears to compare a program's outputs with the test cases, to evaluate the accuracy of the program.
- A system is known as an **untestable system** when the outputs of the program cannot be manually verified as correct or not.
- **Motivation and intuition:** Even though we do not know the accuracy of the output of any particular input, we might grasp the relation between related inputs and their outputs.
- **Metamorphic relations** are compulsory properties of the program's algorithm to be implemented in these circumstances, which includes multiple related inputs and their outputs. When metamorphic relations are identified, we can select an input and utilize metamorphic relations to create new inputs, which can be used for generating test cases.
- **The process of metamorphic testing (4 main steps):**
 - +) Defining and executing test cases with some test case selection strategies.
 - +) Identify the properties of the problem, thereby being able to identify the metamorphic relations.
 - +) Creating and executing new test cases from the original test case, based on the identified metamorphic relations.
 - +) Verifying the metamorphic relations based on the computed outputs.

- **Applications:** This methodology can be used to test numerical algorithms, machine learning models in software testing, or it can be utilized for testing the signal of processing system.

For instance, by using **metamorphic testing** to **test an ascended sorting algorithm**, the metamorphic relation in this circumstance can indicate that if the arrangement of a correct output array is reversed, it should turn into a list of numbers with descending order. If the new output array is truly displayed in descending order, the original algorithm is accurate.

Task 2: Understanding of the *Random testing* methodology

- **Random testing** basically is a testing technique, where the program is tested by selecting random, independent test cases from the input domain.
- **Intuition:** Although this testing methodology is ineffective since it does not use comprehensive information such as the program's, random testing is intuitively simple, and allows statistical quantitative estimation of the reliability of the program.
- **Distribution profiles (2 main approaches):**
 - +) Uniform distributions: Each of the inputs has the equal probability of being selected.
 - +) Operational distributions: Selection probability follows the probability of being used.
- **The process of random testing:**
 - +) Random inputs are identified as valid in the Input domain.
 - +) Tested input is randomly, independently selected from the test domain.
 - +) Test cases are executed with those selected inputs.
 - +) Computed outputs are compared with the expected outputs to evaluate the accuracy of the program.
- **Application:** Random testing is utilized when the errors in a program cannot be recognized, also to check the execution and dependability of the program.

For instance, with a simple algorithm such as $x = 6x - 3x - 2x$, a random number such as 3 can be applied to the algorithm as follows: $6*3 - 3*3 - 2*3 = 3$, which indicates correct algorithm. Nevertheless, there are not enough comprehensive explanations about why the program is efficient since we only know that random inputs can totally be applied to this situation.

Task 3: Testing a sorting program

Program Description

- Input: A list which may contain duplicated integers.
- Output: A list contains the elements of input list in ascending order & without duplicates.

Metamorphic testing (3 metamorphic relations – 3MRs)

- MR 1 (Duplications):

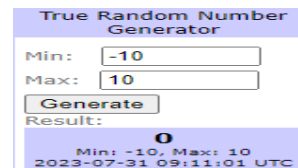
Definitions:

- SI_1 illustrates the source input list of the program.
- FI_1 illustrates the follow-up input list of the program.
- SO_1 illustrates the source output list of the program.
- FO_1 illustrates the follow-up output list of the program.

Metamorphic relation:

If FI_1 is constructed from SI_1 by adding more duplicated elements, then $FO_1 = SO_1$, since the program excludes all the duplicated integers.

Example:



SI_1 (source input): [0, 6, 9, -2, 4, -3, -8]

FI_1 (follow-up input): [0, 6, 6, 9, 9, 9, -2, 4, -3, -3, -8] (adding duplicated numbers)

SO_1 (source output): [-8, -3, -2, 0, 4, 6, 9] (outputted from SI_1)

FO_1 (follow-up output): [-8, -3, -2, 0, 4, 6, 9] (outputted from FI_1)

- SI_1 is generated with random numbers on random.org, and FI_1 is constructing from SI_1 by adding more numbers as duplications such as 6, 9, ...
 - According to the above testing, $FO_1 = SO_1 \Rightarrow$ The program truly excludes the duplicated elements.
- \Rightarrow Successful

- MR 2 (Permutation):

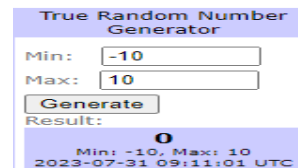
Definitions:

- SI_2 illustrates the source input list of the program.
- FI_2 illustrates the follow-up input list of the program.
- SO_2 illustrates the source output list of the program.
- FO_2 illustrates the follow-up output list of the program.

Metamorphic relation:

If FI_2 is constructed from SI_2 by arranging the integers in SI_2 randomly, then $FO_2 = SO_2$, since the program will re-arrange all the elements into ascended order.

Example:



SI_2 (source input): [0, 6, 9, -2, 4, -3, -8]

FI_2 (follow-up input): [-3, 9, 0, 4, -8, 6, -2] (re-arranging the orders)

SO_2 (source output): [-8, -3, -2, 0, 4, 6, 9] (outputted from SI_2)

FO_2 (follow-up output): [-8, -3, -2, 0, 4, 6, 9] (outputted from FI_2)

- SI_2 is generated with random numbers on random.org, and FI_2 is constructing from SI_2 by re-arranging all the orders of the elements.
 - According to the above testing, $FO_2 = SO_2 \Rightarrow$ The program truly outputs in ascending order.
- \Rightarrow Successful

- MR 3 (Reverse):

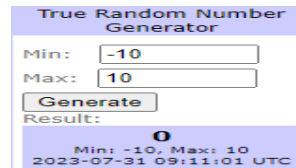
Definitions:

- SI_3 illustrates the source input list of the program.
- SO_3 illustrates the source output list of the program.
- FO_3 illustrates the follow-up output list of the program.

Metamorphic relation:

FO_3 (follow-up *output*) is constructed from SO_3 (source *output*) by reversing the order of every element, and if FO_3 contains integers in descended order, the program functions correctly.

Example:



SI_3 (source input): [0, 6, 9, -2, 4, -3, -8]

SO_3 (source output): [-8, -3, -2, 0, 4, 6, 9]

FO_3 (follow-up output): [9, 6, 4, 0, -2, -3, -8] (outputted from SO_3)

- SI_3 is generated with random numbers on random.org, and SO_3 is the output list.
 - According to the above testing, FO_3 contains the integers in descending order from 9 to 8.
- ⇒ The source output SO_3 is arranged in ascending order.
- ⇒ Successful

Task 4: Testing a program of my choice

- **Program:**

For my option of real-world program, I chose a simple program written in Python called *The_Operator.py* :

Input: A list of integers (values from -10 to 10)

Output: The **sum** of *first* and *last* elements of the input list

```
The_Operator.py x main.py
C: > Users > ZungBii > Desktop > Swinburne > SWE30009 Software Testing and Reliability > The_Operator.py > the_Operator
1 def the_Operator(a):
2     sum = a[0] + a[len(a)-1]
3     print (sum)
```

- **Metamorphic relations (2MRs) :**

Metamorphic relation 1 (MR1):

Source input: [0, 6, 9, -2, 4, -3, -8]

Follow-up input: [0, 6, 6, 9, 9, 9, -2, 4, -3, -3, -8]

```
The_Operator.py x main.py x
C: > Users > ZungBii > Desktop > Swinburne > SWE30009 Software Testing and Reliability > main.py > ...
1 from The_Operator import the_Operator
2
3 a = [0, 6, 9, -2, 4, -3, -8]
4
5 the_Operator(a)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Microsoft Windows [Version 10.0.22621.2870]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ZungBii>C:\Users\ZungBii\AppData\Local\Programs\Python\Python310\python.exe "c:/Users/ZungBii/Desktop/Swinburne/SWE30009 Software Testing and Reliability/main.py"
-8
```

The program's original output (Source output = -8).

```
The_Operator.py x main.py x
C: > Users > ZungBii > Desktop > Swinburne > SWE30009 Software Testing and Reliability > main.py > ...
1 from The_Operator import the_Operator
2
3 a = [0, 6, 6, 9, 9, 9, -2, 4, -3, -3, -8]
4
5 the_Operator(a)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Microsoft Windows [Version 10.0.22621.2870]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ZungBii>C:\Users\ZungBii\AppData\Local\Programs\Python\Python310\python.exe "c:/Users/ZungBii/Desktop/Swinburne/SWE30009 Software Testing and Reliability/main.py"
-8

C:\Users\ZungBii>C:\Users\ZungBii\AppData\Local\Programs\Python\Python310\python.exe "c:/Users/ZungBii/Desktop/Swinburne/SWE30009 Software Testing and Reliability/main.py"
-8
```

The same output after using follow-up input (Follow-up output = -8).

Result:

Test	SI (source input)	FI (follow-up input)	SO (source output)	FO (follow-up output)
T1	[0, 6, 9, -2, 4, -3, -8]	[0, 6, 6, 9, 9, 9, -2, 4, -3, -3, -8]	-8	-8

+) If FI is constructed from SI by adding more duplicated elements, then FO = SO, since the order of first and last elements of the list remains, and so is the value.

+) FO = SO = -8

⇒ The program function correctly.

+) **Mutation analysis:**

Mutants: M1, M2

M1 (outputs the difference between the first and last element) => Output: 8

M2 (outputs the product of the first and last element) => Output: 0

Since output of T1 is -8, totally different from above results:

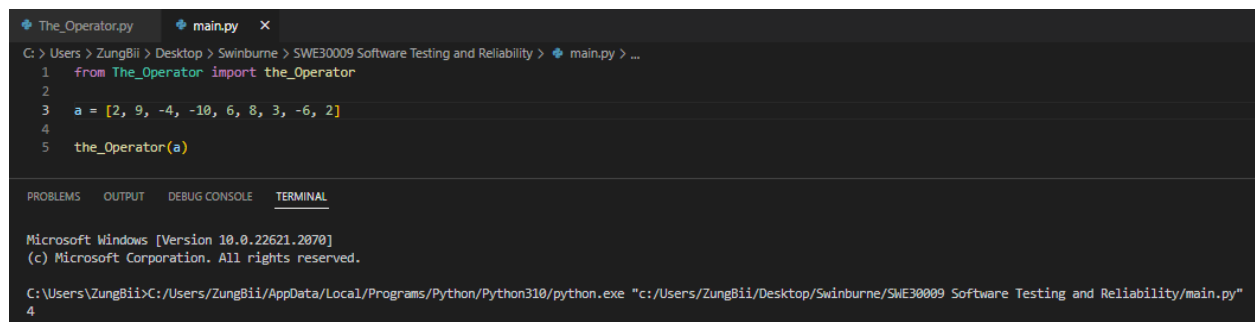
⇒ Test case T1 kills both mutants M1 & M2

⇒ ***Mutation score (MS) = 2/2 (or 1).***

Metamorphic relation 2 (MR2):

Source input: [2, 9, -4, -10, 6, 8, 3, -6, 2]

Follow-up input: [-10, 3, 2, -4, 2, -6, 3, 6, 9]



```
The_Operator.py  main.py  X
C:\Users\ZungBii\Desktop\Swimburne\SWE30009 Software Testing and Reliability > main.py > ...
1  from The_Operator import the_operator
2
3  a = [2, 9, -4, -10, 6, 8, 3, -6, 2]
4
5  the_operator(a)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Microsoft Windows [Version 10.0.22621.2870]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ZungBii>C:\Users\ZungBii\AppData\Local\Programs\Python\Python310\python.exe "c:\Users\ZungBii\Desktop\Swimburne\SWE30009 Software Testing and Reliability/main.py"
4
```

The program's original output (Source output = 4).

```

The_Operator.py  main.py x
C:\> Users > ZungBii > Desktop > Swinburne > SWE30009 Software Testing and Reliability > main.py > ...
1  from The_Operator import the_Operator
2
3  a = [-10, 3, 2, -4, 2, -6, 3, 6, 9]
4
5  the_Operator(a)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2070]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ZungBii>C:/Users/ZungBii/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/ZungBii/Desktop/Swinburne/SWE30009 Software Testing and Reliability/main.py"
4

C:\Users\ZungBii>C:/Users/ZungBii/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/ZungBii/Desktop/Swinburne/SWE30009 Software Testing and Reliability/main.py"
-1

```

The program's new output (Follow-up output = -1).

Result:

Test	SI (source input)	FI (follow-up input)	SO (source output)	FO (follow-up output)
T2	[2, 9, -4, -10, 6, 8, 3, -6, 2]	[-10, 3, 2, -4, 2, -6, 3, 6, 9]	4	-1

+) If FI is constructed from SI by re-arranging the orders of all elements, then $FO \neq SO$, since the value of the first and last element change.

+) $FO \neq SO$ ($4 \neq -1$)

⇒ The program function correctly.

+) **Mutation analysis:**

Mutants: M1, M2

M1 (outputs the difference between the first and last element) ⇒ Output: 0

M2 (outputs the product of the first and last element) ⇒ Output: 4

Since the source output of T2 is 4:

⇒ Test case T2 only kills 1 mutant M1, not M2.

⇒ **Mutation score = 1/2 (or 0,5).**

REPORT COMPLETED!