



SWE30009

Software Testing and Reliability

Assignment 1 Report

Student Name: Lau Ngoc Quyen

Student ID: 104198996

Task 1: Designing The Test Cases

Consider the following program:

```
Input A, B // A and B are real variables
A = A - B
C = A * 2
Output C // C is a real variable
```

Original Program

Requirement:

Design test cases to detect any possible incorrect use of arithmetic operators in the original program.

Original Program: $C = (A - B) * 2$

Incorrect Programs:

- Replace “ - ” with “ + ”, “ * ”, “ / ”
- Replace “ * ” with “ + ”, “ - ”, “ / ”
- Combination of the above

Test Case Design:

- Various test cases with positive, negative, and zero values for A and B.

- Compare original program results with incorrect programs.

Example Test Cases:

Test Case 1: (A = 3, B = 1)

Original Program: $C = (3 - 1) * 2 = 4$

Incorrect Programs:

- $C = (3 + 1) * 2 = 8$
- $C = (3 * 1) * 2 = 6$
- $C = (3 / 1) * 2 = 6$
- $C = (3 - 1) + 2 = 4$ (Ambiguous)

Analysis: **Ambiguous** test case.

Test Case 2: (A = -2, B = 2)

Original Program: $C = (-2 - 2) * 2 = -8$

Incorrect Programs:

- $C = (-2 + 2) * 2 = 0$
- $C = (-2 * 2) * 2 = -8$ (Ambiguous)
- $C = (-2 / 2) * 2 = -2$

Analysis: **Ambiguous** test case.

Test Case 3: (A = 0, B = 5)

Original Program: $C = (0 - 5) * 2 = -10$

Incorrect Programs:

- " $C = (0 + 5) * 2 = 10$ "
- " $C = (0 * 5) * 2 = 0$ "
- " $C = (0 / 5) * 2 = 0$ "
- " $C = (0 - 5) + 2 = -3$ "
- " $C = (0 - 5) - 2 = -7$ "
- " $C = (0 - 5) / 2 = -2.5$ "
- " $C = (0 + 5) + 2 = 7$ "
- " $C = (0 * 5) + 2 = 2$ "
- " $C = (0 / 5) + 2 = 2$ "
- " $C = (0 + 5) - 2 = 3$ "
- " $C = (0 * 5) - 2 = -2$ "
- " $C = (0 / 5) - 2 = -2$ "
- " $C = (0 + 5) / 2 = 2.5$ "
- " $C = (0 * 5) / 2 = 0$ "
- " $C = (0 / 5) / 2 = 0$ "

Analysis: **Non-ambiguous** test case.

Test Case 4: (A = 10, B = -10)

Original Program: $C = (10 - (-10)) * 2 = 40$

Incorrect Programs:

- " $C = (10 + (-10)) * 2 = 0$ "
- " $C = (10 * (-10)) * 2 = -200$ "
- " $C = (10 / (-10)) * 2 = -2$ "
- " $C = (10 - (-10)) + 2 = 22$ "
- " $C = (10 - (-10)) - 2 = 18$ "
- " $C = (10 - (-10)) / 2 = 10$ "
- " $C = (10 + (-10)) + 2 = 2$ "
- " $C = (10 * (-10)) + 2 = -98$ "
- " $C = (10 / (-10)) + 2 = 0$ "
- " $C = (10 + (-10)) - 2 = -2$ "
- " $C = (10 * (-10)) - 2 = -202$ "
- " $C = (10 / (-10)) - 2 = -3$ "
- " $C = (10 + (-10)) / 2 = 0$ "
- " $C = (10 * (-10)) / 2 = -50$ "
- " $C = (10 / (-10)) / 2 = -0.5$ "

Analysis: **Non-ambiguous** test case.

Task 2: Evaluation of Test Case (A = 3, B = 1)

Requirement: Use test case (A = 3, B = 1) to test the program.

Original Program Execution:

Input: (A = 3, B = 1)

Operations:

$A = 3 - 1 \Rightarrow A = 2$

$C = A * 2 \Rightarrow C = 4$

Original Output: C = 4

Testing With Incorrect Operators:

Replacing " - " with " + " :

$A = 3 + 1 \Rightarrow A = 4$

$C = A * 2 \Rightarrow C = 8$ (Different)

Replacing " * " with " + ":

$A = 3 - 1 \Rightarrow A = 2$

$C = A + 2 \Rightarrow C = 4$ (Ambiguous)

Analysis:

The test case (A = 3, B = 1) is effective in detecting most incorrect uses of operators but is ambiguous when " * " is replaced with " + ".

Task 3: Concrete Test Cases

(Based on previous design in Task 1)

Ambiguous Test Cases:

(A = 3, B = 1)

(A = -2, B = 2)

Non-Ambiguous Test Cases:

(A = 0, B = 5)

(A = 10, B = -10)

Conclusion:

- Test cases (A = 0, B = 5) and (A = 10, B = -10) are most effective for detecting incorrect use of arithmetic operators.

Task 4: Detecting Possible Ambiguity “A” Values Test Cases For B = 1

Automatically Detecting Python Program:

```
1 def original_program(A, B):
2     A = A - B
3     C = A * 2
4     return C
5
6 def is_ambiguous(A, B):
7     orig_result = original_program(A, B)
8     # Generate incorrect results by applying various operations to A and B
9     incorrect_results = [(A + B) * 2, (A * B) * 2, (A / B) * 2,
10                          A * B + 2, A - 2, (A + B) / 2,
11                          (A + B) - 2, (A * B) + 2, (A / B) + 2,
12                          (A + B) / 2, (A * B) - 2, (A / B) - 2,
13                          (A + B) * 2, (A * B) / 2, (A / B) / 2]
14     # Check if any of the incorrect results match the original result
15     return any(orig_result == result for result in incorrect_results)
16
17 # Given B value
18 B = 1
19 # Initialize a list to store ambiguous A values
20 ambiguous_A_values = []
21
22 # Iterate through a range of A values from -100 to 100
23 for A in range(-100, 101):
24     # Check if the current A value is ambiguous for the given B value
25     if is_ambiguous(A, B):
26         # Append the current value to the list of ambiguous A values
27         ambiguous_A_values.append(A)
28
29 # Print the ambiguous A values for B=1
30 print("Ambiguous A values for B=1 are:", ambiguous_A_values)
```

Code Execution:

```
1 def original_program(A, B):
2     A = A - B
3     C = A * 2
4     return C
5
6 def is_ambiguous(A, B):
7     orig_result = original_program(A, B)
8     # Generate incorrect results by applying various operations to A and B
9     incorrect_results = [(A + B) * 2, (A * B) * 2, (A / B) * 2,
10                          A * B + 2, A - 2, (A + B) / 2,
11                          (A + B) - 2, (A * B) + 2, (A / B) + 2,
12                          (A + B) / 2, (A * B) - 2, (A / B) - 2,
13                          (A + B) * 2, (A * B) / 2, (A / B) / 2]
14     # Check if any of the incorrect results match the original result
15     return any(orig_result == result for result in incorrect_results)
```

Explanation and Justification:

- “original_program” function: The function subtracts B from A and multiplies the result by 2, representing the original program.

```
1 def original_program(A, B):
2     A = A - B
3     C = A * 2
4     return C
```

Image 1

- “is_ambiguous” function: This function compares the original result of a given A value with incorrect results obtained from various arithmetic operations on A and B.

```
5 def is_ambiguous(A, B):
6     orig_result = original_program(A, B)
7     # Generate incorrect results by applying various operations to A and B
8     incorrect_results = [(A + B) * 2, (A * B) * 2, (A / B) * 2,
9                          A * B + 2, A - 2, (A + B) / 2,
10                          (A + B) - 2, (A * B) + 2, (A / B) + 2,
11                          (A + B) / 2, (A * B) - 2, (A / B) - 2,
12                          (A + B) * 2, (A * B) / 2, (A / B) / 2]
13     # Check if any of the incorrect results match the original result
14     return any(orig_result == result for result in incorrect_results)
```

Image 2

- ambiguous_A_values list: This list stores the values of A that are found to be ambiguous for the given B value.

```
16 # Given B value
17 B = 1
18 # Initialize a list to store ambiguous A values
19 ambiguous_A_values = []
```

Image 3

- Detecting through a range of A values: The code iterates through a range of A values from -100 to 100 to cover a wide range of possible inputs.

```
1 # Iterate through a range of A values from -100 to 100
2 for A in range(-100, 101):
3     # Check if the current A value is ambiguous for the given B value
```

Image 4

- Checking ambiguity: For each A value, the code checks if it is ambiguous using the is_ambiguous function. If the A value is found to be ambiguous, it is appended to the ambiguous_A_values list.

```
25 if is_ambiguous(A, B):
26     # Append the current value to the list of ambiguous A values
27     ambiguous_A_values.append(A)
```

Image 5

- Printing ambiguous A values: The code prints a list of ambiguous A values for a given B value, indicating that the test cases cannot detect any possible incorrect use of arithmetic operators.

```
26 # Print the ambiguous A values for B=1
27 print("Ambiguous A values for B=1 are:", ambiguous_A_values)
```

Image 6

Link to code:

<https://github.com/quynezz/SWE300>

09