HW01: **Model-Based, SPT, BVA, & Eq. Classes**

Name: Vanshaj Tyagi

Course: SSW-567

Question:

Part 1

Parameter	Invalid Class	Equivalence Class	Equivalence Class	Invalid Class
x	x_1	x_2		
y		y_2	y_3	
z		z_2		z_4

A software system is represented by three input parameters (x, y, z). The representative Equivalence Classes values for the three variables are described in the table above.

Where; x_i, y_j, and z_k represent (respectively) selected values of x, y, and z that makes the parameter fall in certain class and i, j, & k are integers between 1 and 4. Empty values in the table indicate nonexistent inputs. For instance, there are no invalid input for parameter y. You are required to provide the necessary test cases to satisfy without exceeding the following requirements:

1. Weak Robust Equivalence Class testing.
2. Strong Normal Equivalence Class testing.

Test cases should be written in a table using $F(x_i, y_j, z_k)$ format.

Part 2

Using a maximum of 3 lines each, describe the following:

1. The difference between validation and verification software quality processes.
2. The difference between prevention and detection software quality strategies.
3. What is the software testing myth that you used to believe in?

Answer:

Part 1

Weak robust testing is an efficient, but less thorough, method. It selects a single, typical example from each valid input group. When it comes to invalid inputs, it assumes that errors will happen one at a time, so it creates separate tests for each invalid input, rather than combining them. This is often called the "**single-fault assumption**." This approach is good for finding simple bugs but may miss more complex issues that arise when multiple invalid inputs are combined.

Test Case	Function Call	Description
1	F(x2, y2, z2)	All valid baseline case
2	F(x1, y2, z2)	Invalid x only
3	F(x2, y3, z2)	Alternative valid y class
4	F(x2, y2, z4)	Invalid z only

Strong normal equivalence class testing is a comprehensive method for testing valid inputs. It tests all possible combinations of valid values from every input category. This means if you have multiple fields, it creates a test case for every possible pairing of a valid value from the first field with a valid value from the second, and so on. This approach is thorough but can create a very large number of test cases.

Test Case	Function Call	Description
1	F(x2, y2, z2)	Valid x2, valid y2, valid z2
2	F(x2, y3, z2)	Valid x2, valid y3, valid z2

Part 2

1. **Validation vs. Verification**
    a. **Verification** asks, "Are we building the product right?" by checking if the software meets its design specifications through methods like code reviews.
    b. **Validation** asks, "Are we building the right product?" by ensuring the software meets the user's actual needs through activities like acceptance testing.

    Verification focuses on the internal process, while validation focuses on the final product.

2. **Prevention vs. Detection**
    a. **Prevention** is a proactive strategy that stops defects from happening in the first place through measures like improving processes and conducting design reviews.
    b. **Detection** is a reactive strategy that finds and fixes defects after they have already occurred through activities like running tests and inspections.

    Prevention is often more cost-effective because it catches problems early, while detection can be more expensive since it finds issues later in the development cycle.

3. **Software Testing Myth**
    a. A common myth is the belief that testing can prove software is completely free of bugs. This is false, as it's impossible to test every scenario in complex software.
    b. Testing can only demonstrate the presence of bugs, not their complete absence.

    Because of this, testing is a sampling process that aims to find as many defects as possible, not to eliminate all of them.