Bledea Mihaela Alexandra
931

1. Equivalence class is a partition of input domain of a program. Equivalence class partitioning is a black-box technique that divides the input domain of a program into classes of data from which test cases can be derived. If a program works correctly for one scenario in an equivalence class, it will work correctly for all the scenarios in the same class, and vice versa. Therefore, it reduces the number of test cases that need to be developed while still reasonably covering the possible scenarios.

Example: -> consider a software application that accepts ages between 18 and 50. The possible equivalence classes would be:

1) Invalid Equivalence Class: any age less than 18 and any age more than 50.
2) Valid Equivalence Class: any age between 18 and 50.

-> we can pick a single test case from each of the equivalence classes.

2. When talking about white box testing, predicate coverage means elaborating test cases such that some path is executed under all possible circumstances. Predicate coverage is achieved when all possible combinations of truth values of the conditions affecting the path have been considered in the test cases.

Example:
```
int example(int a, int b, int c){
        int x =0;
        if(a ==1){
            x++;
        }
        if(b ==1){
            x=x+2;
        }
        if(c==1){
            x=x+3;
        }
        return x;
}
```

all combinations

TEST CASES

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	1
1	1	0
1	1	1

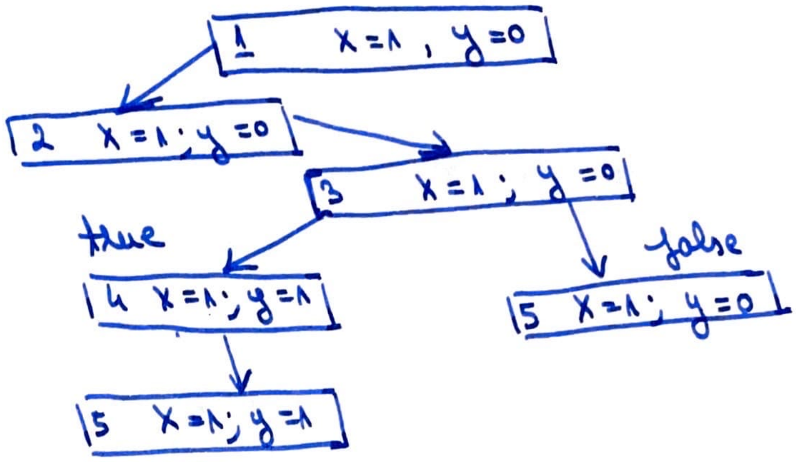
Bledea Mihaela Alexandra 931

3. A symbolic execution tree is a visualization tool used in software testing. It represents the exploration of different paths in a program using symbolic inputs, where each node denotes a branching point and each edge indicates a specific outcome. It helps analyze the program's behaviour for varying inputs.

Example:

```
1  int example(int x) {
2       int y = 0;
3       if (x == 1)
4           y = 1;
5       return y;
6  }
```

For X = 1



4. Liveness properties are linear-time properties that say that "something good will happen in the future", especially in mutual exclusion problems.

Example: Consider a system where multiple ~~processors~~ processes are sharing a single printer. A liveness property in the system could be "Every print request made by a process eventually gets served". This means that no matter when or by which process a print request is made, it will eventually be processed and the document will be printed.

Liveness properties are about guaranteeing that something good will eventually happen, but they don't provide an upper bound on the waiting time.