Take Home A - Individual work

Astalus Adrian
Claudiu
group 931

Q1: Black box testing - equivalence classes

In black box testing, equivalence classes are considered to be partitions of the input domain of a program. These classes divide the input data into groups or classes that are predicted to have a similar behaviour ~~one~~ or have the same output/result. The main idea is that if a certain value behaves in a specific way, then it is likely that all the values of the same class will ~~have~~ behave the same.

ex: we have a 'register' feature on our app; it consists of some fields that must be filled in; possible equivalence classes: all input fields are valid, invalid e-mail provided, not all required fields are filled in, e-mail or username already in use, etc.

Q2: White box testing - predicate coverage

Predicate coverage is a metric used in white box testing in order to determine whether or not all the possible combinations of truth values of the conditions from a selected path have been explored during the execution of the test cases. If all the possible cases have been covered, then predicate coverage is achieved. The goal of predicate coverage is to uncover any possible logical errors in the code.

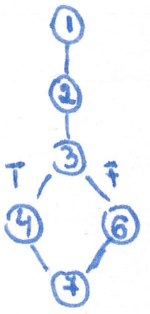
ex: we have a method that computes the discount to be applied to a product based on its price & quantity. This method consists of several if-else statements. Test cases should cover all of the possible branches of the statements.

Q3: Symbolic execution tree

The symbolic execution tree is a graphical representation of a program's control flow. Using this method, we are able to analyse and explore all the feasable paths of a program, thus detecting any potential errors in the program. Moreover, ~~using~~ with the help of this tree we can generate test cases that offer more code coverage in a more efficient manner.

The tree consists of nodes, associated with each statement, directed arcs that represent the transitions between each statement. For conditional statements, a node has two arcs, one for each outcome (True or False).

ex:
```
1 int a=2;
2 int b=3
3 if(a+b>6){
4     print("A");
5 else{
6     print("B");
  }
7 print("C");
```



Q4: Model checking - liveness properties

Liveness properties represent a set of properties of model checking that focus on the idea that something good will happen at some point in the future, rather than the idea that nothing bad will occur.

These properties are important for validating systems that are concurrent or reactive. They help to detect situations in which a system deadlocks or becomes unresponsive.

ex: in the case of a software that is used to manage traffic lights at an intersection, a ~~livel~~ liveness property would be that eventually, every direction of the traffic is shown the green light.