TA7

1. Explain the differences and relationships between software quality assurance and quality control, between software quality assurance/control and software testing, and between software verification and validation.

Software quality assurance and quality control:

Quality assurance refers to all activities used to measure and enhance the software product usually done before the release. While quality control has to do with activities designed test the software product usually done after the release. Regardless they both aim to have a improve the software.

Software quality assurance / control and software testing:

In testing, we design the test cases, execute the software program, and confirm that we have the correct output. Quality assurance / control is confirming that the product is corresponding to its requirements. Testing is done to find the parts that do not work the way they should in order to fix them

Software verification and validation:

In verification we check whether a software product conforms to its requirements and specification. While in validation, we check that a software product has the requirements and specifications that were set by the customer.

2. Use one sentence to explain each of the following 4 techniques to perform software verification and validation:

- Testing
- Inspection
- Formal method
- Static analysis

Testing: checks if an error occurred due to any activity while running the project (dynamically). Testing depends on the size of body of a project be tested (unit testing, system testing and so on).

Inspection: defined as review practice done on software project to know about the presence of any defect.

Formal Method: technique used in software verification and validation with the help of well-defined mathematical techniques. One can say its instances are taken from theoretical computer science.

Static analysis: analysis done on a program when it is not being executed (non-dynamic), code review.

3. Use one sentence to explain each of the following levels of software testing and the actor who will contact the test:

- Unit test
- Function or use case test
- Component test
- Integration test
- System test
- Acceptance test

Unit test: The programmer tests different modules separately while still developing the software.

Function or use-case test: a team of software testers will contact this test; they will identify test cases that exercise the whole system on every transaction basis.

Component test: a developer conducts this test. It involves testing of multiple functionalities as a single software and is used to check the transformation of data between various subsystems.

Integration test: Team of software testers conducts this test in which different modules are combined to form subsystems and test cases are executed to find the interaction of modules in different ways

System test: team of software testers will conduct this test in which teste the entire software system and checks whether the software implemented meets the requirement. This is the final testing done before the software is delivered to the clients.

Acceptance test: checks whether the software developer meets the users' need. This test is conducted by the end users or customers.

4. The following is an example Java method. Using this example to explain the difference between black-box testing and white-box testing of this method.

```
int max2(int x, int y) {if (x >= y) {x = 2 * x; return x;} else {y = 2 * y; return y;}}
```

In white-box testing: the tester, beforehand, knows the structure of the program. In this case, the program has an if-else condition, multiplication, and return statement. The tester will be testing while all the program failure scenarios are kept in mind. In this java method example, if x < y then the program control should go to the else statement. But, if x >= y, then, it should go to the if statement.

In black-box testing: the internal structure (the functionality and expected behavior of the program) is known to the tester and tests are done according to that. In this Java code example, the program should return double the value of x or y depending on which is the greater one. So when a tester tests max2(2,4) he or she expects to see double the value of 4 (8).

5. To test a Factorial function/method “int factorial (int n)”, what are the 3 equivalence classes of input value n to create test cases? Explain your answer.

Factorial functions must be done on positive integers and only positive integers. There is no factorial of a negative number. So, in this case we will be testing when n < 0 (negative), n == 0, and n > 0 () positive. We expect the result of the first to be and error, the result of the second one should be 1 and the result of the last should be n!.

6. To test an array sorting function/method “void sortArray (int[] array)”, list 5 boundary values of input array to create test cases. Explain your answer.

- Test to see if array is empty (array == []) to confirm that no further processing required and it should be returned.
- Test to see if the elements are unsorted to confirm the processing time to sort a completely unsorted array.
- Test to see if the array is sorted to confirm that the processing time will be reduced because there will be no need to sort anything.
- Test to see if the first element is the biggest element and the rest of the array is sorted to confirm that the processing time will be reduced as the array is already sorted and each iteration will swap the biggest element with the one after until it is at the end of the array.

- Test to see if the array is sorted and the element at the end is the smallest element to confirm the change in processing time as the array will be sorted except for the smallest element which will be replaced with the element before it until it reaches the start of the array.

7. Consider the simple case of testing 2 variables, x and y, where x must be a non-negative number, and y must be a number between -5 and +15. Utilizing boundary value analysis, list the test cases. Explain your answer.

In boundary value analysis the input variables should be minimum value, maximum value, just above minimum value, just below maximum value, and a value in between (normal value).

Y variable value	X variable values	System acceptance
-6	-1	Should not accept
-5	0	Should accept
-4	1	Should accept
14	MAX_VALUE – 1	Should accept
15	MAX_VALUE	Should accept
16	MAX_VALUE+1	Should not accept

Based on the above information, combine the test cases for x and y variables. So, we will have 6 values for x for every value of y. This means that will have 36 cases in total

8. Consider the following UML activity diagram that models some code.

a) How many logical paths are there? List them all.

b) How many paths are required to cover all the statements? List them all.

c) How many paths are required to cover all the branches? List them all.

a. There are 4 logical paths.
   C1 → S3
   C1 → C2
   C1 → S3 → C2
   C1 →S1 → C2 → S3
b. One
   C1 →S3 → C2 → S3
c. Two
   C1 → C2 → S3
   C1 →S1 → C2 → S3