Testing Principles

To understand this, consider a scenario where you are moving a file from folder ***folder1*** to Folder ***folder2***.

Think of all the probable ways for testing.

Apart from the usual scenarios, you can also test the following conditions

1. Trying to move the file when it is Open
2. You do not have the security rights to paste the file in Folder ***folder2***
3. Folder ***folder2*** is on a shared drive and storage capacity is full.
4. Folder ***folder2*** already has a file with the same name, in fact, the list is endless
5. Or suppose you have 15 input fields to test, each having 5 possible values, the number of combinations to be tested would be 5^15

If you were to test the entire possible combinations project EXECUTION TIME & COSTS would rise exponentially. We need certain principles and strategies to optimize the testing effort.

Principles as follows:

1) Exhaustive testing is not possible

Testing all the functionalities using all valid and invalid inputs and preconditions is known as *Exhaustive testing*.

Assume we have to test an input field which accepts age between 18 to 20 so we do test the field using 18, 19, 20. In case the same input field accepts the range between 18 to 100 then we have to test using inputs such as 18, 19, 20, 21, …., 99, 100. It's a basic example, you may think that you could achieve it using automation tool. Imagine the same field accepts some billion values. It's impossible to test all possible values due to release time constraints.

If we keep on testing all possible test conditions then the software execution time and costs will rise. So instead of doing exhaustive testing, risks and priorities will be taken into consideration whilst doing testing and estimating testing efforts.

2) Defect Clustering

Defect Clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.

As per the *Pareto Principle* (80-20 Rule), 80% of issues comes from 20% of modules and remaining 20% of issues from remaining 80% of modules. So we do emphasize testing on the 20% of modules where we face 80% of bugs.

3) Pesticide Paradox

Pesticide Paradox in software testing is the process of repeating the same test cases again and again, eventually, the same test cases will no longer find new bugs. So to overcome this Pesticide Paradox, it is necessary to review the test cases regularly and add or update them to find more defects.

4) Testing shows a presence of defects

Testing shows the presence of defects in the software. The goal of testing is to make the software fail. Sufficient testing reduces the presence of defects. In case testers are unable to find defects after repeated regression testing doesn't mean that the software is bug-free.

Testing talks about the presence of defects and don't talk about the absence of defects.

5) Absence of Error – fallacy

99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and the software is not addressing the business needs.

The software which we built not only be a 99% bug-free software but also it must fulfil the business needs otherwise it will become an unusable software.

6) Early Testing

Defects detected in early phases of SDLC are less expensive to fix. So conducting early testing reduces the cost of fixing defects.

7) Testing is context dependent

Testing approach depends on the context of the software we develop. We do test the software differently in different contexts. For example, online banking application requires a different approach of testing compared to an e-commerce site.