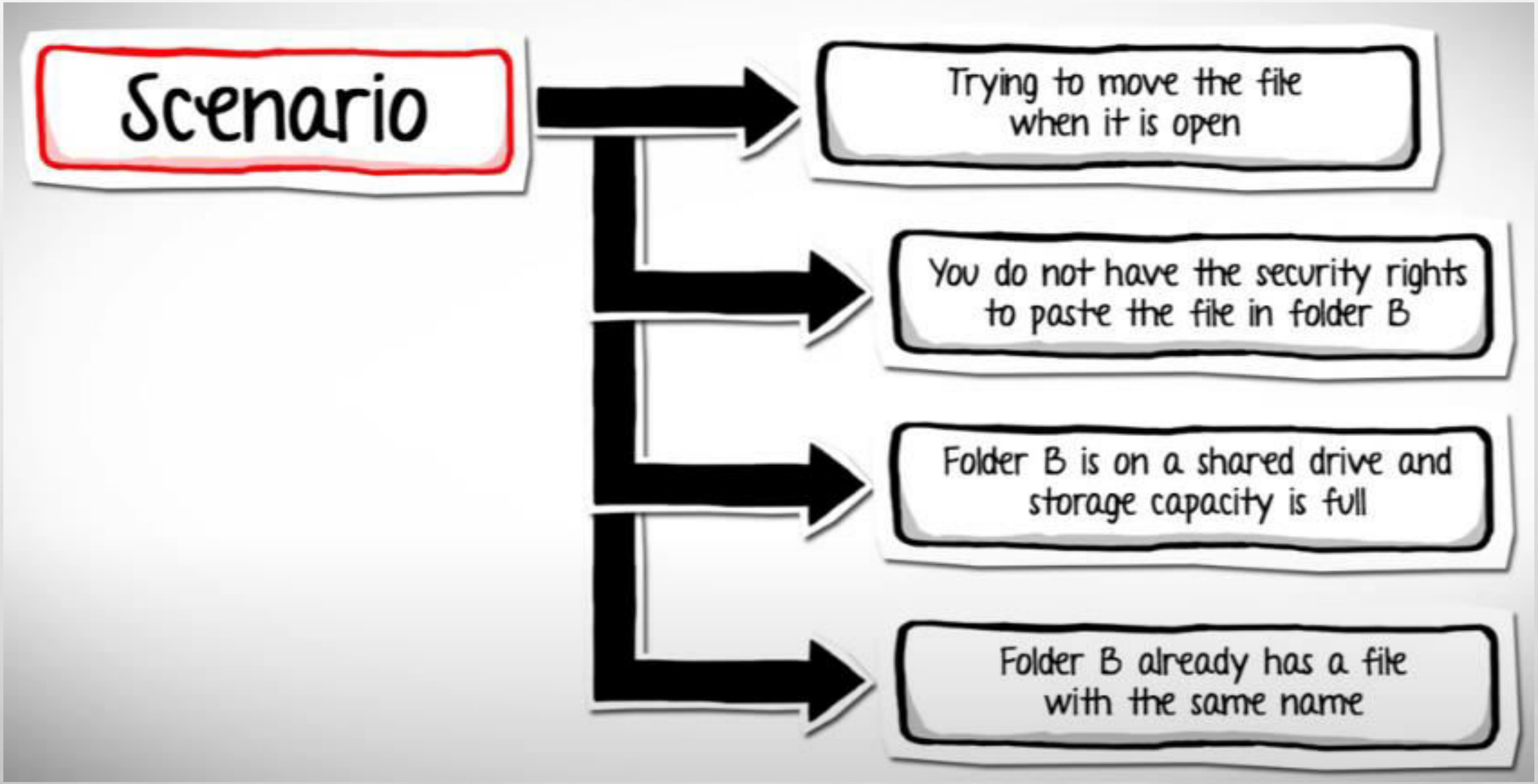
TESTING PRINCIPLES

Seven Principles of Software Testing

- EXHAUSTIVE TESTING IS NOT POSSIBLE
- DEFECT CLUSTERING
- PESTICIDE PARADOX
- TESTING SHOWS A PRESENCE OF DEFECTS
- EARLY TESTING
- TESTING IS CONTEXT DEPENDENT
- ABSENCE OF ERROR - FALLACY



Scenario

- Trying to move the file when it is open
- You do not have the security rights to paste the file in folder B
- Folder B is on a shared drive and storage capacity is full
- Folder B already has a file with the same name

1. EXHAUSTIVE TESTING IS NOT POSSIBLE

- TESTING ALL THE FUNCTIONALITIES WITH VALID AND INVALID COMBINATIONS OF INPUT DATA DURING ACTUAL TESTING IS IMPOSSIBLE.
- THEREFORE, TESTING OF A FEW COMBINATIONS IS DONE BASED ON PRIORITY USING DIFFERENT TECHNIQUES.

A Website has 4 pages, 3 menus with 2 options each.
Option has 5 fields, 2 types of inputs each fields.
Each input type has 50 possible values.
The total test conditions for exhaustive testing

A Website has 4 pages, 3 menus with 2 options each.
Option has 5 fields, 2 types of inputs each fields.
Each input type has 50 possible values.
The total test conditions for exhaustive testing

4 pages * 3 menus * 2 options * 5 fields * 2 types * 50 values = 30,000 tests

2. DEFECT CLUSTERING

- IN A PROJECT, A SMALL NUMBER OF MODULES CONTAIN MOST OF THE DEFECTS DETECTED.
- THIS IS THE APPLICATION OF THE PARETO PRINCIPLE TO SOFTWARE TESTING (80-20 RULE)
    - ❑ 80% OF ISSUES COMES FROM 20% OF MODULES AND
    - ❑ REMAINING 20% OF ISSUES FROM REMAINING 80% OF MODULES.

80% of work is completed by 20% of your team

There is often a wide performance gap between your top performers and the rest of the team.

80% of customers only use 20% of software features

Most users don't use power features as they find power features to be annoying

3. PESTICIDE PARADOX

- IF YOU EXECUTE THE SAME SET OF TEST CASES AGAIN AND AGAIN OVER THE PERIOD OF TIME THESE SET OF TEST CASES CANNOT IDENTIFY NEW DEFECTS IN THE SYSTEM.
- SO TO OVERCOME THIS PESTICIDE PARADOX, IT IS NECESSARY TO REVIEW THE TEST CASES REGULARLY AND ADD OR UPDATE THEM TO FIND MORE DEFECTS.

Repetitive use of the same pesticide mix to destroy insects during farming over the time lead to the insects developing resistance to the pesticide, Thereby making thereby pesticides ineffective on insects

The same applies to software testing

4. TESTING SHOWS A PRESENCE OF DEFECTS

- TESTING TALKS ABOUT THE PRESENCE OF DEFECTS & DON’T TALK ABOUT THE ABSENCE OF DEFECTS
- SOFTWARE TESTING CAN ENSURE THAT DEFECTS ARE PRESENT, BUT IT CAN NOT PROVE THAT SOFTWARE IS DEFECTS FREE.
- EVEN MULTIPLE TESTING CAN NEVER ENSURE THAT SOFTWARE IS 100% BUG-FREE.

An application might seem to be error-free after going through different stages of testing.
But during the production in the environment, the user may come across any defect which did not occur during the testing.

5. EARLY TESTING

- TESTING NEEDS TO BE PERFORMED ON REQUIREMENT DOCUMENTS, SPECIFICATION OR ANY OTHER TYPE OF DOCUMENT.
- SO THAT IF REQUIREMENTS ARE INCORRECTLY DEFINED THEN IT CAN BE FIXED IMMEDIATELY RATHER THAN FIXING THEM IN THE DEVELOPMENT PHASE.
- THE COST INVOLVED IN FIXING SUCH DEFECTS IS VERY LESS WHEN COMPARED TO THOSE THAT ARE FOUND DURING THE LATER STAGES OF TESTING.

Assume two scenarios, first is identifying an incorrect requirement in the requirement gathering phase and the second is identifying a bug in the fully developed functionality.

Assume two scenarios, first is identifying an incorrect requirement in the requirement gathering phase and the second is identifying a bug in the fully developed functionality.

It is cheaper to change the incorrect requirement compared to fixing the fully developed functionality which is not working as intended

6. TESTING IS CONTEXT DEPENDENT

- ALL THE DEVELOPED SOFTWARE’S ARE NOT IDENTICAL.
- DIFFERENT DOMAINS ARE TESTED DIFFERENTLY.
- THUS, TESTING IS PURELY BASED ON THE CONTEXT OF THE SOFTWARE.

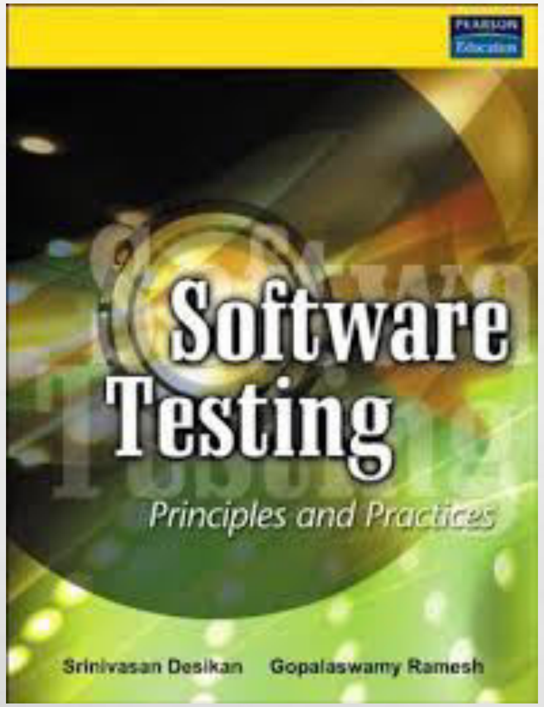
Testing any POS system at a retail store will be different than testing an ATM machine

7. ABSENCE OF ERROR - FALLACY

- IF THE SOFTWARE IS TESTED FULLY AND NO DEFECTS ARE FOUND BEFORE RELEASE, THEN YOU CAN CONSIDER THE SOFTWARE TO BE 99% DEFECT-FREE.
- BUT IF THE SOFTWARE IS TESTED AGAINST WRONG REQUIREMENTS, THEN IT IS UNSTABLE.
- SO, SOFTWARE WHICH WE BUILT NOT ONLY BE A 99% BUG-FREE , BUT ALSO IT MUST FULFIL THE BUSINESS NEEDS OTHERWISE IT WILL BECOME AN UNUSABLE SOFTWARE.

Testing an application related to an e-commerce site and the requirements against "Shopping Cart" functionality is wrongly interpreted and tested. Finding more defects would not help in moving the application into the next phase or in the production environment.

REFERENCES


PEARSON Education

Software Testing

Principles and Practices

Srinivasan Desikan Gopalaswamy Ramesh

Guru99

edureka!

Software Testing Help