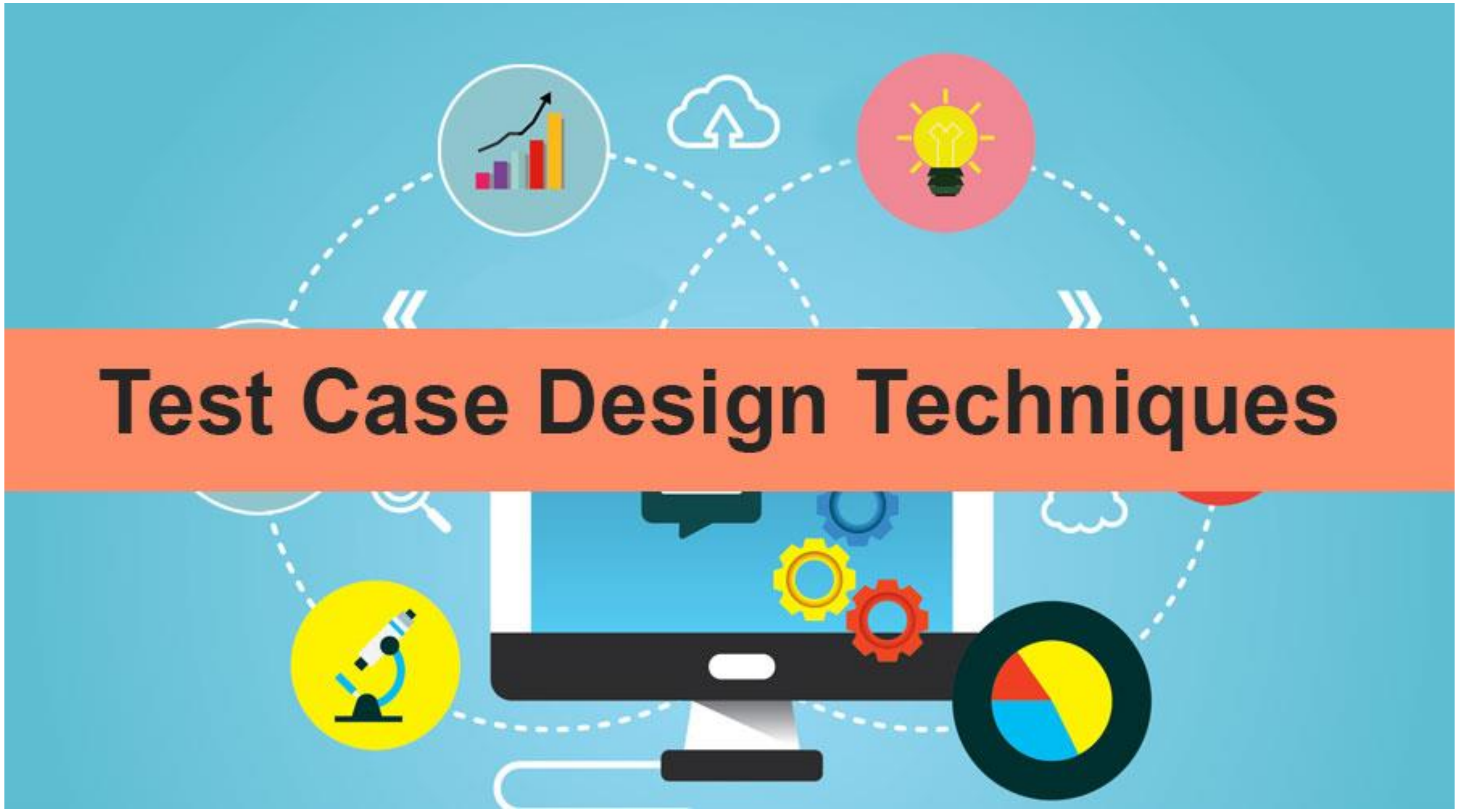
System Analysis and Design

Software Testing II

ISTQB®
Certified Tester
Foundation Level

Prof. Rania Elgohary
rania.elgohary@cis.asu.edu.eg
Dr. Yasmine Afify
yasmine.afify@cis.asu.edu.eg



Test case design techniques are used to pick the test cases in a systematic manner.
By using these techniques, we could save lots of testing time and get the good test coverage.

Black, White and Experienced based

Black (Specification Based)

- Based on **requirements**
- From the requirements, tests are created
- Specification Models can be used for systematic test case design

Techniques
- Equivalence Partitioning
- Boundary Value Analysis
- Decision Tables
- State Transition Testing
- Pairwise Testing

White (Structure Based)

- Based on **code and the design** of the system
- The tests provide the ability to derive the extent of coverage of the whole application

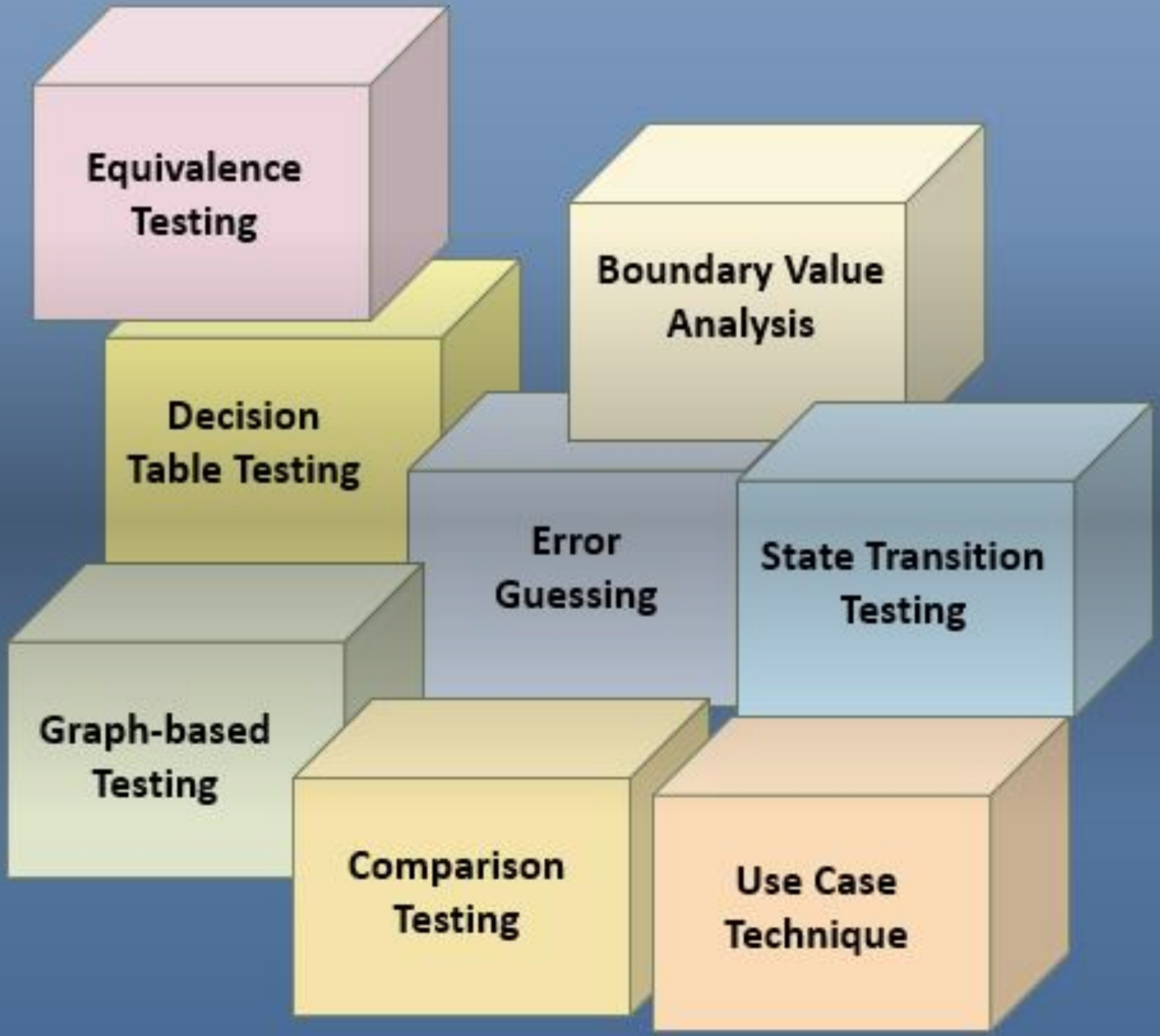
Techniques
- Statement coverage
- Branch Coverage
- Decision Coverage

Experience (Black box)

- Based on the **knowledge of the tester**
- Using past experienced use & intuition to “guess” where errors may occur

Techniques
- Error Guessing
- Exploratory Testing

Black Box Testing Techniques



www.educba.con

Specification-based (black-box) Equivalence Partitioning Testing EP

Equivalence Partitioning testing, a black-box test technique to reduce # of required test cases.
What is it?

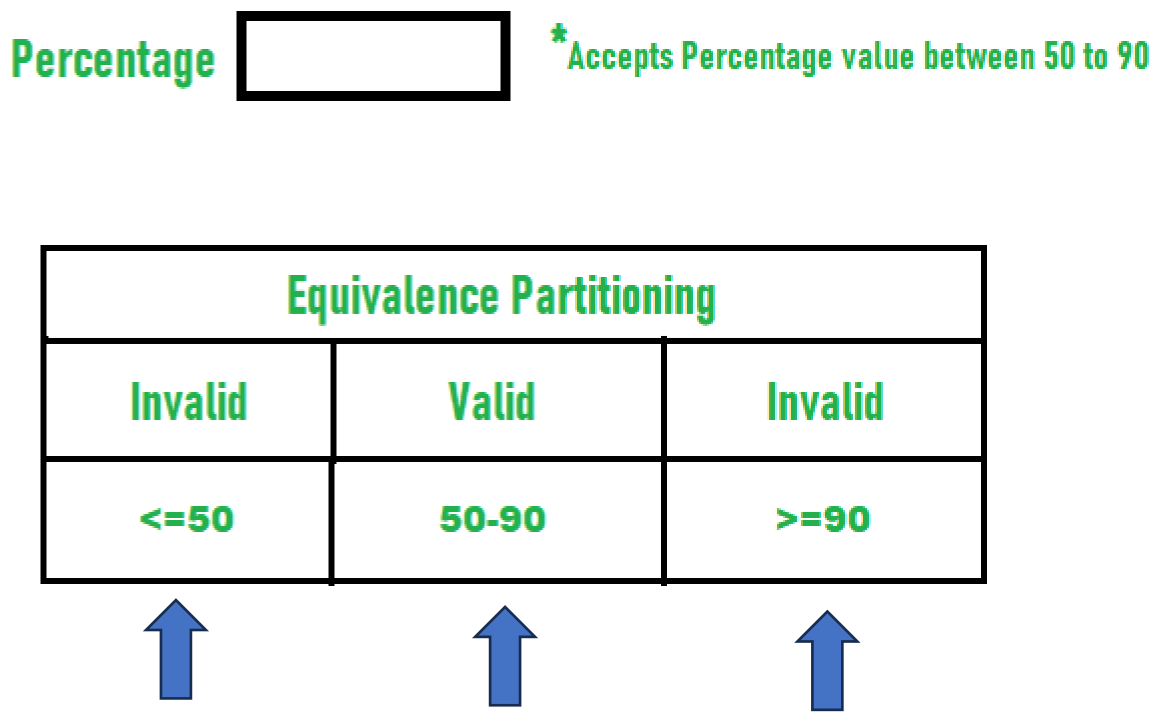
steps in equivalence testing:
- identify classes of inputs with same behavior
- test on at least one member of each equivalence class
- assume behavior will be same for all members of class

criteria for selecting equivalence classes:
- *coverage* : every input is in one class
- *disjointedness* : no input in more than one class
- *representation* : if error with 1 member of class, will occur with all

Specification-based (black-box) Equivalence Partitioning Testing EP

- Equivalence Partitioning testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
- Divide the test conditions into classes (groups).
- From each group we do test only one condition.
- Assumption is that all the conditions in one group work in the same manner. If a condition from a group works, then all conditions from that group work and vice versa.
- It reduces lots of rework and also gives a good test coverage.

1. Specification-based (black-box) Equivalence Partitioning EP



Percentage

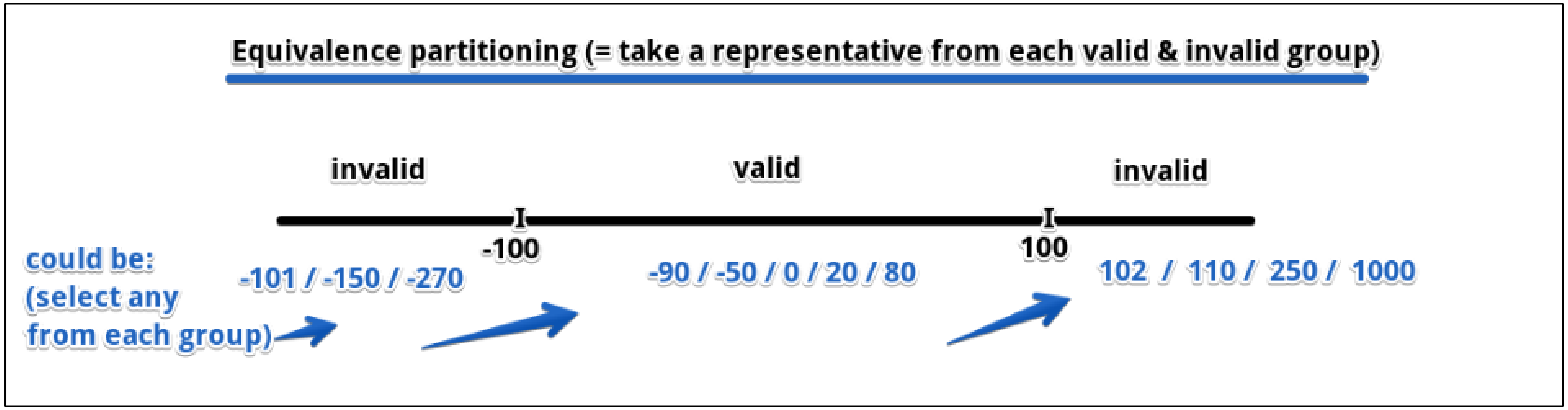
*Accepts Percentage value between 50 to 90

Equivalence Partitioning		
Invalid	Valid	Invalid
<=50	50-90	>=90

Inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be processed in the same way.

Specification-based (black-box) Equivalence Partitioning EP

- A program which accepts an integer in the range -100 to +100



Specification-based (black-box) Boundary Value Analysis BVA

It is a black-box testing technique that believes and extends the concept that the density of defect is more towards the boundaries.

This is done for the following reasons:

- Programmers usually are not able to decide whether they have to use <= operator or < operator when trying to make comparisons.
- Different terminating conditions of for-loops, while loops, and repeat loops may cause defects to move around the boundary conditions.
- The requirements themselves may not be clearly understood, especially around the boundaries, thus causing even the correctly coded program to not perform the correct way.

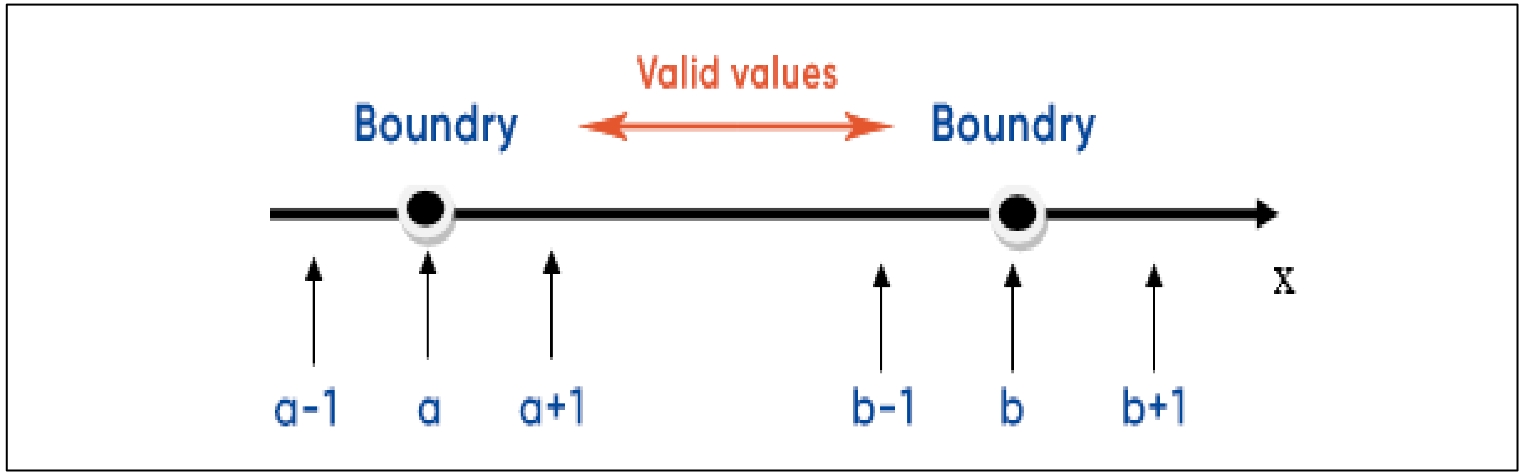
Specification-based (Black box) Boundary value analysis BVA

- Take the test conditions as partitions and design the test cases by getting the **boundary values of the partition**. The boundary between two partitions is the place where the behavior of the application varies. The test conditions on either side of the boundary are called boundary values.
- Get both **valid** boundaries (from the valid partitions) and **invalid** boundaries (from the invalid partitions).

  For example: To test a field which accepts only an amount more than

  10 and less than 20.
- Then we take the boundaries as 10-1, 10, 10+1, 20-1, 20, 20+1. Instead of using lots of test data, we just use 9, 10, 11, 19, 20 and 21.

Specification-based (black-box) Boundary Value Analysis



- 6 test cases.
- Can be applied on numeric values and dates.
- Boolean present a problem for BVA.
- Also called robustness testing.

Specification-based (black-box) Decision Table Testing

- Decision tables are a precise and compact way to model complicated logic. They are ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions.
- Each column of the decision table represents a rule.
- Number of test cases = number of rules.

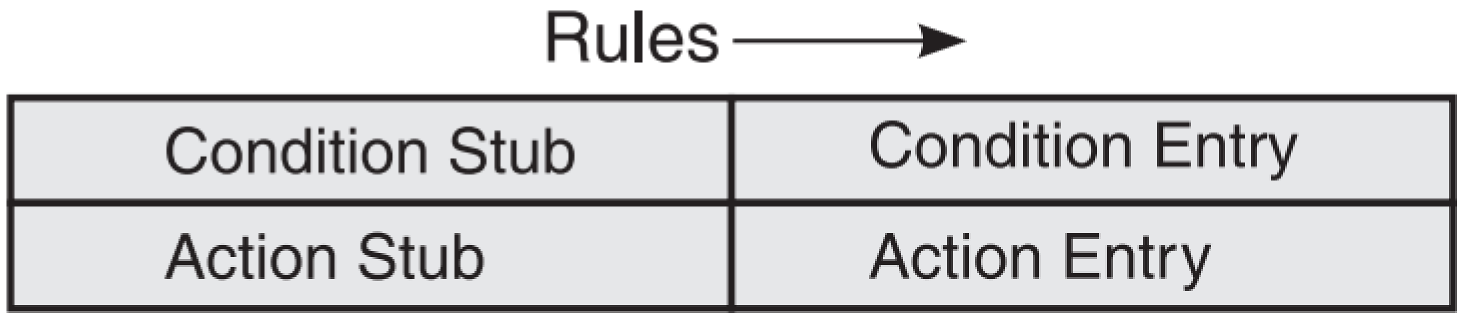


FIGURE 4.9 Structure of Decision Table.

Specification-based (black-box) Decision Table Testing

Applications of Decision Tables This technique is useful for applications characterized by any of the following:

a. Prominent if-then-else logic.

b. Logical relationships among input variables.

c. Calculations involving subsets of the input variables.

d. Cause-and-effect relationships between inputs and outputs

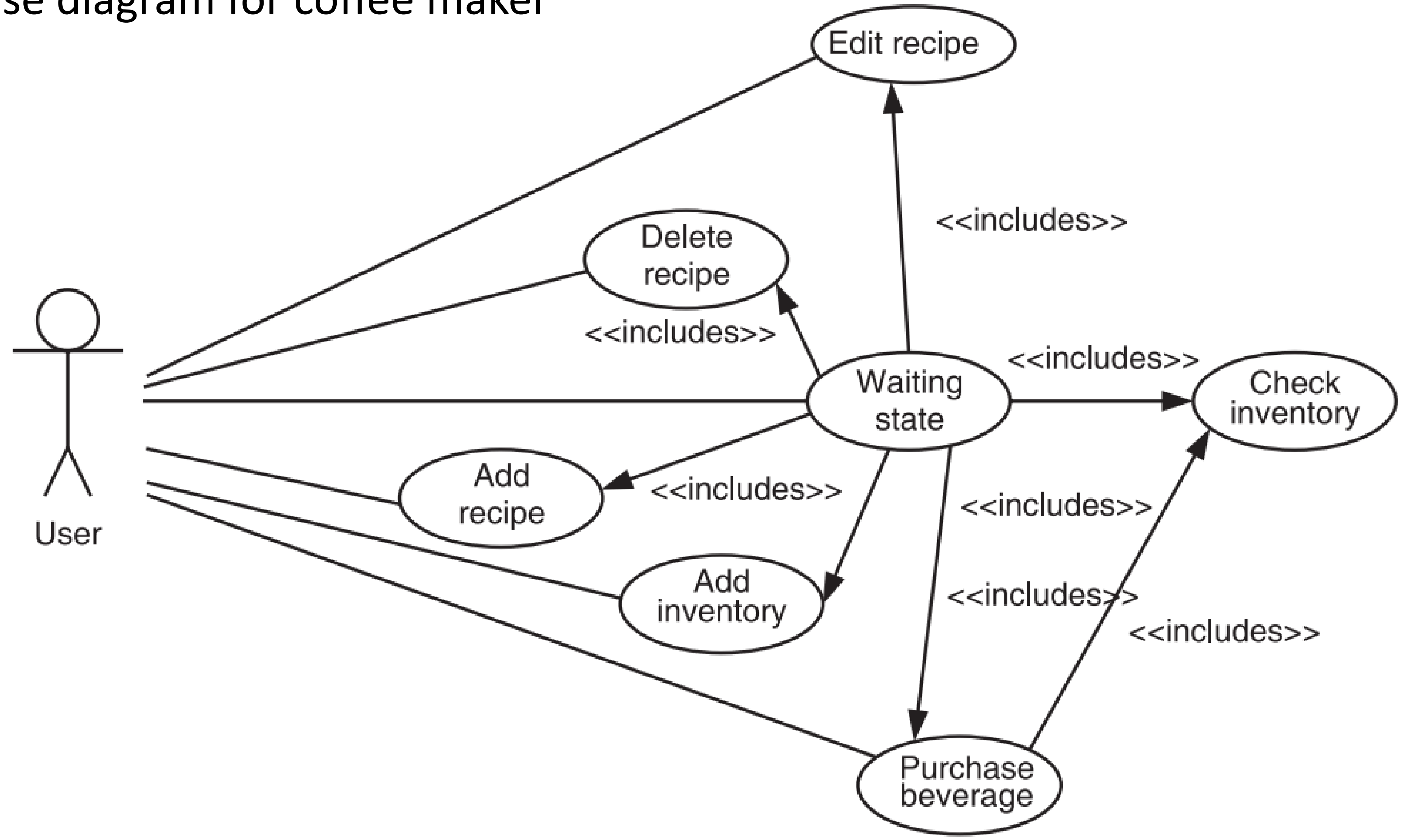
	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
C_1: $a < b + c$?	F	T	T	T	T	T	T	T	T	T	T
C_2: $b < a + c$?	—	F	T	T	T	T	T	T	T	T	T
C_3: $c < a + b$?	—	—	F	T	T	T	T	T	T	T	T
C_4: $c = b$?	—	—	—	T	T	T	T	F	F	F	F
C_5: $a = c$?	—	—	—	T	T	F	F	T	T	F	F
C_6: $b = c$?	—	—	—	T	F	T	F	T	F	T	F
a_1: Not a triangle	×	×	×								
a_2: Scalene											×
a_3: Isosceles							×		×	×	
a_4: Equilateral				×							
a_5: Impossible					×	×		×			

FIGURE 4.10 Example of Decision Table.

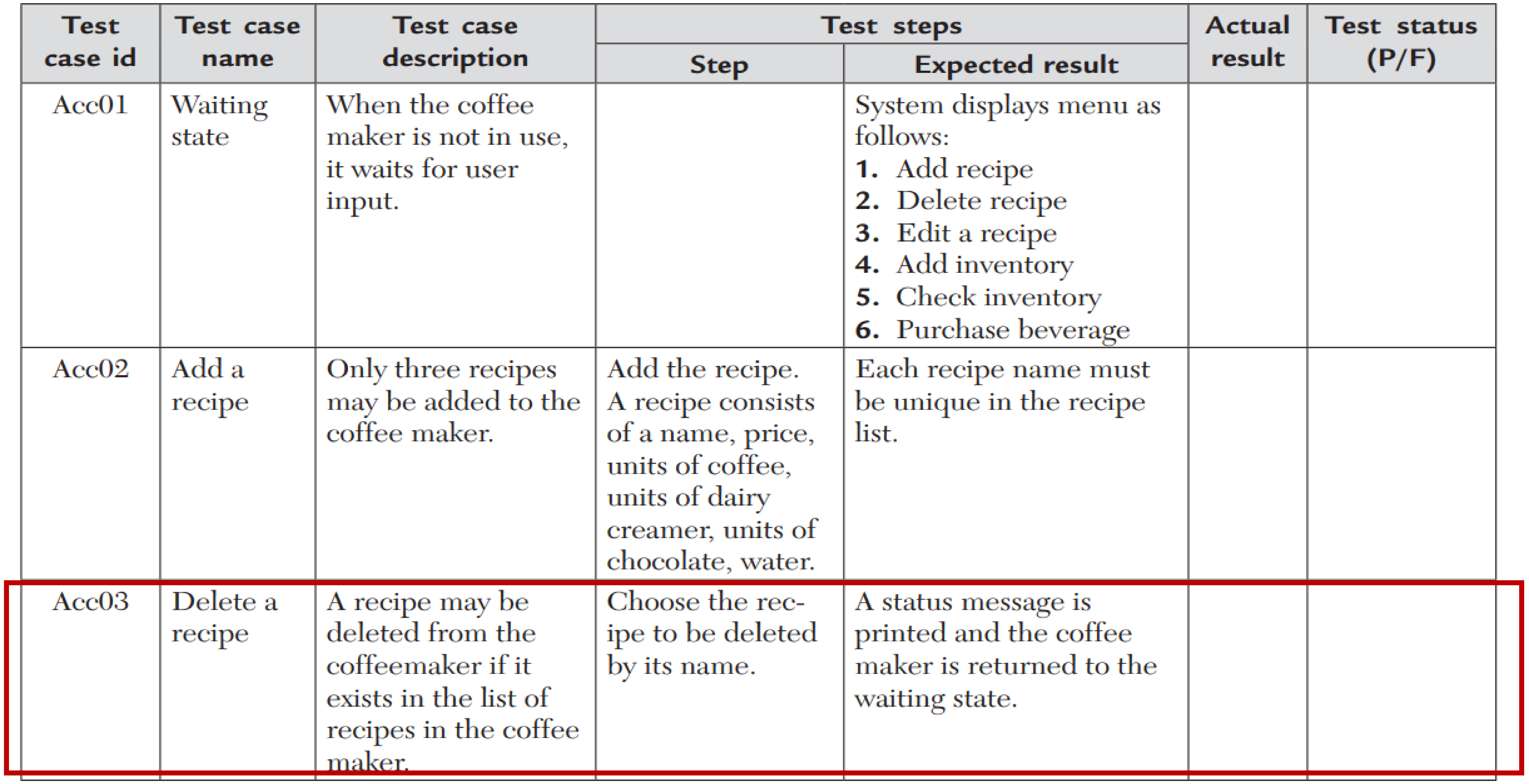
Specification-based (black-box) Use Case Testing

- Tests can be derived from use cases.
- Each use case has **preconditions** which need to be met for a use case to work successfully.
- Each use case terminate with **postconditions** which are the observable results and final state of the system after the use case has been completed.
- Designing test cases from use cases **may be combined** with other specification-based test techniques for details in inside the use case itself.
- Use cases are very useful for designing **acceptance test cases** with customer/user participation.

Use case diagram for coffee maker



Acceptance Test Cases



Test case id	Test case name	Test case description	Test steps		Actual result	Test status (P/F)
			Step	Expected result		
Acc01	Waiting state	When the coffee maker is not in use, it waits for user input.		System displays menu as follows: **1.** Add recipe **2.** Delete recipe **3.** Edit a recipe **4.** Add inventory **5.** Check inventory **6.** Purchase beverage		
Acc02	Add a recipe	Only three recipes may be added to the coffee maker.	Add the recipe. A recipe consists of a name, price, units of coffee, units of dairy creamer, units of chocolate, water.	Each recipe name must be unique in the recipe list.		
Acc03	Delete a recipe	A recipe may be deleted from the coffeemaker if it exists in the list of recipes in the coffee maker.	Choose the recipe to be deleted by its name.	A status message is printed and the coffee maker is returned to the waiting state.		

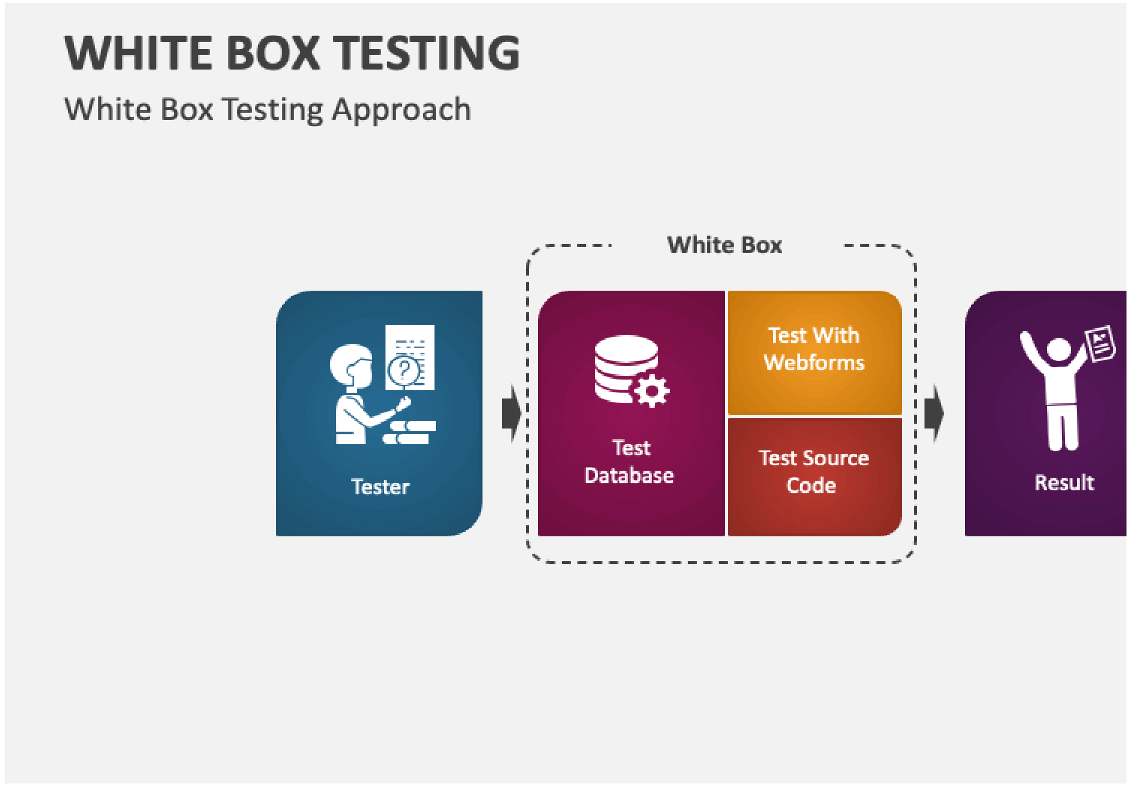
Test case id	Test case name	Test case description	Test steps		Actual result	Test status (P/F)
			Step	Expected result		
Acc04	Edit a recipe	The user will be prompted for the recipe name they wish to edit.	Enter the recipe name along with various units.	Upon completion, a status message is printed and the coffee maker is returned to the waiting state.		
Acc05	Add inventory	Inventory may be added to the machine at any time. (Inventory is measured in integer units.)	Type the inventory: coffee, dairy creamer, water.	Inventory is added to the machine and a status message is printed.		
Acc06	Check inventory	Inventory may be checked at any time.	Enter the units of each item.	System displays the inventory.		
Acc07	Purchase beverage	The user will not be able to purchase a beverage if they do not deposit enough money.	Enter the units and amount.	**1.** System dispensed the change, if user paid more than the price of the beverage. **2.** System returns user's money if there is not enough inventory.		

Advantages Of Black-box Testing

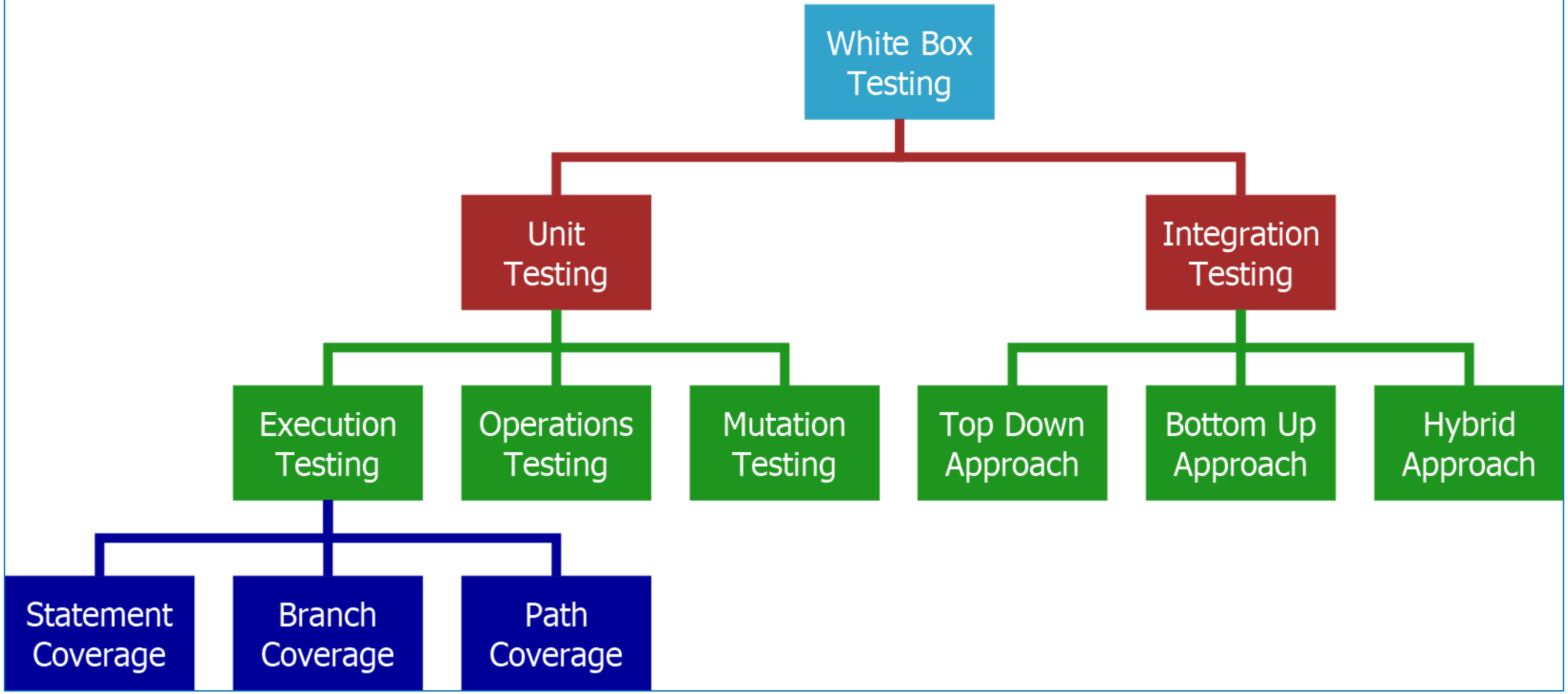
- Tests are performed from the user’s point of view. So, there is higher chance of meeting customer’s expectations.
- There is unbiased testing as both the tester and the developer work independently.
- It is suitable for the testing of very large systems.
- There is no need for any technical knowledge or language specification.
- Test cases can be designed as soon as the requirements are finalized.

White-box Testing

- Testers need access to the source code to identify and evaluate various paths, conditions, and potential issues within the codebase.
- This method aims to ensure that all code paths are executed and that the software functions as intended.



Types of White Box Testing



Practical Challenges with White-box Testing

1. Difficult for software developer to pin-point defects from his own creations: No one would like to point out errors from their own creations. So, usually we select a different test team.

2. Even a completely tested code may not satisfy real customer requirements: Developers do not have a full appreciation and favour to external customer’s requirements or the domain knowledge. This means that even after thorough verification and validation, common user scenarios may get left out.

WHITE BOX TESTING

Disadvantages of White Box Testing

White Box Testing

Since a skilled tester is needed to perform white box testing, the costs are increased.

Sometimes it is impossible to look into every corner to find hidden errors that may create problems as many paths will go untested.

It is difficult to maintain white box testing as specialized tools like code analyzers and debugging tools are required.

3. Experience-based Testing

- **Experience-based** testing is where tests are derived from the tester’s **skill** and **insight** and their **experience** with **similar applications** and technologies.
- When used to **supplement** systematic techniques, these techniques can be useful in identifying special tests not easily captured by formal techniques.
- However, this technique may yield **widely varying degrees of effectiveness**, depending on the testers’ experience.

3. Experience-based Testing
3.1 Error Guessing

- A commonly used experience-based technique is **error guessing**.
- Generally, testers **anticipate defects** based on experience.
- A structured approach to the error guessing technique is to enumerate a list of possible defects and to design tests that attack these defects. This systematic approach is called **fault attack**.

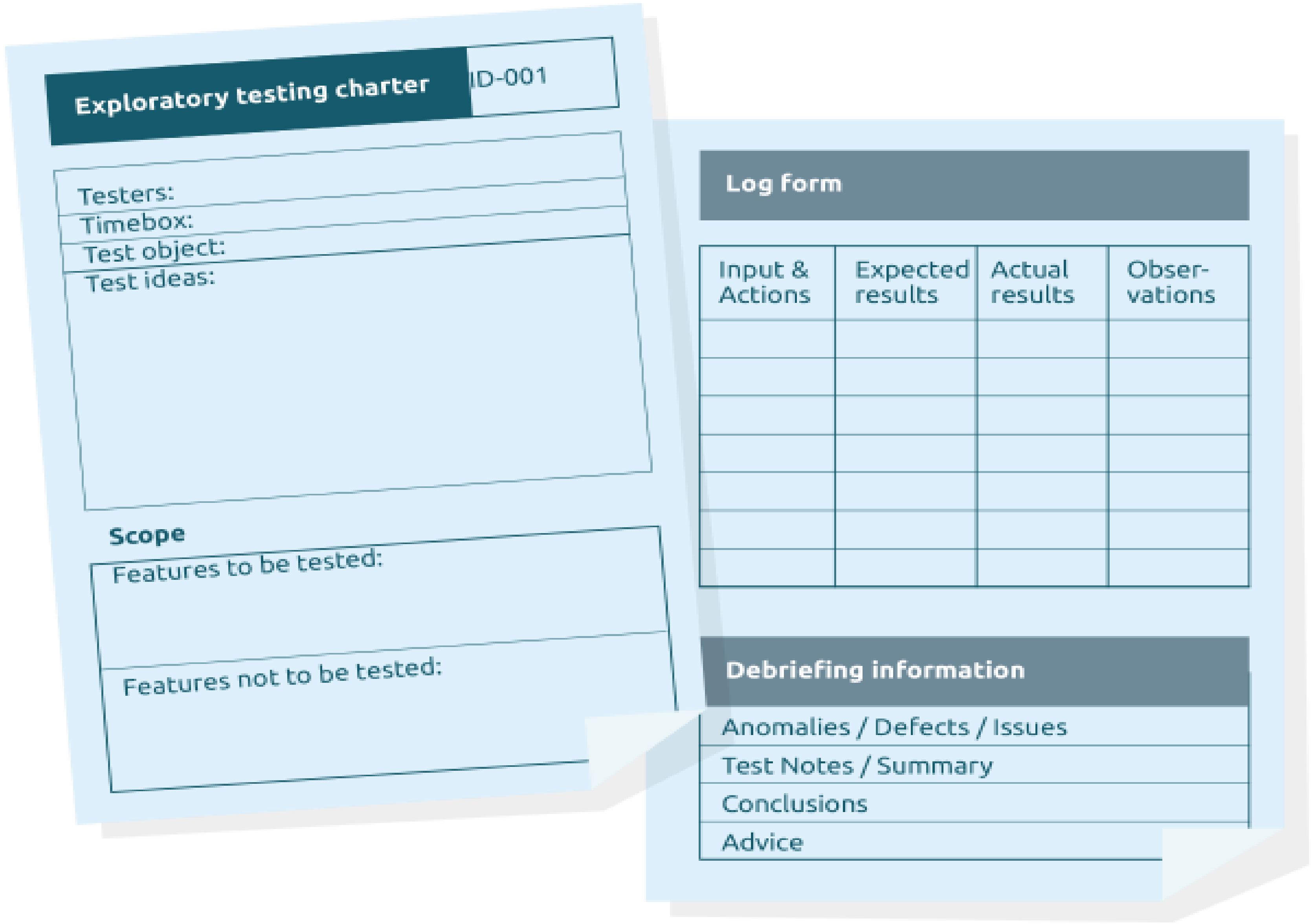
Exploratory Testing

01 LEARNING

02 TEST DESIGN

03 EXECUTION

04 ANALYSIS



Exploratory testing charter

ID-001

Testers:

Timebox:

Test object:

Test ideas:

Scope

Features to be tested:

Features not to be tested:

Log form

Input & Actions	Expected results	Actual results	Obser-vations

Debriefing information

Anomalies / Defects / Issues

Test Notes / Summary

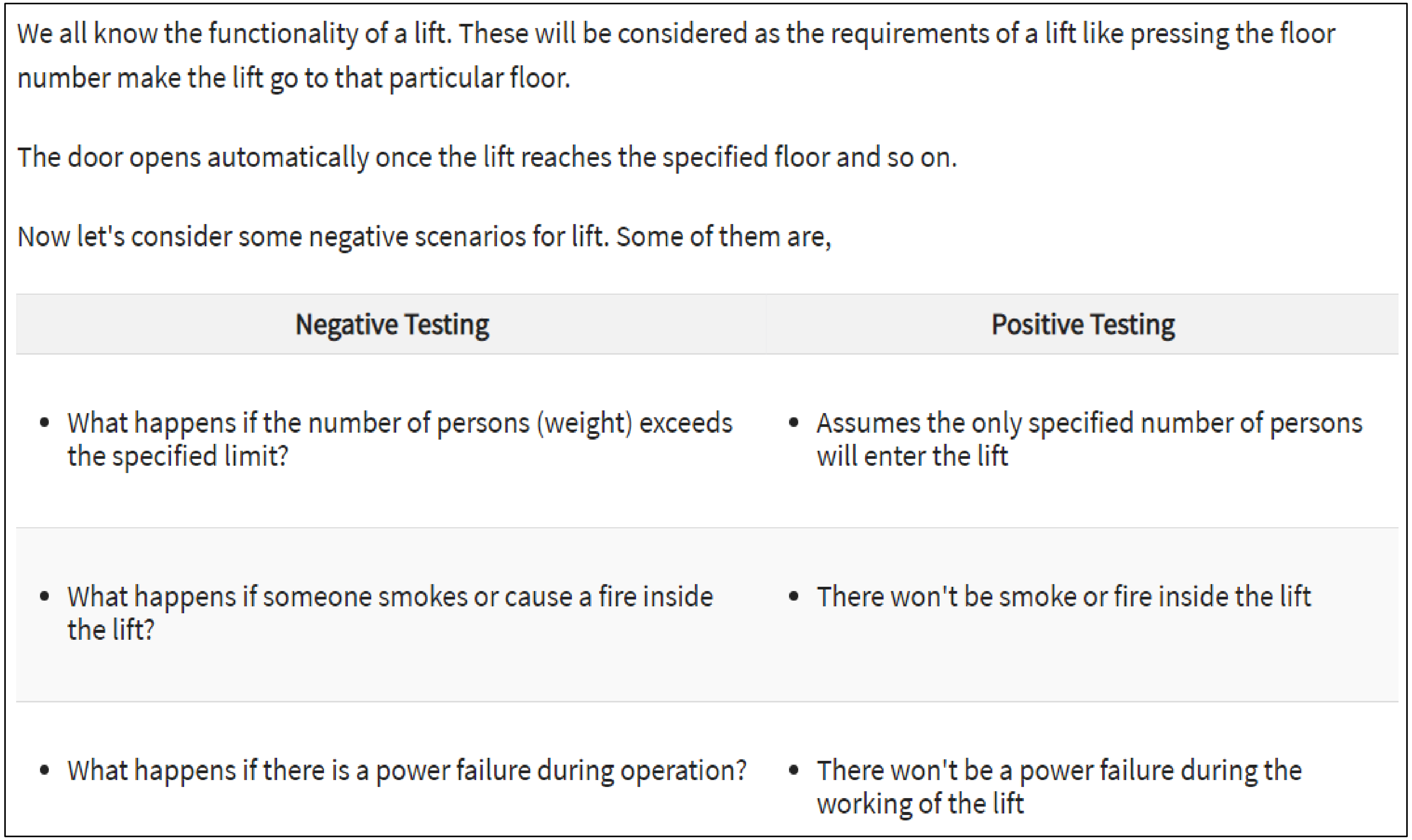
Conclusions

Advice

3. Experience-based Testing
3.2 Exploratory Testing

- Exploratory testing is **concurrent** test design, test execution, test logging and learning, based on a **test charter** containing test objectives, and carried out within **time-boxes**.
- Testing where the tester actively controls the design of the tests as those tests are performed and **uses information gained** while testing to design new and better tests.
- Most **useful where** there are few or inadequate specifications and severe time pressure, or in order to augment or complement other, more formal testing.

Negative vs. Positive Test Cases



We all know the functionality of a lift. These will be considered as the requirements of a lift like pressing the floor number make the lift go to that particular floor.

The door opens automatically once the lift reaches the specified floor and so on.

Now let's consider some negative scenarios for lift. Some of them are,

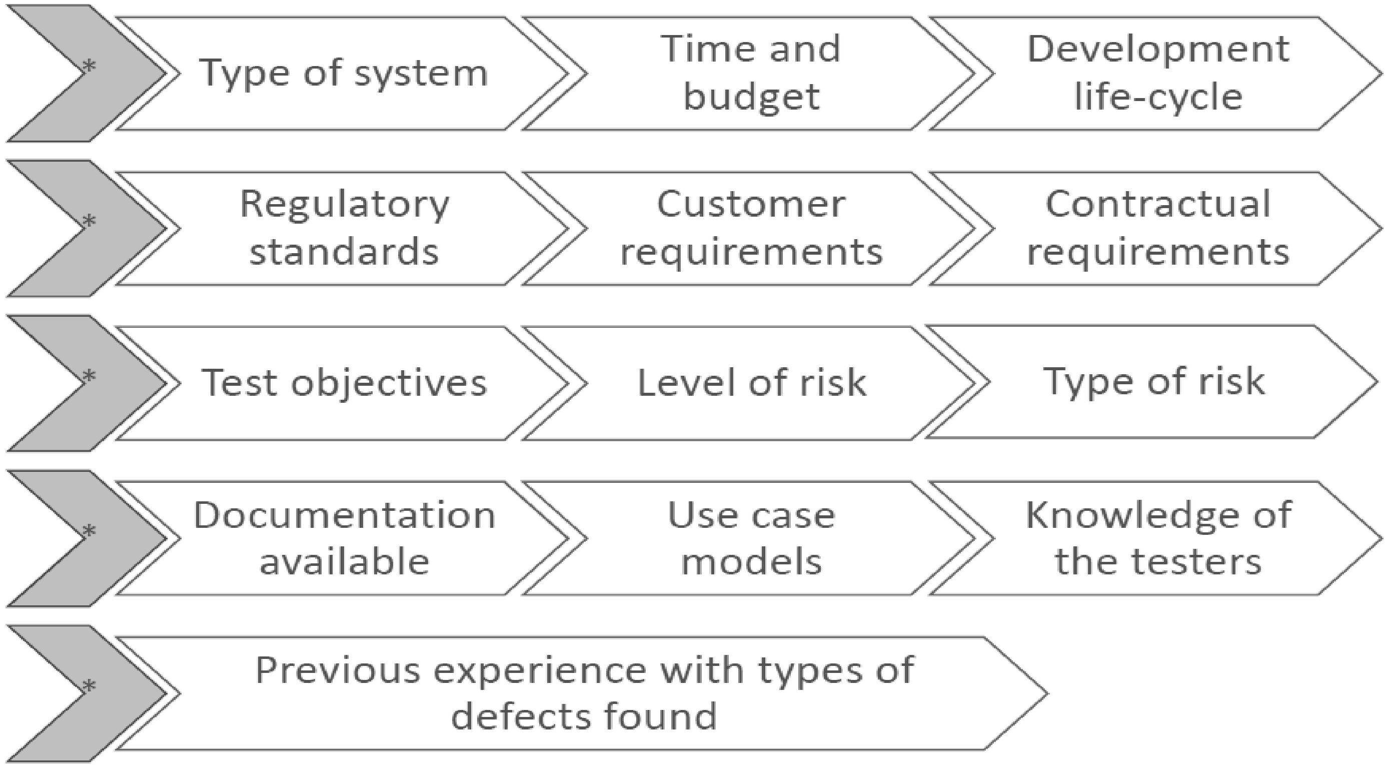
Negative Testing	Positive Testing
• What happens if the number of persons (weight) exceeds the specified limit?	• Assumes the only specified number of persons will enter the lift
• What happens if someone smokes or cause a fire inside the lift?	• There won't be smoke or fire inside the lift
• What happens if there is a power failure during operation?	• There won't be a power failure during the working of the lift

Prioritization of Test Cases

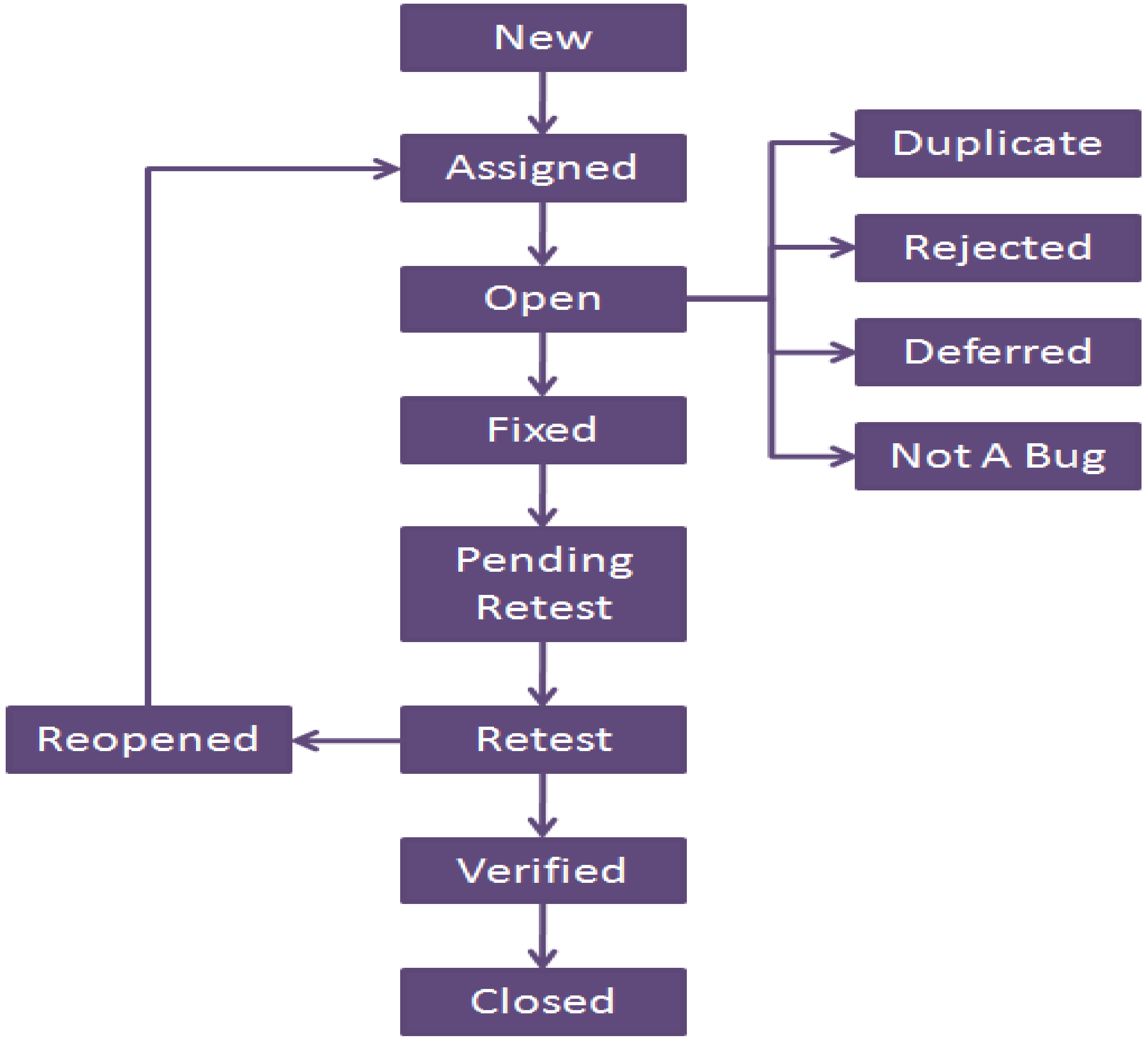
- The goal of prioritization of test cases is to **reduce the set** of test cases based on some rational, criteria, while aiming to select the most appropriate tests.
- The order of test execution is based on the results of risk analysis and test cases covering the most important risks are executed first. Using risk analysis at the beginning of a project highlights the potential problem areas. This helps developers and managers to mitigate the risks.

Choosing test techniques

- The choice of which test techniques to use depends on a number of factors, including:



Bug Life Cycle



Bug / Defect Lifecycle
http://ISTQBExamCertification.com

Bug Life Cycle

Life cycle which a bug goes through during its lifetime, from its discovery to fixation.

- **New:** When a defect is logged and posted for the first time.
- **Assigned:** After the tester has posted the bug, the lead of the tester approves that the bug is genuine, and he assigns the bug to the developing team.
- **Open:** The developer has started analyzing and working on the defect fix. If the developer feels that the defect is not appropriate then it may get transferred to any of the states namely **Duplicate, Deferred or Not a Bug**-based upon the specific reason.

Bug Life Cycle

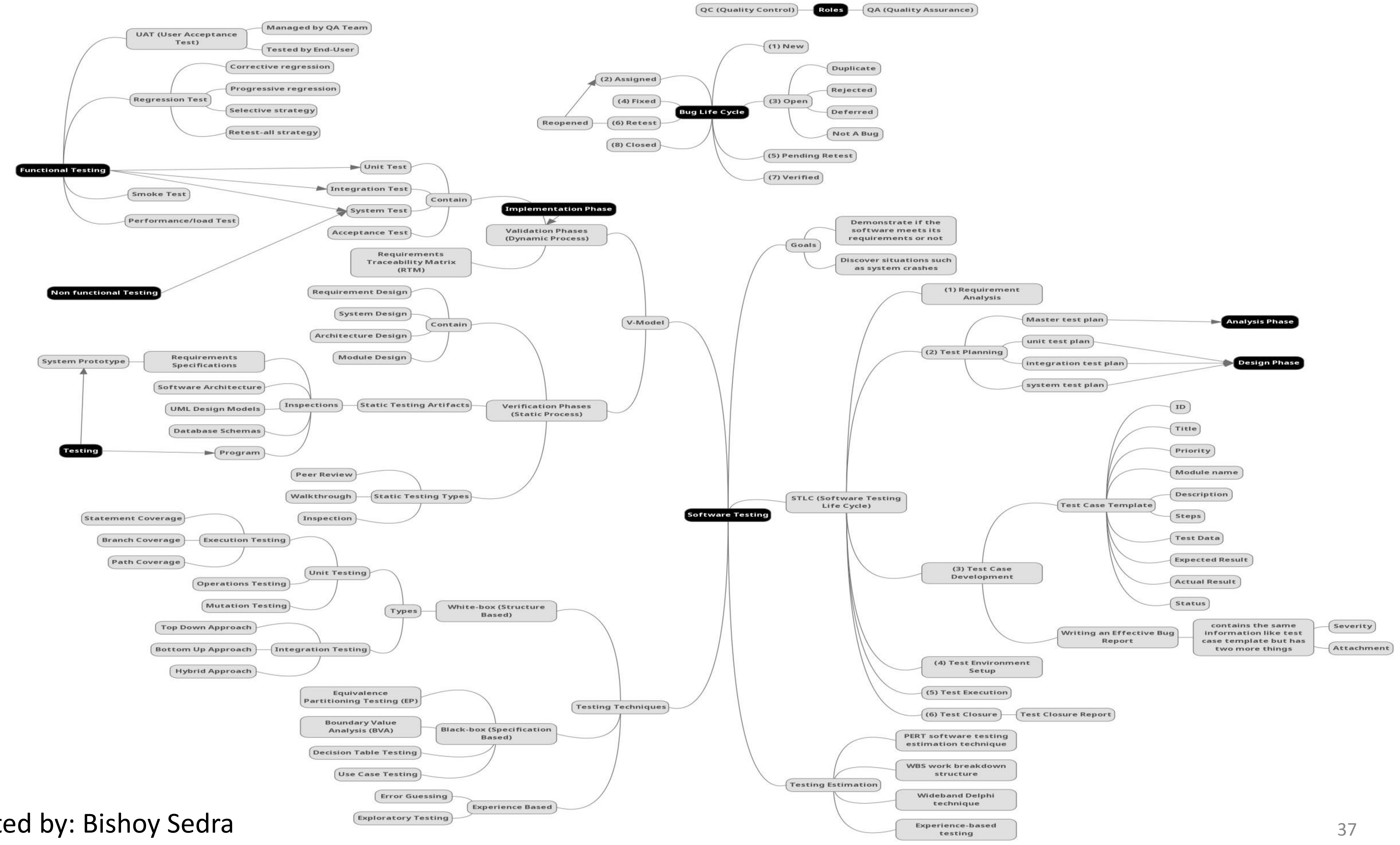
- **Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as ‘Fixed’ and the bug is passed to testing team.
- **Pending retest:** Here the testing is pending on the testers end.
- **Retest:** The tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.

Bug Life Cycle

- **Verified:** The tester tests the bug again after it got fixed by the developer. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “verified”.
- **Reopen:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “reopened”. The bug goes through the life cycle once again.
- **Closed:** If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “closed”. This state means that the bug is fixed, tested and approved.

Bug Statuses

- **Duplicate:** If the bug is repeated twice**.**
- **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug.
- **Deferred:** The bug is expected to be fixed in next releases. The priority of the bug may be low, lack of time for the release or it may not have major effect on software.
- **Not a bug:** If there is no change in the functionality of the application. For an example: If customer asks for some change in the look and feel of the application like change of color of some text then it is not a bug.


Roles
QC (Quality Control)
QA (Quality Assurance)
Bug Life Cycle
(1) New
(2) Assigned
(3) Open
Duplicate
Rejected
Deferred
Not A Bug
(4) Fixed
(5) Pending Retest
(6) Retest
(7) Verified
(8) Closed
Reopened
Functional Testing
UAT (User Acceptance Test)
Managed by QA Team
Tested by End-User
Regression Test
Corrective regression
Progressive regression
Selective strategy
Retest-all strategy
Smoke Test
Performance/load Test
Unit Test
Integration Test
System Test
Acceptance Test
Contain
Non functional Testing
Implementation Phase
Validation Phases (Dynamic Process)
Requirements Traceability Matrix (RTM)
V-Model
Requirement Design
System Design
Architecture Design
Module Design
Contain
Verification Phases (Static Process)
Static Testing Artifacts
Inspections
Requirements Specifications
System Prototype
Software Architecture
UML Design Models
Database Schemas
Program
Testing
Static Testing Types
Peer Review
Walkthrough
Inspection
Software Testing
Goals
Demonstrate if the software meets its requirements or not
Discover situations such as system crashes
STLC (Software Testing Life Cycle)
(1) Requirement Analysis
(2) Test Planning
Master test plan
unit test plan
integration test plan
system test plan
Analysis Phase
Design Phase
(3) Test Case Development
Test Case Template
ID
Title
Priority
Module name
Description
Steps
Test Data
Expected Result
Actual Result
Status
Writing an Effective Bug Report
contains the same information like test case template but has two more things
Severity
Attachment
(4) Test Environment Setup
(5) Test Execution
(6) Test Closure
Test Closure Report
Testing Estimation
PERT software testing estimation technique
WBS work breakdown structure
Wideband Delphi technique
Experience-based testing
Testing Techniques
White-box (Structure Based)
Types
Unit Testing
Execution Testing
Statement Coverage
Branch Coverage
Path Coverage
Operations Testing
Mutation Testing
Integration Testing
Top Down Approach
Bottom Up Approach
Hybrid Approach
Black-box (Specification Based)
Equivalence Partitioning Testing (EP)
Boundary Value Analysis (BVA)
Decision Table Testing
Use Case Testing
Experience Based
Error Guessing
Exploratory Testing

Exercise: Replace With Key Term

- The focus is testing the interactions between components.
- Concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.
- Individual program units, functions or object classes are tested.
- Test that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work.
- Users of a system test the system in their own environment.

Exercises

- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults. Comment.
- List levels of functional testing.
- What is meant by inspection?
- Mention three advantages of white-box testing.
- Describe static testing types.

Exercises

- Demonstrate the bug life cycle.
- User testing is essential, even when comprehensive system and release testing have been carried out. Comment.
- There is no separate user acceptance testing process in agile methods. Comment.
- List factors to choose test techniques.
- Identify software testing life cycle phases.

Differentiate

- System testing and inspection
- Verification and validation
- Integration and system testing
- White and black box testing
- Regression and smoke testing
- Test case and bug report

Replace with Key Term(s)

- Concurrent test design, execution, logging and learning, based on a test charter containing test objectives, and carried out within time-boxes.
- Set of activities for ensuring quality in the processes by which products are developed.
- Tests are derived from the tester’s skill and intuition and their familiarity with similar applications and technologies.
- Enumerate a list of possible defects and to design tests that attack these defects. This systematic approach is called fault attack.

Replace with Key Term(s)

- A document that gives a summary of all the tests conducted during STLC and gives a detailed analysis of bugs removed and errors found.
- Testing the system to check that changes have not ‘broken’ previously working code.
- Precise and compact way to model complicated logic. It is ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions.
- Customers test a system to decide whether or not it is ready to be accepted and deployed in the real environment.

Complete

- Strength of ----- testing is that it creates combinations of conditions that otherwise might not have been exercised during testing.
- The purpose of a ----- is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.
- Black-box testing techniques include -----, ----- and -----.
- In -----, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be processed in the same way.
- Acceptance testing usually uses ----- testing technique.
- The primary goals of the system testing process are ----- and -----.

References

- https://estb.org.eg/English/Pages/Resources/ISTQBSyllabi.aspx
- R. Chopra. Software Quality Assurance: A Self-Teaching Introduction. ISBN: 978-1-683921-68-4
- https://www.testorigen.com/exploring-the-latest-advances-in-white-box-testing-techniques/

software
test
testing
code
requirements
edit
development
process
system
may
tests
used
also