UNIT III

CHAPTER 3

Test Case Design Techniques

University Prescribed Syllabus for the Academic Year 2022-2023

Software Testing Methodologies : White Box Testing, Black Box Testing, Grey Box Testing. Test Case Design Techniques : Static Techniques : Informal Reviews, Walkthroughs, Technical Reviews, Inspection. Dynamic Techniques: Structural Techniques: Statement Coverage Testing, Branch Coverage Testing, Path Coverage Testing, Conditional Coverage Testing, Loop Coverage Testing Black Box Techniques: Boundary Value Analysis, Equivalence Class Partition, State Transition Technique, Cause Effective Graph, Decision Table, Use Case Testing, Experienced Based Techniques: Error guessing, Exploratory testing

Levels of Testing : Functional Testing: Unit Testing, Integration Testing, System Testing, User Acceptance Testing, Sanity/Smoke Testing, Regression Test, Retest. Non-Functional Testing: Performance Testing, Memory Test, Scalability Testing, Compatibility Testing, Security Testing, Cookies Testing, Session Testing, Recovery Testing, Installation Testing, Adhoc Testing, Risk Based Testing, I18N Testing, L1ON Testing, Compliance Testing.

3.1 Software Testing Methodologies 3-3
GQ. Explain Software Testing Methodologies. 3-3
3.1.1 White Box Testing? 3-3
3.1.2 Black Box Testing 3-3
3.1.3 Grey Box Testing 3-3
3.2 Test Case Design Techniques 3-3
GQ. Explain Test Case Design Techniques with example. 3-3
GQ. What is a Test Case ? 3-4
3.2.1 Static Techniques 3-4
GQ. What is subject to static testing ? 3-4
3.2.2 Informal Reviews 3-6
GQ. What is informal review in software testing ? 3-6
3.2.3 Walkthrough 3-6
GQ. What are Differences between Inspection and Walkthrough ? 3-6
3.2.4 Technical Reviews 3-7
GQ. What is Technical review in software testing ? 3-7
3.2.5 Inspection 3-7
3.3 Dynamic Techniques 3-8
GQ. Explain Dynamic Techniques. 3-8
GQ. What does dynamic testing do ? 3-8
3.3.1 Structural Techniques 3-9
GQ. Explain structural Techniques with Example. 3-9
GQ. Explain types of Structural Testing 3-10
3.3.2 Statement Coverage Testing 3-11
GQ. Explain statement Coverage Testing with source code structure ? 3-11
3.3.3 Branch Coverage Testing 3-13
GQ. Explain Branch coverage testing with example. 3-13
GQ. How to calculate Branch coverage ? 3-13
3.3.4 Path Coverage Testing 3-15
GQ. What is path coverage testing technique ? 3-15
3.3.5 Conditional Coverage Testing 3-15
GQ. What is condition coverage testing ? 3-15
3.3.6 Loop Coverage Testing 3-17
GQ. Why Loop Testing ? 3-17
GQ. How to do Loop Testing – Complete Methodology ? 3-18
3.4 Black Box Techniques 3-19

GQ. Explain Black Box Techniques. 3-19
3.4.1 Boundary Value Analysis 3-19
3.4.2 Equivalence Class Partition....3-19
3.4.3 State Transition Technique 3-19
3.4.4 Cause Effective Graph 3-20
3.4.5 Decision Table 3-22
GQ. Explain Decision Table with example. 3-22
3.4.6 Use Case Testing 3-23
GQ. Explain Use case Testing. 3-23
3.5 Experienced Based Techniques....3-24
GQ. Explain Experienced Based Techniques. 3-24
GQ. What are Different types of experience-based testing techniques? 3-24
3.5.1 Error Guessing 3-24
GQ. Explain Error Guessing with example based testing. 3-24
3.5.2 Exploratory Testing 3-25
GQ. Write a note on Exploratory Testing ? 3-25
3.6 Levels of Testing 3-25
3.6.1 Functional Testing....3-25
GQ. Write a note on Functional Testing ?....3-25
3.6.1.1 Unit Testing 3-26
GQ. Explain Unit Testing with example....3-26
3.6.1.2 Integration Testing 3-27
GQ. Write a note on Integration Testing with example ? 3-27
GQ. What are the approaches used in Integration Testing? 3-27
3.6.1.3 System Testing 3-28
GQ. Write a note on System Testing ? 3-28
3.6.1.4 User Acceptance Testing....3-28
GQ. Explain types of Acceptance Testing....3-28
3.6.2 Sanity/Smoke Testing 3-29
GQ. Explain Sanity / Smoke Testing with example.....3-29
GQ. What is Smoke testing and Sanity testing ? 3-29
GQ. What are important differences : Smoke vs Sanity testing ?....3-31
3.6.3 Regression Test....3-32
GQ. What is Regression Testing ? Definition, Tools & How to Get Started ? 3-32
GQ. What is regression testing ? 3-32
GQ. When to apply regression testing ? 3-32
GQ. Why is regression testing important ? 3-33
GQ. How to perform regression testing ?....3-33
3.6.4 Retest....3-34
GQ. What is retesting ?....3-34
GQ. What is Regression testing vs. retesting: key differences? 3-34
3.7 Non-Functional Testing....3-34
3.7.1 Performance Testing....3-35
GQ. Explain performance testing with example....3-35
3.7.2 Memory Test....3-35
3.7.3 Scalability Testing....3-35
GQ. Why does Scalability Testing? 3-36
GQ. What to test in Scalability Testing? 3-36
GQ. How to do Scalability Testing? 3-36
GQ. Write a note on Compatibility Testing. 3-37
3.7.4 Compatibility Testing....3-37
3.7.5 Security Testing 3-37
3.7.6 Cookies Testing 3-38
GQ. Write a note on : Cookies Testing with example..3-38
GQ. What Exactly Is a Cookie ? 3-38
GQ. What is the purpose of cookies ? 3-38
GQ. What exactly is Cookie Testing ? 3-38
GQ. What Is the Function of Cookies ? 3-39
GQ. What is the Cookie's Content ? 3-39
GQ. Where do cookies get saved ? 3-39
3.7.7 Session Testing 3-41
GQ. Write a note on session testing. 3-41
3.7.8 Recovery Testing....3-42
GQ. Write a note on Recovery Testing. 3-42
3.7.9 Installation Testing....3-42
3.7.10 Ad hoc Testing....3-42
GQ. Explain Ad hoc Testing. 3-42
GQ. When execute Adhoc Testing ? 3-43
3.7.11 Risk Based Testing....3-43
GQ. Explain Risk Based Testing with example. 3-43
GQ. How to perform risk based testing ?....3-44
3.7.12 I18N Testing 3-44
GQ. Explain I18N Testing. 3-44
3.7.13 L1ON Testing....3-45
GQ. Explain L1ON Testing. 3-45
3.7.14 Compliance Testing 3-46
GQ. Explain Compliance Testing.. 3-46
GQ. When to use Compliance Testing? 3-46
GQ. How to do a compliance check? 3-47
GQ. What Is a Decision Table ? 3-48
GQ. Give short note on : Decision Table Examples. ..3-49
• **Chapter Ends** 3-52

3.1 SOFTWARE TESTING METHODOLOGIES

GQ. Explain Software Testing Methodologies.

- Methodologies can be considered as the set of testing mechanisms used in the software development lifecycle from Unit Testing to System Testing. Selecting an appropriate testing methodology is considered to be the core of the testing process.
- With the means of security, compatibility, and usability, a software product should be tested by using the proper testing methodology.
- Basically, there are 3 testing techniques that are used for testing. They are White Box Testing, Black Box Testing, and Grey Box Testing. Each of the testing techniques is briefed below for your better understanding.

3.1.1 White Box Testing?

The white box testing technique is used to examine the program structure and business logic, it validates the code or program of an application. It is also called *Clear Box Testing, Glass Box Testing, or Open Box Testing*.

White Box Testing Techniques include :

- **Statement Coverage :** Examines all the programming statements.
- **Branch Coverage :** Series of running tests to ensure if all the branches are tested.
- **Path Coverage :** Tests all the possible paths to cover each statement and branch.

3.1.2 Black Box Testing

- The Black Box testing method is used to test the functionality of an application based on the requirement specification.
- Unlike White Box Testing it does not focus on the internal structure/code of the application.

Black Box Techniques include

(1) Boundary Value analysis
(2) Equivalence Partitioning (Equivalence Class Partitioning)
(3) Decision Tables
(4) Domain Tests
(5) State Models
(6) Exploratory Testing (Requires less preparation and also helps to find the defects quickly).

3.1.3 Grey Box Testing

- This method of testing is performed with less information about the internal structure of an application.
- Generally, this is performed like Black Box Testing only but for some critical areas of application, White Box Testing is used.

3.2 TEST CASE DESIGN TECHNIQUES

GQ. Explain Test Case Design Techniques with example.

- The most significant and crucial phase in the development of software is its testing phase.
- Not only does testing helps to determine the quality of a product but it also allows one to modify and upgrade the product in terms of end-user friendliness and usability.
- In this article, we will address the fundamental notion of test case design techniques of various kinds.
- Test cases are the fundamental building blocks which when put together form the testing phase.

GQ. What is a Test Case ?

- They are often a pre-defined set of instructions addressing the steps to be taken in order to determine whether or not the end product exhibits the desired outcome. These instructions may include predefined sets of inputs, conditions along with their respective end results.
- However, in order to be through with one's testing one could often end up with too many test cases. To avoid such scenarios, one should find the best test case design technique as per ones requirement so as to reduce a significant number of test cases.
- These test case design techniques help create effective test cases covering the various features that determine the quality and value of a product.

3.2.1 Static Techniques

- Static testing is a software testing method that involves the examination of a program, along with any associated documents, but does not require the program to be executed.
- Dynamic testing, the other main category of software testing, involves interaction with the program while it runs. The two methods are frequently used together to try to ensure the basic functionalities of a program.
- Instead of executing the code, static testing is a process of checking the code and designing documents and requirements before it's run in order to find errors.
- The main goal is to find flaws in the early stages of development. It is normally easier to find the sources of possible failures this way.

GQ. What is subject to static testing ?

- It's common for code, design documents and requirements to be static tested before the software is run in order to find errors.
- Anything that relates to functional requirements can also be checked.
- More specifically, the process will involve reviewing written materials that, altogether, give a wider view of the tested software application as a whole.
- Some examples of what's tested include :
  - Requirement specifications
  - User documents
  - Source code
  - User documents
  - Design documents
  - Web page content
  - Test cases, test data and test scripts
  - Specification and matrix documents

Benefits of static testing

Some benefits of static testing include :

- Early detection and correction of any coding errors.
- Reduces cost in early stages of development -- in terms of the amount of rework needed to fix any errors.
- Reduced timescales for development.
- Feedback received in this stage will help improve the overall functioning of the software. Once other testing types like dynamic testing start, there won't be as many errors found. This means code is overall more maintainable.
- This process will also help give developers a better idea of the quality issues found in the software.
- With automated tools, this process can be quite fast to review code and other documents.
- Static testing can also boost the amount of communication between teams.

Static testing techniques

- Static testing is carried out with two different steps or techniques -- review and static analysis.
- Static review is typically carried out to find and remove errors and ambiguities found in supporting documents. Documents reviewed include software requirements specifications, design and test cases.
- The documents can be reviewed in multiple ways, such as in a walkthrough, peer review or as an inspection.
- The next step, static analysis, is where the code written by developers is analysed. The evaluation is done in order to find any structural defects that may lead to errors when run.
- Some other techniques used while performing static testing include use case requirements validation, functional requirement validation, architecture review and field dictionary validation.
- Use case requirements ensure possible end-user actions are properly defined.
- Functional requirements will identify any necessary requirements for the software. Reviews of architecture means business-level processes are analysed. Field dictionary validation will analyse UI fields.
- Static testing may also be conducted either manually or through automation with the use of various software testing tools.

Types of static testing reviews

Reviews, the first step in static testing, can be conducted in numerous ways. Reviews are performed to find and remove errors found in supporting documents. This process can be carried out in four different ways :

1. **Informal :** Informal reviews will not follow any specific process to find errors. Co-workers can review documents and provide informal comments.
2. **Walkthrough :** The author of whichever document is being reviewed will explain the document to their team. Participants will ask questions, and any notes are written down.
3. **Inspection :** A designated moderator will conduct a strict review as a process to find defects.
4. **Technical/peer reviews :** Technical specifications are reviewed by peers in order to detect any errors.

Differences between static testing vs. dynamic testing

- Dynamic testing is a method of assessing the feasibility of a software program by giving input and examining output. The dynamic method requires that the code be compiled and run.
- The biggest and most notable difference between static and dynamic testing is that dynamic testing requires code to be executed. Functional behaviour and performance are checked to confirm if the code works properly.
- Static testing will analyse the code, requirement documents and design documents, while dynamic testing will look at the functional behaviour of software systems like memory usage and performance.
- Static testing essentially gives an assessment of code, while dynamic testing will try and find active bugs. The cost of finding code defects in dynamic testing will normally cost more in terms of time and money than in static testing.
- The two types of testing are meant not meant to be mutually exclusive, however. Ideally, they should be used together. Static testing is about the prevention of defects, whereas dynamic testing is about finding active defects.
- Types of dynamic testing include unit testing, integration testing, system testing and performance testing.

3.2.2 Informal Reviews

GQ. What is informal review in software testing ?

- Informal review is **an informal static test technique performed on any kind of requirements, design, code, or project plan.**
- During informal review, the work product is given to a domain expert and the feedback/comments are reviewed by the owner/author.
- Informal reviews are applied many times during the early stages of the life cycle of the document.
- A two person team can conduct an informal review. In later stages these reviews often involve more people and a meeting.
- The goal is to keep the author and to improve the quality of the document. The most important thing to keep in mind about the informal reviews is that they are **not documented.**

Informal reviews

- **Informal reviews** take place between two or three people.
- The review conference is scheduled at their convenience.
  - This meeting is generally scheduled during the free time of the team members.
  - There is no planning for the meeting.
  - If any errors occur, they are not corrected in the informal reviews.
  - There is no guidance from the team.
  - This review is less effective compared to the formal review.

3.2.3 Walkthrough

- Walkthrough is a method of conducting informal group/individual review. In a walkthrough, author describes and explain work product in a informal meeting to his peers or supervisor to get feedback. Here, validity of the proposed solution for work product is checked.
- It is cheaper to make changes when design is on the paper rather than at time of conversion. Walkthrough is a static method of quality assurance. Walkthrough are informal meetings but with purpose.

GQ. What are Differences between Inspection and Walkthrough ?

Sr. No.	Inspection	Walkthrough
1.	It is formal.	It is informal.
2.	Initiated by project team.	Initiated by author.
3.	A group of relevant persons from different departments participate in the inspection.	Usually team members of the same project take participation in the walkthrough. Author himself acts walkthrough leader.
4.	Checklist is used to find faults.	No checklist is used in the walkthrough.
5.	Inspection processes includes overview, preparation, inspection, and rework and follow up.	Walkthrough process includes overview, little or no preparation, little or no preparation examination (actual walkthrough meeting), and rework and follow up.
6.	Formalized procedure in each step.	No formalized procedure in the steps.

Sr. No.	Inspection	Walkthrough
7.	Inspection takes longer time as list of items in checklist is tracked to completion.	Shorter time is spent on walkthrough as there is no formal checklist used to evaluate program.
8.	Planned meeting with the fixed roles assigned to all the members involved.	Unplanned
9.	Reader reads product code. Everyone inspects it and comes up with detects.	Author reads product code and his teammate comes up with the defects or suggestions.
10.	Recorder records the defects.	Author make a note of defects and suggestions offered by teammate.
11.	Moderator has a role that moderator making sure that the discussions proceed on the productive lines.	Informal, so there is no moderator.

3.2.4 Technical Reviews

GQ. What is Technical review in software testing ?

- It is less formal review
- It is led by the trained moderator but can also be led by a technical expert
- It is often performed as a peer review without management participation
- **Defects** are found by the experts (such as architects, designers, key users) who focus on the content of the document.
- In practice, technical reviews vary from quite informal to very formal

The goals of the technical review are :

1. To ensure that an early stage the technical concepts are used correctly
2. To access the value of technical concepts and alternatives in the product
3. To have consistency in the use and representation of technical concepts
4. To inform participants about the technical content of the document

3.2.5 Inspection

- An inspection is defined as formal, rigorous, in depth group review designed to identify problems as close to their point of origin as possible. Inspections improve reliability, availability, and maintainability of software product.
- Anything readable that is produced during the software development can be inspected. Inspections can be combined with structured, systematic testing to provide a powerful tool for creating defect-free programs.
- Inspection activity follows a specified process and participants play well-defined roles. An inspection team consists of three to eight members who plays roles of moderator, author, reader, recorder and inspector.
- **For example,** designer can acts as inspector during code inspections while a quality assurance representative can act as standard enforcer.

Stages in the Inspections process

1. **Planning :** Inspection is planned by moderator.
2. **Overview meeting :** Author describes background of work product.

3. **Preparation :** Each inspector examines work product to identify possible defects.
4. **Inspection meeting :** During this meeting, reader reads through work product, part by part and inspectors points out the defects for every part.
5. **Rework :** Author makes changes to work product according to action plans from the inspection meeting.
6. **Follow-up :** Changes made by author are checked to make sure that everything is correct.

3.3 DYNAMIC TECHNIQUES

GQ. Explain Dynamic Techniques.

- **Dynamic Testing** is a software testing method used to test the dynamic behaviour of software code.
- The main purpose of dynamic testing is to test software behaviour with dynamic variables or variables which are not constant and finding weak areas in software runtime environment.
- The code must be executed in order to test the dynamic behaviour.
- We all know that Testing is verification and validation, and it takes 2 Vs to make testing complete.
- Out of the 2 Vs, Verification is called a Static testing and the other "V", Validation is known as dynamic testing.

☞ Dynamic Testing Example

- Let's understand How to do Dynamic Testing with an example:
- Suppose we are testing a Login Page where we have two fields say "Username" and "Password" and the Username is restricted to Alphanumeric.
- When the user enters Username as "xyz", the system accepts the same. Whereas when the user enters as xyz @ 123 then the application throws an error message. This result shows that the code is acting dynamically **based on the user input.**
- Dynamic testing is when you are working with the actual system by providing an input and comparing the actual behaviour of the application to the expected behaviour. In other words, working with the system with the intent of finding errors.
- So based on the above statements we can say or conclude that dynamic testing is a process of validating software applications as an end user under different environments to build the right software.

GQ. What does dynamic testing do ?

- The main aim of the Dynamic tests is to ensure that software works properly during and after the installation of the software ensuring a stable application without any major flaws (this statement is made because no software is error free, testing only can show presence of defects and not absence)
- The main purpose of the dynamic test is to ensure consistency to the software; lets discuss this with an example.
- In a Banking Application, we find different screens like My Accounts Section, Funds Transfer, Bill Pay, etc.. All these screens contain amount field which accepts some characters.
- Let's say My Accounts field displays amount as **25,000** and Funds Transfer as **$25,000** and Bill pay screen as **$25000** though the amount is the same, the way amount is displayed is not the same hence making the software no consistent.
- Consistency is not only limited to the functionality it also refers to different standards like performance, usability, compatibity etc, hence it becomes very important to perform Dynamic Testing.

☞ Types of Dynamic Testing

- Dynamic Testing is classified into two categories
  1. White Box Testing
  2. Black Box Testing

- The below pictorial representation gives us an idea about types of Dynamic Testing, Levels of Testing, etc.

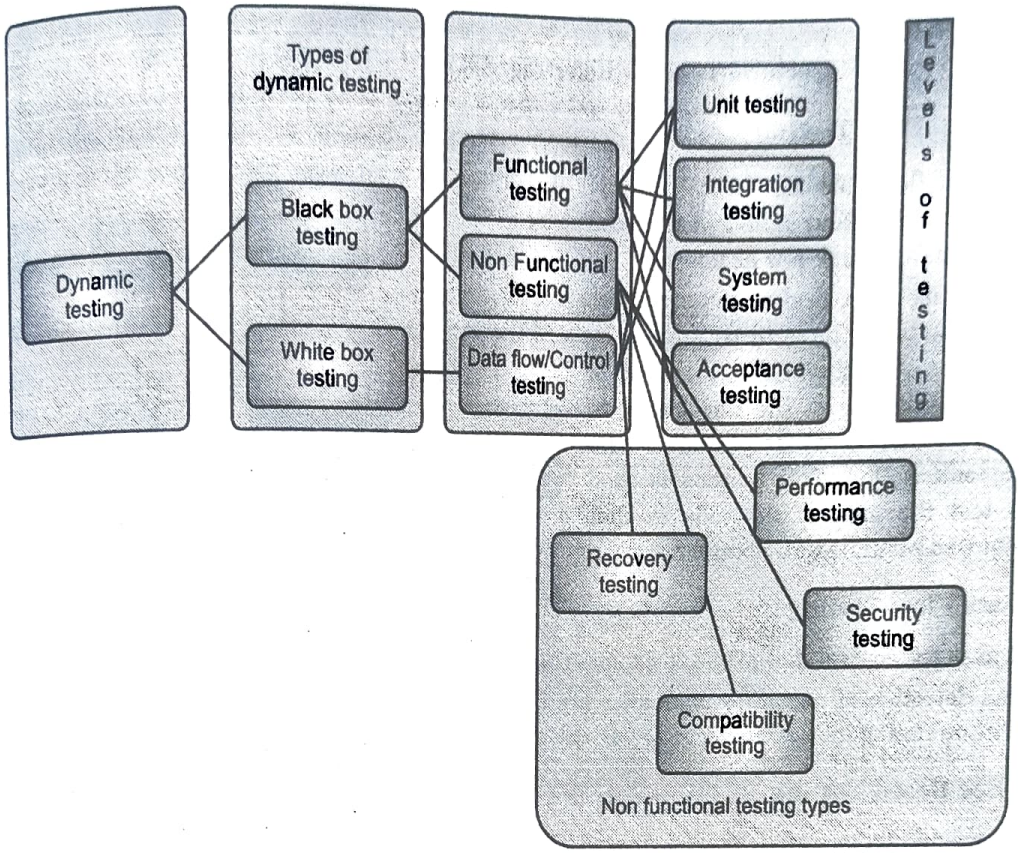


Fig. 3.3.1 : Dynamic Testing

☞ **Let us discuss briefly each type of testing and its intended purpose**

1. **White Box Testing** White Box Testing is a software testing method in which the internal structure/ design is known to the tester. The main aim of White Box testing is to check on how System is performing based on the code. It is mainly performed by the Developers or White Box Testers who has knowledge on the programming.

2. **Black Box Testing** Black Box Testing is a method of testing in which the internal structure/ code/design is **NOT** known to the tester. The main aim of this testing to verify the functionality of the system under test and this type of testing requires to execute the complete test suite and is mainly performed by the Testers, and there is no need of any programming knowledge.

3.3.1 Structural Techniques

GQ. Explain structural Techniques with Example.

- **Structural testing** is a type of software testing which uses the internal design of the software for testing or in other words the software testing which is performed by the team which knows the development phase of the software, is known as structural testing.
- Structural testing is basically related to the internal design and implementation of the software i.e. it involves the development team members in the testing team. It basically tests different aspects of the software according to its types. Structural testing is just the opposite of behavioral testing.

☞ Types of Structural Testing

GQ. Explain types of Structural Testing.

There are 4 types of Structural Testing: (Refer Fig. 3.3.2)

1. Control Flow Testing

- Control flow testing is a type of structural testing that uses the programs's control flow as a model.
- The entire code, design and structure of the software have to be known for this type of testing. Often this type of testing is used by the developers to test their own code and implementation. This method is used to test the logic of the code so that required result can be obtained.

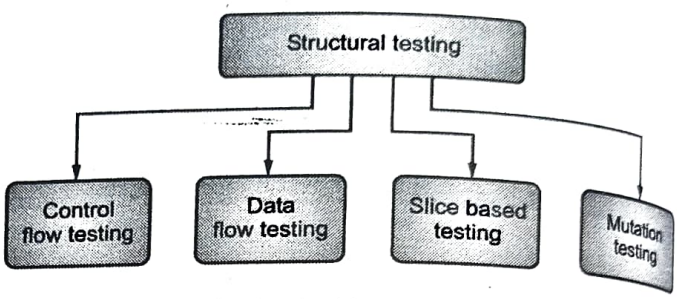


Fig. 3.3.2 : Structural testing

2. Data Flow Testing

- It uses the control flow graph to explore the unreasonable things that can happen to data.
- The detection of data flow anomalies are based on the associations between values and variables. Without being initialized usage of variables. Initialized variables are not used once.

3. Slice Based Testing

It was originally proposed by Weiser and Gallagher for the software maintenance. It is useful for software debugging, software maintenance, program understanding and quantification of functional cohesion. It divides the program into different slices and tests that slice which can majorly affect the entire software.

4. Mutation Testing

- Mutation Testing is a type of Software Testing that is performed to design new software tests and also evaluate the quality of already existing software tests.
- Mutation testing is related to modification a program in small ways. It focuses to help the tester develop effective tests or locate weaknesses in the test data used for the program.

☞ Advantages of Structural Testing

(1) It provides thorough testing of the software.
(2) It helps in finding out defects at an early stage.
(3) It helps in elimination of dead code.
(4) It is not time consuming as it is mostly automated.

☞ Disadvantages of Structural Testing

(1) It requires knowledge of the code to perform test.
(2) It requires training in the tool used for testing.
(3) Sometimes it is expensive.

☞ Structural Testing Tools

- JBehave
- Cucumber
- Junit
- Cfix

3.3.2 Statement Coverage Testing

GQ. Explain statement Coverage Testing with source code structure.

- Statement coverage is one of the widely used software testing. It comes under white box testing.
- Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.
- Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

- In white box testing, concentration of the tester is on the working of internal source code and flow chart or flow graph of the code.
- Generally, in the internal source code, there is a wide variety of elements like operators, methods, arrays, looping, control statements, exception handlers, etc.
- Based on the input given to the program, some code statements are executed and some may not be executed.
- The goal of statement coverage technique is to cover all the possible executing statements and path lines in the code.
- Let's understand the process of calculating statement coverage by an example:
- Here, we are taking source code to create two different scenarios according to input values to check the percentage of statement coverage for each scenario.

Source Code Structure

- Take input of two values like a = 0 and b = 1.
- Find the sum of these two values.
- If the sum is greater than 0, then print "This is the positive result."
- If the sum is less than 0, then print "This is the negative result."

```
input (int a, int b)
{
Function to print sum of these integer values (sum = a+b)
If (sum>0)
{
Print (This is positive result)
} else
{
Print (This is negative result)
}
}
```

- So, this is the basic structure of the program, and that is the task it is going to do.
- Now, let's see the two different scenarios and calculation of the percentage of Statement Coverage for given source code.

☞ **Scenario 1 : If a = 5, b = 4**

```
print (int a, int b) {
int sum = a+b;
if (sum>0)
print ("This is a positive result")
else
print ("This is negative result")
}
```

- In scenario 1, we can see the value of sum will be 9 that is greater than 0 and as per the condition result will be **"This is a positive result."** The statements highlighted in yellow color are executed statements of this scenario.
- To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 5.

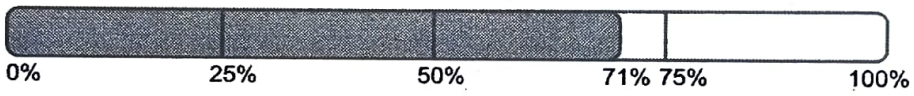
$$\text{Total number of statements} = 7$$

$$\text{Number of executed statements} = 5$$

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

$$\text{Statement coverage} = 5/7*100$$

$$= 500/7 = 71\ \%$$



Likewise, in scenario 2

☞ **Scenario 2 : If A = -2, B = -7**

```
print (int a, int b) {
int sum = a+b;
if (sum>0)
print ("This is a positive result")
else
print ("This is negative result")
}
```

- In scenario 2, we can see the value of sum will be -9 that is less than 0 and as per the condition; result will be **"This is a negative result."** The statements highlighted in yellow colour are executed statements of this scenario.
- To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 6.

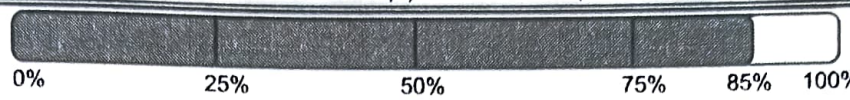
$$\text{Total number of statements} = 7$$

$$\text{Number of executed statements} = 6$$

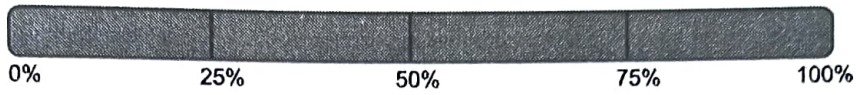
$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

$$\text{Statement coverage} = 6/7*100 \text{
}$$

$$= 600/7 = 85\ \%$$



- But, we can see all the statements are covered in both scenario and we can consider that the overall statement coverage is 100%.



- So, the statement coverage technique covers dead code, unused code, and branches.

3.3.3 Branch Coverage Testing

GQ. Explain Branch coverage testing with example.

- Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
- Branch coverage technique is a white box testing technique that ensures that every branch of each decision point must be executed.
- However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two.
- Decision coverage technique covers all branches of each decision point whereas branch testing covers all branches of every decision point of the code.
- In other words, branch coverage follows decision point and branch coverage edges. Many different metrics can be used to find branch coverage and decision coverage, but some of the most basic metrics are: finding the percentage of program and paths of execution during the execution of the program.
- Like decision coverage, it also uses a control flow graph to calculate the number of branches.

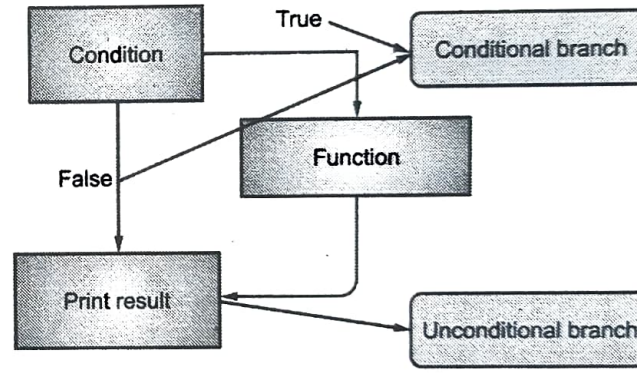


Fig. 3.3.3 : Control Flow Graph

GQ. How to calculate Branch coverage.

- There are several methods to calculate Branch coverage, but path finding is the most common method.
- In this method, the number of paths of executed branches is used to calculate Branch coverage.
- Branch coverage technique can be used as the alternative of decision coverage. Somewhere, it is not defined as an individual technique, but it is distinct from decision coverage and essential to test all branches of the control flow graph.

☞ **Let's understand it with an example :**

1. Read X
2. Read Y
3. IF X + Y > 100 THEN
4. Print "Large"
5. ENDIF
6. If X + Y<100 THEN
7. Print "Small"
8. ENDIF

This is the basic code structure where we took two variables X and Y and two conditions. If the first condition is true, then print "Large" and if it is false, then go to the next condition. If the second condition is true, then print "Small."

☞ **Control flow graph of code structure**

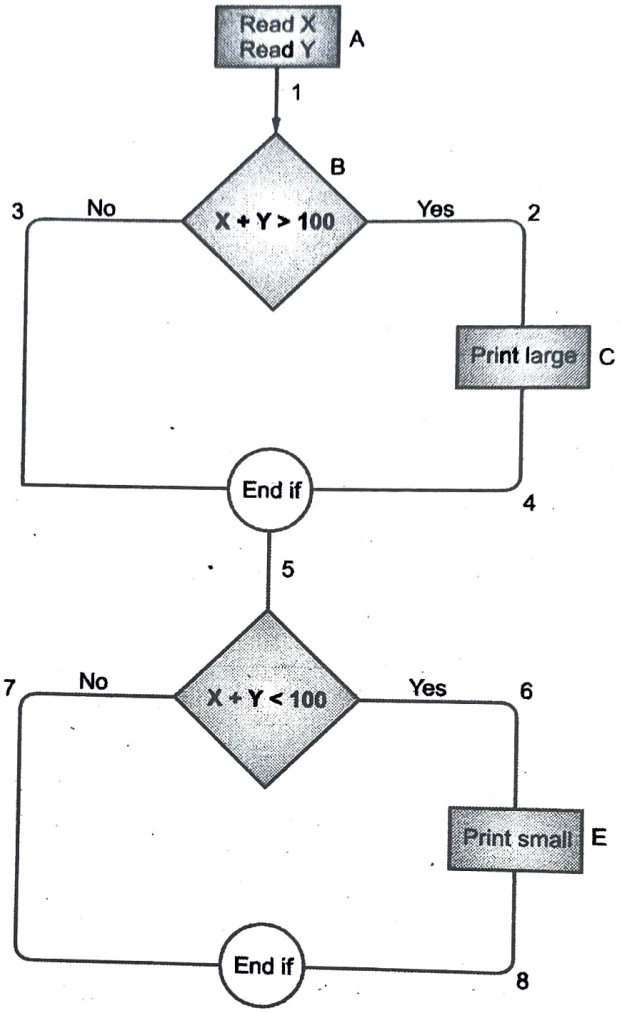


Fig. 3.3.4 : Control flow graph Structure

- In the above diagram, control flow graph of code is depicted. In the first case traversing through "Yes" decision, the path is **A1-B2-C4-D6-E8**, and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path.
- To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

Path 1 - A1-B2-C4-D6-E8

Path 2 - A1-B3-5-D7

Branch Coverage (BC) = Number of paths

= 2

Case	Covered Branches	Path	Branch coverage
Yes	1, 2, 4, 5, 6, 8	A1-B2-C4-D6-E8	
No	3,7	A1-B3-5-D7	

3.3.4 Path Coverage Testing

GQ. What is path coverage testing technique ?

- Path coverage testing is **a specific kind of methodical, sequential testing in which each individual line of code is assessed**. As a type of software testing, path coverage testing is in the category of technical test methods, rather than being part of an overarching strategy or "philosophy" of code.
- It is labor-intensive and is often reserved for specific vital sections of code.
- Techopedia Explains Path Coverage Testing
- The way that path coverage testing works is that the testers must look at each individual line of code that plays a role in a module and, for complete coverage; the testers must look at each possible scenario, so that all lines of code are covered.
- In a very basic example, consider a code function that takes in a variable "x" and returns one of two results : if x is greater than 5, the program will return the result "A" and if x is less than or equal to 5, the program will return the result "B."
- The code for the program would look something like this :

```
input x
if x > 5 then
return A
else return B
```

- In order for path coverage testing to effectively "cover all paths," the two test cases must be run, with x greater than 5 and x less than or equal to 5.
- Obviously, this method becomes much more complicated with more complex modules of code.
- Experts generally consider path coverage testing to be a type of white box testing, which actually inspects the internal code of a program, rather just relying on external inputs and strategies that are considered black box testing, which do not consider internal code.

3.3.5 Conditional Coverage Testing

GQ. What is condition coverage testing ?

- **Condition coverage testing** is a type of white-box testing that tests all the conditional expressions in a program for all possible outcomes of the conditions. It is also called *predicate coverage*.
- **Condition coverage vs. branch coverage** In *branch coverage*, all conditions must be executed at least once. On the other hand, in *condition coverage*, all possible outcomes of all conditions must be tested at least once.
- For example, consider the code snippet below :

```
int a = 10;
if (a > 0){
cout<<"a is positive";
}
```

- *Branch coverage* requires that the condition a > 0 is executed at least once.
- *Condition coverage* requires that both the outcomes a > 0 = True and a > 0 = False of the condition a > 0 are executed at least once.

Examples

Example 1 : Consider the code snippet below, which will be used to conduct *condition coverage testing* :

```
int num1 = 0;
if(num1>0){
cout<<"valid input";
}else{
cout<<"invalid input";
}
```

Condition coverage testing

The condition coverage testing of the code above will be as follows :

Test case number	num1>0	Final output
1	True	True
2	False	False

Example 2 : Consider the code snippet below, which will be used to conduct *condition coverage testing*:

```
int num1 = 0;
int num2 = 0;
if((num1>0 || num2<10)){
cout<<"valid input";
}else{
cout<<"invalid input";
}
```

Condition coverage testing

The condition coverage testing of the code above will be as follows :

Test case number	num1>0	num2<10	Final output
1	True	Not required	True
2	False	True	True
3	False	False	False

Example 3 : Consider the code snippet below, which will be used to conduct *condition coverage testing* :

```
int num1 = 0;
int num2 = 0;
if((num1>0) && (num1+num2<15)){
cout<<"valid input";
}else{
cout<<"invalid input";
}
```

☞ **Condition coverage testing**

The condition coverage testing of the code above will be as follows :

test case number	Num 1 > 0	num1 + num2 < 15	Final output
1	True	True	True
2	True	False	False
3	False	Not required	False

Example 4 : Consider the code snippet below, which will be used to conduct *condition coverage testing* :

```
int num1 = 0;
int num2 = 0;
if((num1>0 || num2<10) && (num1+num2<15)){
cout<<"valid input";
}else{
cout<<"invalid input";
}
```

☞ **Condition coverage testing**

The condition coverage testing of the code above will be as follows :

Test case number	Num 1 > 0	Num 2 < 10	num1 + num 2 < 15	Final output
1	True	don't care	True	True
2	True	don't care	False	False
3	False	True	True	True
4	False	True	False	False
5	False	False	Not required	False

3.3.6 Loop Coverage Testing

- Loop Testing is a kind of software testing that focuses exclusively on the correctness of loop constructions. It is a component of Control Structure Testing (path testing, data validation testing, condition testing).
- Loop testing is an example of white-box testing. This approach is used to test software loops.

☞ **Types of Loop Tests**

The following are some examples of loop tests –

(1) Simple loop (2) Nested loop (3) Concatenated loop (4) Unstructured loop

GQ. Why Loop Testing ?

Following are some of the reasons why loop testing is performed –

- Testing can help to resolve loop recurrence concerns.
- Loop testing can indicate constraints in efficiency and operations.
- The loop's uninitialized variables can be identified by testing loops.
- It aids in the identification of loop initiation issues.

GQ. How to do Loop Testing - Complete Methodology ?

It must be tested at three separate stages within the testing loop –

- When the loop is activated.
- When the loop is executed.
- When the loop is terminated.

The following is the testing technique for all of these loops –

☞ Simple Loop

The following is how a simple loop is tested –

- Ignore the entire loop.
- Make a single pass across the loop.
- Make a number of passes through the loop where a<b, n is the maximum limit of passes.
- Make b, b – 1; b + 1 passes through the loop, where "b" is the highest amount of passes through the loop allowed.

☞ Nested Loop

You must perform the following steps to create a nested loop –

- Adjust all the other loops to their smallest value and begin with the innermost loop.
- Initiate a simple loop test on the innermost loop and keep the outside loops at their smallest iteration parameter value.
- Conduct the test for the following loop and make your way outwards.
- Keep testing till the outermost loop is reached.

☞ Concatenated Loops

- If two loops in a chained loop are free of one other, they are checked as simple loops; otherwise, they are tested as nested loops.
- However, if the loop counter for one loop is utilized as the starting value for the others, the loops are no longer considered separate.

☞ Unstructured Loops

For unstructured loops, the architecture must be restructured to represent the use of structured programming techniques.

☞ Limitation in Loop testing

- Loop issues are especially common in low-level applications.
- The flaws discovered during loop testing are not significant.
- Numerous defects may be identified by the operating system, resulting in storage boundary breaches, identifiable pointer failures, and so on.

☞ Summary

- Loop testing is a type of White Box testing in software engineering. This approach is used to evaluate loops in the program.
- Loop testing can indicate constraints in functionality and operations.
- Loop issues are especially common in low-level applications.

3.4 BLACK BOX TECHNIQUES

- In order to systematically test a set of functions, it is necessary to design test cases.
- Testers can create test cases from the requirement specification document using the following Black Box Testing techniques:

GQ. Explain Black Box Techniques.

3.4.1 Boundary Value Analysis

- The name itself defines that in this technique, we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries.
- Boundary refers to values near the limit where the behavior of the system changes.
- In boundary value analysis, both valid and invalid inputs are being tested to verify the issues.

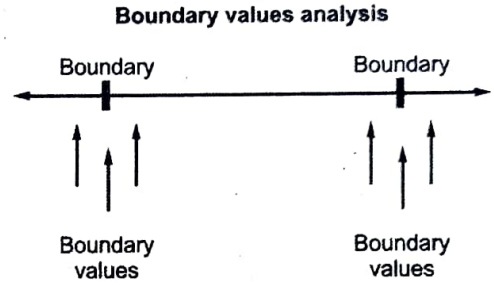


Fig. 3.4.1 : Boundary Value Analysis

☞ **For Example :** (Refer Fig. 3.4.1)

If we want to test a field where values from 1 to 100 should be accepted, then we choose the boundary values : 1 – 1, 1, 1 + 1, 100 – 1, 100, and 100 + 1. Instead of using all the values from 1 to 100, we just use 0, 1, 2, 99, 100, and 101.

3.4.2 Equivalence Class Partition

- This technique is also known as Equivalence Class Partitioning (ECP). In this technique, input values to the system or application are divided into different classes or groups based on its similarity in the outcome.
- Hence, instead of using each and every input value, we can now use any one value from the group/class to test the outcome. This way, we can maintain test coverage while we can reduce the amount of rework and most importantly the time spent.

☞ **For Example :**

As present in the above image, the "AGE" text field accepts only numbers from 18 to 60. There will be three sets of classes or groups.

☞ **Two invalid classes will be :**

(a) Less than or equal to 17.

(b) Greater than or equal to 61.

- A valid class will be anything between 18 and 60.
- We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities. So, testing with any one value from each set of the class is sufficient to test the above scenario.

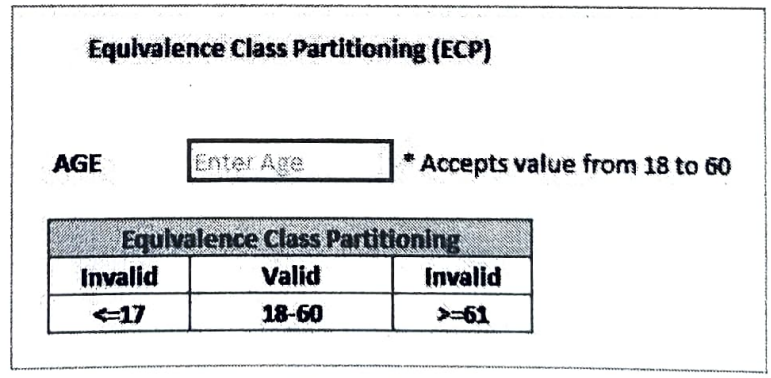


Fig. 3.4.2

3.4.3 State Transition Technique

- State Transition Testing is a technique that is used to test the different states of the system under test. The state of the system changes depending upon the conditions or events. The events trigger states which become scenarios and a tester needs to test them.
- A systematic state transition diagram gives a clear view of the state changes but it is effective for simpler applications. More complex projects may lead to more complex transition diagrams thereby making it less effective.

☞ **For Example :**

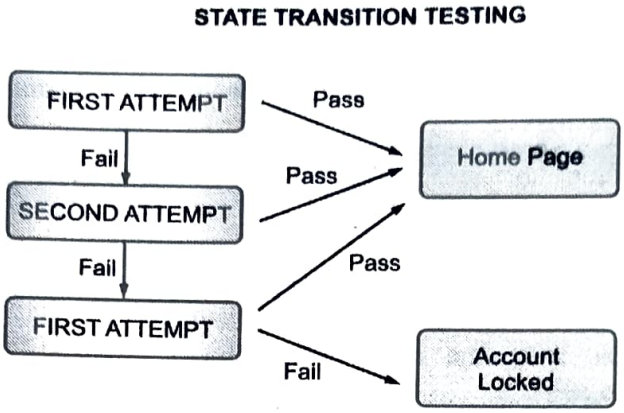


Fig. 3.4.3 : State Transition Testing

3.4.4 Cause Effective Graph

- The Cause and Effect Graph is a dynamic test case writing technique. Here causes are the input conditions and effects are the results of those input conditions.
- Cause-Effect Graph is a technique that starts with a set of requirements and determines the minimum possible test cases for maximum test coverage which reduces test execution time and cost.
- The goal is to reduce the total number of test cases, still achieving the desired application quality by covering the necessary test cases for maximum coverage.
- But at the same time obviously, there are some downsides to using this test case writing technique. It takes time to model all your requirements into this Cause-Effect Graph before writing test cases.
- The Cause-Effect Graph technique restates the requirements specification in terms of the logical relationship between the input and output conditions. Since it is logical, it is obvious to use Boolean operators like AND, OR and NOT.

☞ **Notations Used :**

AND - For effect E1 to be true, both the causes C1 and C2 should be true

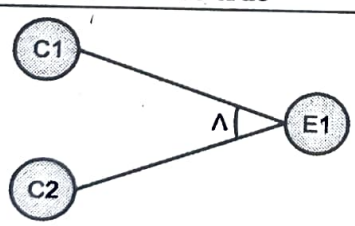


Fig. 3.4.4

OR - For effect E1 to be true, either of causes C1 OR C2 should be true

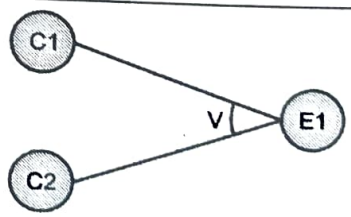


Fig. 3.4.5

NOT – For Effect E1 to be True, Cause C1 should be false

Fig. 3.4.6

MUTUALLY EXCLUSIVE – When only one of the cause will hold true

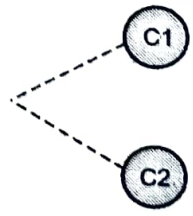


Fig. 3.4.7

☞ **Now let's try to implement this technique with some examples :**

- Draw a Cause and Effect graph based on a requirement/situation.
- Cause and Effect Graph is given, draw a Decision table based on it to draw the test case.

Let's see both of them one by one.

Example : Draw A Cause And Effect Graph According To Situation

Situation

The "Print message" is software that reads two characters and, depending on their values, messages are printed.

- The first character must be an "A" or a "B".
- The second character must be a digit.
- If the first character is an "A" or "B" and the second character is a digit, then the file must be updated.
- If the first character is incorrect (not an "A" or "B"), the message X must be printed.
- If the second character is incorrect (not a digit), the message Y must be printed.

☑ **Solution :**

The Causes of this situation are :

C1 – First character is A
C2 – First character is B
C3 – the Second character is a digit

The Effects (results) for this situation are :

E1 – Update the file
E2 – Print message "X"
E3 – Print message "Y"

☞ **ET'S START !!**

First, draw the Causes and Effects as shown in Fig. 3.4.8.

Key – Always go from Effect to Cause (left to right). That means, to get effect "E", what causes should be true.

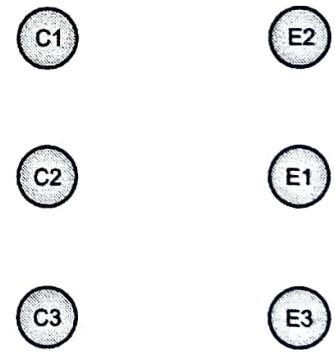


Fig. 3.4.8

☞ **In this example, let's start with Effect E1.**

Effect E1 is for updating the file. The file is updated when

- The first character is "A" and the second character is a digit
- The first character is "B" and the second character is a digit
- The first character can either be "A" or "B" and cannot be both.

☞ **Now let's put these 3 points in symbolic form :**

For E1 to be true – the following are the causes:

- C1 and C3 should be true
- C2 and C3 should be true
- C1 and C2 cannot be true together. This means C1 and C2 are mutually exclusive.

☞ **Now let's draw this :**

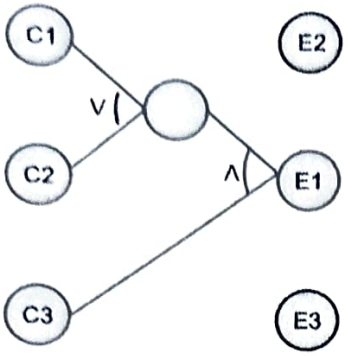


Fig. 3.4.9

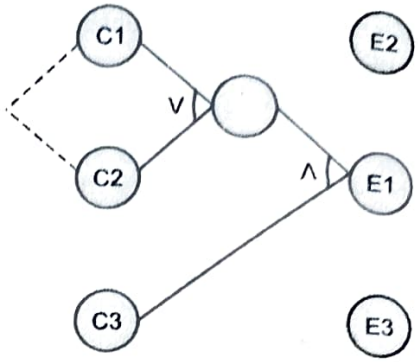


Fig. 3.4.10

So as per shown in Fig. 3.4.9, for E1 to be true the condition is (C1 ∨ C2) ∧ C3

- The circle in the middle is just an interpretation of the middle point to make the graph less messy.
- There is a third condition where C1 and C2 are mutually exclusive. So the final graph for effect E1 to be true is shown in Fig. 3.4.10.

☞ **Let's move to Effect E2**	☞ **For Effect E3.**
• E2 states print message "X". Message X will be printed when the First character is neither A nor B. • This means Effect E2 will hold true when either C1 OR C2 is invalid. So the graph for Effect E2 is shown in Fig. 3.4.11.	• E3 states print message "Y". Message Y will be printed when the Second character is incorrect. • This means Effect E3 will hold true when C3 is invalid. So the graph for Effect E3 is shown in Fig. 3.4.12.

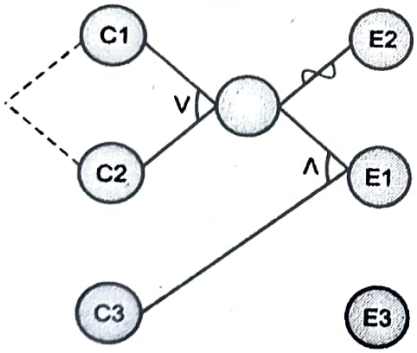
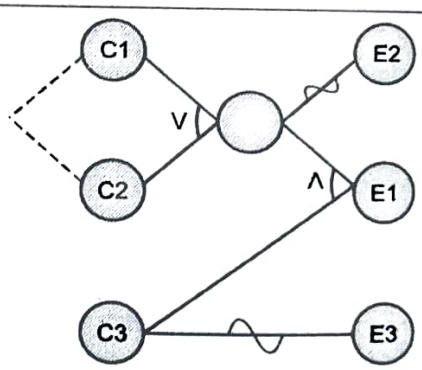

	

Fig. 3.4.11	**Fig. 3.4.12**

- This completes the Cause and Effect graph for the above situation.

3.4.5 Decision Table

GQ. Explain Decision Table with example.

- As the name itself suggests, wherever there are logical relationships like :

```
If
{
   (Condition = True)
   then action1 ;
}
else action2; /*(condition = False)*/
```

- Then a tester will identify two outputs (action1 and action2) for two conditions (True and False). So based on the probable scenarios a Decision table is carved to prepare a set of test cases.

☞ **For Example :**

- Take an example of XYZ bank that provides an interest rate for the Male senior citizen as 10% and 9% for the rest of the people.

Decision Table / Cause-Effect				
Decision Table	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
C1 – Male	**F**	**F**	**T**	**T**
C2 – Senior Citizen	**F**	**T**	**F**	**T**
Actions				
A1 – Interest Rate 10%				**X**
A2 – Interest Rate 9%	**X**	**X**	**X**	

Table 3.4.1 : Decision Table / Cause Effect

- In this example condition, C1 has two values as true and false, C2 also has two values as true and false.
- The total number of possible combinations would then be four. This way we can derive test cases using a decision table.

3.4.6 Use Case Testing

GQ. Explain Use case Testing.

☞ **What Is A Use Case ?**

- Use case testing is a technique that helps to identify test cases that cover the entire system, on a transaction by transaction basis, from start to finish. It is a description of a particular use of the system by a user. It is used widely in developing tests or systems for acceptable levels. In a use case, there will be a set of actions for the user to complete.
- The actions can be :
  - Withdraw funds,
  - Balance transfer, and/or
  - Balance enquiry,
  - Other actions relating to the software that is being developed.

☞ **Use Case Example**

- When developing use cases, a test case table is usually developed. There will be a success scenario as well as the steps that the user should complete.
- Examples of the steps can be :
  - Insert card,
  - Enters a PIN,
  - Allows access to the account.
  - Validates card and asks for a PIN,
  - Validates a Pin, and
- Following this, there will also be a list of extensions within the table. It could happen, for example, that upon validating the card, the system determines that something is incorrect.

- The extensions can be listed as follows :
  2a) Card not valid (Display message and reject card),
  3a) Pin not valid (Display message and ask for re-try – twice), and
  4a) Pin invalid 3 times (eat card and exit).
- Many times, software testers and developers refer to users as 'actors'. Use cases are associated with the following :
  - Actors (human users, external hardware or other components or systems), and
  - Subjects (the component or system to which the use case is applied).
- Each use case specifies some behaviour that a subject can perform in collaboration with one or more actors.
- A use case specifies a type of behaviour that a subject can perform in collaboration with one or more actors, and it can be described by interactions and activities as well as preconditions, post conditions and natural language where appropriate.
- To start with, let's understand **'What is Use Case?'** and later we will discuss **'What is Use Case Testing ?'**.
- A use case is a tool for defining the required user interaction. If you are trying to create a new application or make changes to an existing application, several discussions are made. One of the critical discussions you have to make is how you will represent the requirement for the software solution.
- Business experts and developers must have a mutual understanding about the requirement, as it's very difficult to attain.
- Any standard method for structuring the communication between them will really be a boon. It will, in turn, reduce the miscommunications and here is the place where Use case comes into the picture.

3.5 EXPERIENCED BASED TECHNIQUES

GQ. Explain Experienced Based Techniques.

- Black Box techniques are used to derive test cases from the specification available. White Box techniques supplement the Black Box techniques and use the code to derive test cases and increase the test coverage.
- How do we test when there is insufficient/no documentation and also a limited time frame ? What do we use to derive test cases and test results in such a scenario ?
- Experience-based techniques come in handy in such situations. They use the tester's experience to understand the most important areas of a system i.e. areas most used by the customer and areas that are most likely to fail. They tap into the tester's experience of defects found in the past when testing similar systems.
- Even when specifications are available, it helps adding tests from past experience.

GQ. What are Different types of experience-based testing techniques?

There are mainly two techniques under this category

(1) Error Guessing	(2) Exploratory Testing

3.5.1 Error Guessing

GQ. Explain Error Guessing with example based testing.

- Error Guessing is a simple technique that takes advantage of a tester's skill, intuition and experience with similar applications to identify special tests that formal Black Box techniques could not identify. For example, pressing the Esc key might have crashed a similar application in the past or pressing the

Back button or Enter key on a webpage, JavaScript errors etc.

- This technique completely depends on the tester's experience. The more experienced the tester is, the more errors he can identify.
- Several testers and/or users can also team up to document a list of possible errors, and this can add a lot of value.
- Another way of error guessing is the creation of defect and failure lists. These lists can use available defect and failure data as a starting point and can then expand by using the testers' and users' experience. This list can be used to design tests and this systematic approach is known as fault attack.

This is a classic example of Experience Based Testing

- In this technique, the tester can use his/her experience about the application behaviour and functionalities to guess the error-prone areas.
- Many defects can be found using error guessing where most of the developers usually make mistakes.

Few common mistakes that developers usually forget to handle

- Divide by zero.
- Accepting the Submit button without any value.
- File upload with less than or more than the limit size.
- Handling null values in text fields.
- File upload without attachment.

3.5.2 Exploratory Testing

GQ. Write a note on Exploratory Testing ?

- This is used when specifications are either missing or inadequate and where there is severe time pressure.
- A test charter of test objectives is first created and based on the test objectives, test cases are designed, executed and logged, all these activities occur concurrently along with learning about the application within a fixed time frame (time-box).
- The test objectives help maintain the focus and maximize testing within the limited time frame.

3.6 LEVELS OF TESTING

- Testing levels are the procedure for finding the missing areas and avoiding overlapping and repetition between the development life cycle stages.
- We have already seen the various phases such as **Requirement collection, designing, coding testing, deployment, and maintenance** of SDLC (Software Development Life Cycle).

3.6.1 Functional Testing

GQ. Write a note on Functional Testing ?

- Functional testing is a type of black-box testing. It does not require any knowledge of code. Functional testing is done to test the functionality of the product i.e. the product functions as expected by the customer.
- Functionality is verified by providing the input data and verifying that the output is as expected as in the requirement document.
- Functionality testing covers the functionality of a product, usability i.e. application or product is easy to use, etc.

☞ **Types of Functional Testing**

1. Unit Testing 2. Integration Testing 3. System Testing 4. Acceptance Testing

☞ **Testing is performed in the below order :**

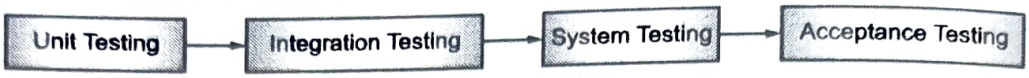


Fig. 3.6.1

3.6.1.1 Unit Testing

GQ. Explain Unit Testing with example.

- Unit testing is the testing of individual components of a product. This testing is done by the developer itself.
- Unit test cases are prepared before the test execution by the developers only. It is very important to execute all the unit test cases as it helps to detect the defect at an early stage.
- Moreover, if the defect is not detected in unit testing, then it could lead to Major and Critical issues at the later stage which indeed would cost high to fix the same and will also take more time than the estimated time.
- While doing unit testing if the developer integrates two components to test to save time and the other component is not yet ready, then the developer uses stubs or drivers to perform his testing. If a defect occurs on the integrated system, then it becomes difficult to find the module, because of which the issue has occurred.
- Thus, testing a component individually is very important.
- Stubs and drivers are basically used when one component calls the function from another component that is not yet developed, thus to create the test environment for testing of the unit, stubs and drivers are used by the developers.

☞ **Stubs**

- A stub is a dummy code that has the same Input parameters as the actual module but has a simplified behaviour.
- Stub follows a **top-down integration approach** i.e. stub is used when the sub-modules are not developed.

☞ **Example :**

If there are three modules i.e. module X, module Y, module Z. Module X is developed and it calls a function from module Y & Z but modules Y and Z are under development. In this case, the stub of module Y & Z is created to test module X. (Refer Fig. 3.6.2)

Fig. 3.6.2

☞ **Drivers**

- The driver follows the **bottom-up integration** approach. It is used when the main module is not ready and the other module needs to be tested.
- In that case, a dummy is created for module X so that Module Y & Z can call the function from Module X.

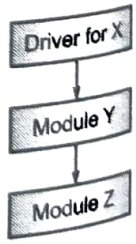


Fig. 3.6.3

3.6.1.2 Integration Testing

GQ. Write a note on Integration Testing with example.

- Integration testing is the testing when two or more modules are integrated or merged to test if the product works fine.
- Modules are integrated in a planned manner as per the integration plan.

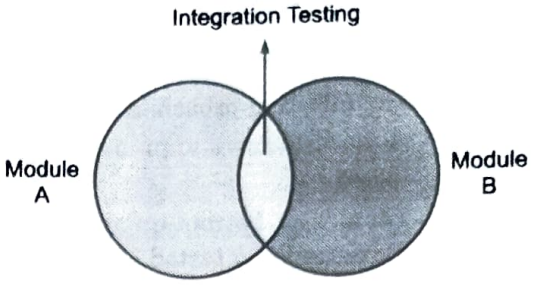


Fig. 3.6.4 : Integration Testing

☞ **Example**

- The computer which we use has individual components as a Monitor, Screen, Key Board, and Mouse.
- Once the monitor and keyboard are ready, they should be integrated to verify if the output is as expected.

☞ **Approaches used in Integration Testing are:**

GQ. What are the approaches used in Integration Testing.

1. Top-Down approach 2. Bottom-up approach 3. Mixed approach 4. Big Bang approach

1. Top-down approach

- In Top-down integration testing, the top-level module is developed and tested first. After that immediate sub-modules are integrated with the top-level module and are tested.
- Stubs are required to complete the integration testing with a top-level module in case the module to be integrated is not developed and tested.

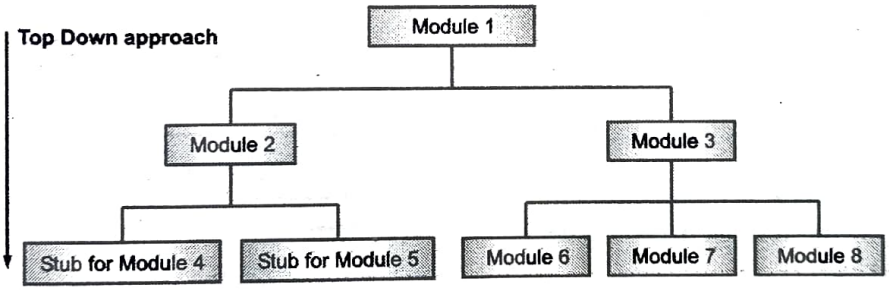


Fig. 3.6.5 : Top down approach

- To test module 2 in integration testing when modules 4 & 5 are not developed, stubs are created to test the same in the top-down approach.

2. Bottom-up Approach

- In the bottom-up approach, sub-modules are developed and tested first and the whole system is tested.

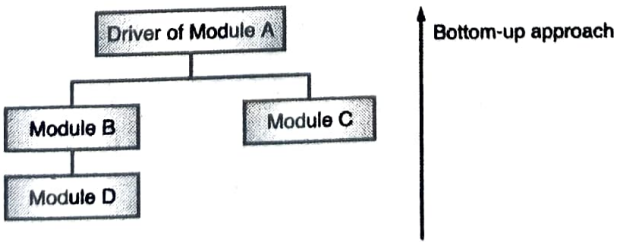


Fig. 3.6.6 : Bottom Up Approach

- To test Module C, the Driver of module A is created so that function can be called.

3. Mixed Approach

- The mixed approach is a **combination of both top-down and bottom-up** approaches.
- In the top-down approach, testing can start only when the top modules are developed and the unit tested.
- Same way, bottom-up approach testing, can start only when the modules at the bottom have been developed and tested.
- The mixed approach overcomes this shortcoming as in the mixed approach, testing can start anytime once the modules are developed.

4. Big Bang Approach

- In the Big bang **approach, all the modules are integrated** into one go and are tested.
- A drawback of this method is that if any error is detected while testing, it would be difficult to find the error-causing module.
- Once the defect cause is detected, it will be fixed but it would cost high & will be time-consuming as the defect found is at a later stage.

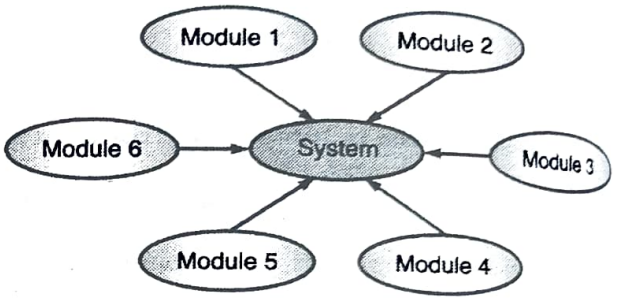


Fig. 3.6.7 : Big Bang Approach

3.6.1.3 System Testing

GQ. Write a note on System Testing.

- In system testing, a system as a whole i.e. a completely developed system is tested to find defects.
- All the modules are integrated to test as a complete system. Scenarios prepared for system testing are run to find the defects.
- All the modules of a product are integrated and tested to verify that the built product is as per the customer requirement. End-to-end testing is performed to cover the complete scenarios.

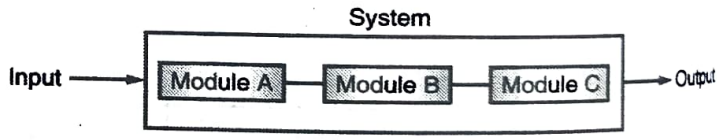


Fig. 3.6.8 : System Testing

3.6.1.4 User Acceptance Testing

- Once the System testing is complete, the Product is ready to be released to the Production environment. But before deployment, the customer's acceptance and approval are required.
- The customer verifies whether the product is all that they wanted or not. For this acceptance testing is done.

Types of Acceptance Testing

GQ. Explain types of Acceptance Testing.

1. **Alpha Testing :** Alpha testing is done internally by the members of the company who developed the product. This testing is not performed by the developers and testers who developed and tested the product.
2. **Beta Testing :** Beta testing is done by the actual users in the Production environment. Beta version of the application is released in the real environment to get feedback from the users which indeed helps to reduce the chances of failure of the product in the production environment.

3. **User Acceptance Testing :** Testing is done at the user end wherein the prepared UAT test cases are executed. Replica of the production environment is created for the customer to perform the user acceptance testing.

3.6.2 Sanity/Smoke Testing

GQ. Explain Sanity / Smoke Testing with example.

Smoke Testing can be understood as a software testing process that determines whether the deployed software build is stable or not.

☞ **Some Pros**

- It helps to find issues in the early phase of testing.
- Improves the overall quality of the system.
- Very limited number of test cases is required to do the Smoke testing.

☞ **Some Cons**

- Smoke testing does is not detailed.
- Smoke testing sometimes causes wastage of time if the software build is not stable.
- This type of test is more suitable if you can automate; otherwise, a lot of time is spent manually executing the test cases.

Sanity Testing is a type of software testing that is performed after receiving a software build. The goal is to determine that the proposed functionality works approximately as expected.

☞ **Some Pros**

- It helps to find related and missing objects.
- It saves lots of time and effort because it is focused on one or few areas of functionality.
- It is the simplest way to assure the quality of the product before it is developed.

☞ **Some Cons**

- Its scope is relatively narrow, compared to the other types of testing.
- Sanity testing focuses only on the commands and functions of the software.
- Most of the time Sanity testing is not at all scripted, which may be a problem for some testers.

GQ. What is Smoke testing and Sanity testing ?

- Smoke testing in the practice of software development and software testing has become a commonly used technique for the early and continuous detection of software defects.
- There are many ways that smoke testing can be applied in software applications and systems.
- Smoke testing, also known as build verification testing or confidence testing, is shorter than the Sanity test and includes only quick scenarios that point at major areas of the product.

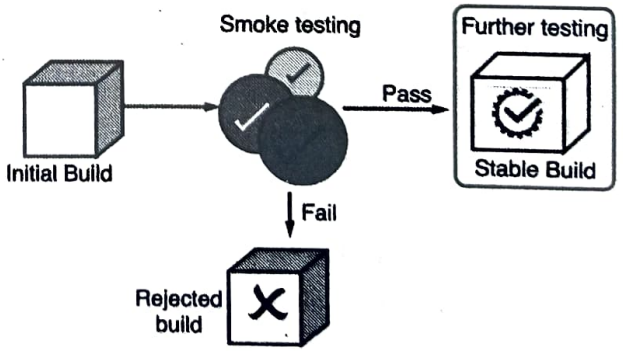


Fig. 3.6.9 : Smoke Testing

- It is performed on initial builds before they are released for extensive testing. Smoke testing is one of the important functional testing types.
- The main idea of this test is to probe these areas in search of smoke that will signal there is a problem hiding below the surface.
- On the other hand, Sanity testing is a type of software test that is performed on a stable build after minor changes to code or functionality. The goal is to determine if the proposed functionality works as expected.

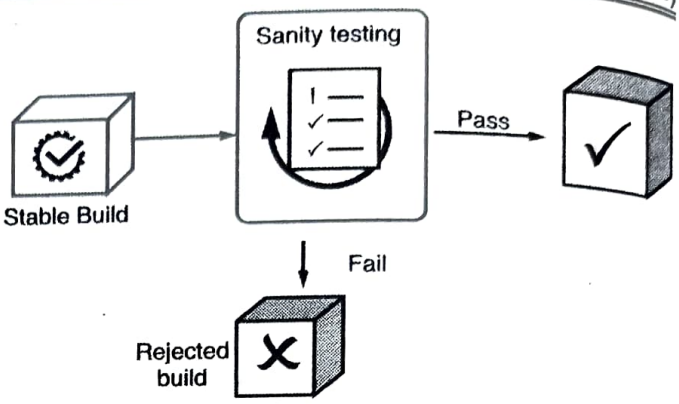


Fig. 3.6.10 : Sanity Testing

- Sanity testing is similar to Regression testing in the fact that you choose specific scenarios that cover all the AUT or application area but shorter and based on the highest risk factors. It is executed at an end in the process of a software development Lifecycle.

☞ Pros of Smoke testing

- Smoke tests are very useful when you either don't have time or don't know where to start and need a quick test to show you the level of your product.
- The results of this test help decide if a build is stable enough to continue with further testing.
- It helps to find issues in the early phase of testing.
- It helps to find issues introduced in integration of modules.
- Improves the effectiveness of the QA team. Many of the problems could be discovered with a Smoke test, thus helping to save time and resources.
- Faster troubleshooting of new and regression bugs. This is due to the high coverage, shallow depth nature of Smoke testing suites.
- Very limited number of test cases is required to do the Smoke testing.
- Smoke testing is quite easy to perform as no special efforts of the testing team are required.
- Improves the overall quality of the system.

☞ Cons of Smoke testing

- Smoke testing does not cover detailed testing.
- Sometimes even after Smoke testing the whole application, critical issues arise in integration and system testing.
- Smoke testing sometimes causes the wastage of time if the software build is not stable.
- This type of test is more suitable if you can automate; otherwise, a lot of time is spent manually executing the test cases.
- This testing is not equal to or a substitute for complete functional testing.

☞ How to automate Smoke testing ?

- Smoke tests can be either manual or automated, but the best way to do a Smoke test is by using an automation tool and programming the smoke suite to run when a new build is created.
- Automation testing is an efficient method that allows to automate most of the testing efforts and there are many tools available to perform this type of test.

- One place where you can find all the tools you need to automate your Smoke test is PractiTest. It is an end-to-end test management tool that gives you control of the entire testing process - from manual testing to automated testing and CI.
- Designed for testers by testers, PractiTest can be customized to your team's ever-changing needs.
- With fast professional and methodological support, you can make the most of your time and release products quickly and successfully to meet your user's needs.
- Some tools that can be used to perform the Smoke Automation test process are:
  - Selenium
  - Robotium
  - Test Complete
  - Appium
  - Cucumber
  - Watir
  - Jenkins (CI tool)
  - Calabash

☞ **Pros of Sanity testing**

- It saves lots of time and effort because it is focused on one or few areas of functionality.
- It is used to verify that a small functionality of the application is still working fine after a minor change.
- Since no documentation is required, these tests can be carried out in a lesser time as compared to other formal tests.
- In case issues are found during Sanity testing, the build is rejected. This saves a lot of time and resources.
- It is the simplest way to assure the quality of the product before it is developed.
- Sanity testing is simple to understand and to carry out and very effective in detecting bugs.
- It can reduce the effort during Smoke and Regression testing.
- It helps to find related and missing objects.

☞ **Cons of Sanity testing**

- Its scope is relatively narrow, compared to the other types of testing.
- Sanity testing focuses only on the commands and functions of the software.
- Most of the time Sanity testing is not at all scripted, which may be a problem for some testers.
- Sanity testing goes nowhere near the design structure level, so it's very difficult for the developers to understand how to fix the issues found during the Sanity testing.

☞ **How to automate Sanity testing ?**

- Sanity tests can be either manual or automated. But beyond that, Sanity tests are more often scripted and automated because they are derived from existing Regression tests.
- Automation saves time and provides faster results for the development team to take action on.
- One place where you can find all the tools you need to automate your Sanity testing is PractiTest.

GQ. What are important differences: Smoke vs Sanity testing ?

Smoke Testing	Sanity Testing
Smoke Testing is performed to ascertain that the critical functionalities of the program are working fine.	Sanity testing is done at random to verify that each functionality is working as expected.
Smoke testing exercises the entire system from end to end.	Sanity testing exercises only the particular component of the entire system.
The main objective of the testing is to verify the stability of the system.	The main objective of the testing is to verify the rationality of the system.
Smoke testing is usually documented and scripted.	Sanity testing is not documented and is unscripted.

Smoke Testing	Sanity Testing
This testing is performed by the developers or testers.	Sanity testing in software testing is usually performed by testers.
It is a well elaborate and planned testing.	This is not a planned test and is done only when there is a shortage of time.
This is a wide and deep testing.	This is a wide and shallow testing.
Smoke testing is a subset of Acceptance testing.	Sanity testing is a subset of Regression Testing.

Similarities between Smoke and Sanity Testing	Explanation
Save time	Smoke and Sanity tests save time by quickly determining whether an application is working properly or not.
Save cost	Due to time and effort savings, costs are reduced.
Integration risk	Integration problems are minimized because end-to-end testing is performed on every build so that functionality-based problems are discovered earlier.
Quality improvement	The main problems are detected and corrected much earlier in the software test cycle, which increases the quality of the software.
Evaluation of progress	Assessing development progress becomes an easier task. Since with each compilation, it is certified that the end-to-end product is working correctly after the addition of new features.

3.6.3 Regression Test

GQ. What is Regression Testing ? Definition, Tools & How to Get Started ?
GQ. What is regression testing ?

- **Regression testing is a software testing practice that ensures an application still functions as expected after any code changes, updates, or improvements.**
- **Regression testing is responsible for the overall stability and functionality of the existing features. Whenever a new modification is added to the code, regression testing is applied to guarantee that after each update, the system stays sustainable under continuous improvements.**
- **Changes in the code may involve dependencies, defects, or malfunctions. Regression testing targets to mitigate these risks, so that the previously developed and tested code remains operational after new changes.**
- **Generally, an application goes through multiple tests before the changes are integrated into the main development branch. Regression testing is the final step, as it verifies the product behaviors as a whole.**

GQ. When to apply regression testing?

Typically, regression testing is applied under these circumstances:

- **A new requirement is added to an existing feature**
- **A new feature or functionality is added**
- **The codebase is fixed to solve defects**

- The source code is optimized to improve performance
- Patch fixes are added
- Changes in configuration

GQ. Why is regression testing important ?

- Test automation is a necessary element in software development practices. Similarly, automated regression testing is also considered a critical puzzle piece.
- With a rapid regression testing process, product teams can receive more informative feedback and respond instantly.
- Regression testing detects new bugs early in the deployment cycle so that businesses do not have to invest in costs and maintenance efforts to resolve the built-up defects.
- Sometimes a seemingly mild modification might cause a domino effect on the product's key functions.
- That's why developers and testers must not leave any alteration, even the smallest, that goes out of their control scope.

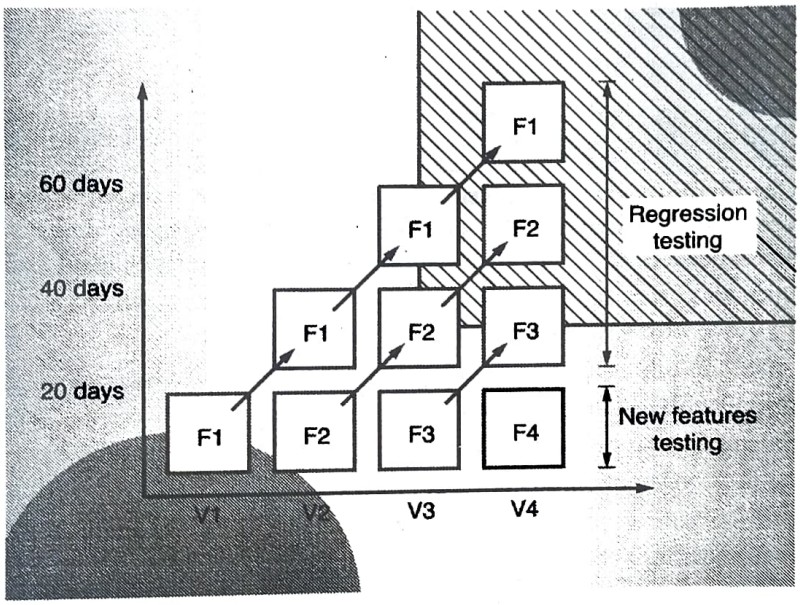


Fig. 3.6.11 : Regression testing

- Functional tests only inspect behaviours of the new features and capabilities, yet dismiss how compatible they are with the existing ones. Therefore, without regression testing, it is more difficult and time-consuming to investigate the root cause and the architecture of the product.
- In other words, if your product undergoes frequent modification, regression testing will be the filter that ensures quality as the product is improved.

GQ. How to perform regression testing ?

Regression testing practices vary among organizations. However, there are a few basic steps :

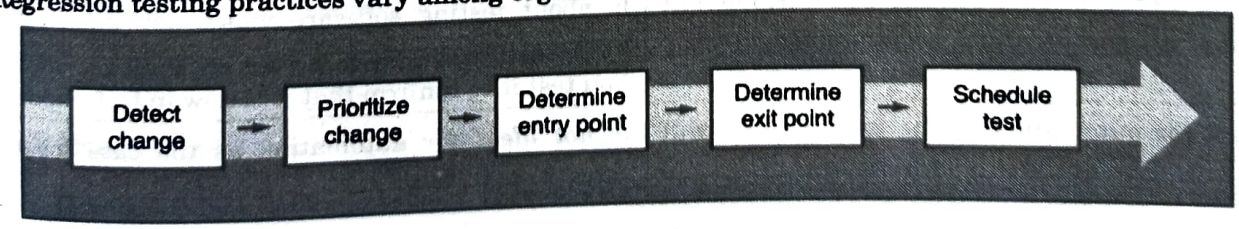


Fig. 3.6.12

Detect Changes in the Source Code

Detect the modification and optimization in the source code; then identify the components or modules that were changed, as well as their impacts on the existing features.

1. **Prioritize Those Changes and Product Requirements**

   Next, prioritize these modifications and product requirements to streamline the testing process with the corresponding test cases and testing tools.

2. **Determine Entry Point and Entry Criteria**

   Ensure whether your application meets the preset eligibility before the regression test execution.

3. **Determine Exit Point**

   Determine an exit or final point for the required eligibility or minimum conditions set in step three.

4. **Schedule Tests**

   Finally, identify all test components and schedule the appropriate time to execute.

   Is time-based execution to test constant changes one of your priorities in testing? If so, learn how to easily manage and set up time-based tests orchestration with TestOps.

3.6.4 Retest

GQ. What is retesting?

- Where retesting differs from regression testing is that, instead of being designed to search through all the previous updates and features of the software to find *unforeseen* defects and bugs, retesting is designed to test specific defects that you've *already detected* (typically during your regression testing).
- In other words, regression testing is about searching for defects, whereas retesting is about fixing specific defects that you've already found.
- They can therefore occur in one and the same testing process, where :
  1. You update your software with a new feature
  2. You test the existing functionality (**regression testing**)
  3. You detect a bug in your existing functionality
  4. You fix the bug
  5. You **retest** that functionality (and hope that it works!)

GQ. What is Regression testing vs. retesting: key differences?

- You could say that regression testing is a *type of* retesting. Retesting essentially means to test something again. And when you are regression testing, you're testing something that you've tested numerous times before.
- But determining what the two have in common might confuse more than it will help. So for the sake of clarity, here's an overview of the key differences.

Regression Testing	Retesting
Involves testing a *general area* of the software.	Involves testing a *specific feature* of the software.
Is about testing software which *was* working, but now, due to updates, *might not* be working.	Is about testing software which you *know was not* working, but which you believe to have been fixed. You test it to confirm that it is now in fact fixed.
Is ideal for automation as the testing suite will grow with time as the software evolves.	*Is not ideal* for automation as the case for testing changes each time.

3.7 NON-FUNCTIONAL TESTING

- Non-functional testing is done to verify the non-functional aspects of a product. **E.g.** Performance of any product comes under non-functional activity.
- Non-functional testing is as important as functional testing is. If it is not done properly, it can lead to major issues in the Product.

Example :

If Performance testing is not done for new functionality added to the application, when the application goes live it becomes too slow to work with and the customer gets affected with the same.

3.7.1 Performance Testing

GQ. Explain performance testing with example.

- Performance testing is done to verify if the system meets the non-functional requirement identified in the SRS document. It verifies **how the system behaves and performs like response time, throughput,** etc.
- **There are various types of Performance testing done as given in Fig. 3.7.1.**

1. **Load Testing**
   - Load testing is done to check how the system responds when the load is increased on the system. When the load is at a peak, it verifies whether the application works as expected or not ?
   - Basically, the load is constantly increased on the system by increasing the number of users using the same application at the same time till the time the load reaches its threshold.

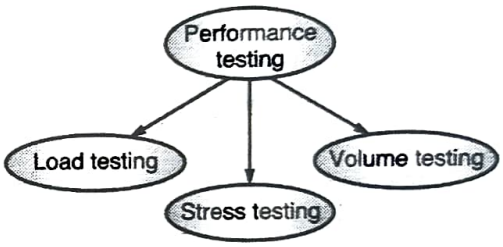


Fig. 3.7.1 : Performance Testing

   - **Example :** If the application's response time is 30 seconds with the number of logged-in users as 1000, then testing is done by increasing the number of users on the site till the number reaches 1000. It verifies that the response time is not increased as the load increases.
2. **Stress Testing**
   - Stress testing is **done beyond the system's capacity** to check at which point it fails. In this, invalid or illegal inputs are used to test so as to stress the capabilities of a product.
   - Stress is increased on the system by increasing the number of users till the time the application breaks.
   - **Example :** If the application's response time is 30 seconds with the number of logged-in users as 1000, then testing is done by increasing the number of users on the site till the number reaches (more than 1000 users) wherein the application breaks down. The number of users is increased beyond the capability of the product to handle it.
3. **Volume Testing**

   Volume testing is done with a huge amount of data **to verify the efficiency & response time of the software** and also to check for any data loss.

3.7.2 Memory Test

If ever there was a piece of embedded software ripe for reuse it is the memory test. This article shows how to test for the most common memory problems with a set of three efficient, portable, public-domain memory test functions.

3.7.3 Scalability Testing

- **Scalability Testing** is a non-functional testing method that measures performance of a system or network when the number of user requests is scaled up or down.
- The purpose of Scalability testing is to ensure that the system can handle projected increase in user traffic, data volume, transaction counts frequency, etc. It tests system ability to meet the growing needs.
- It is also referred to as performance testing, as such; it is focused on the behaviour of the application when deployed to a larger system or tested under excess load.
- In Software Engineering, Scalability Testing is to measure at what point the application stops scaling and identify the reason behind it.

GQ. Why does Scalability Testing ?

- Scalability testing lets you determine how your application scales with increasing workload.
- Determine the user limit for the Web application.
- Determine client-side degradation and end user experience under load.
- Determine server-side robustness and degradation.

GQ. What to test in Scalability Testing ?

Here are few Scalability Testing Attributes :

- Response Time
- Screen transition
- Throughput
- Time (Session time, reboot time, printing time, transaction time, task execution time)
- Performance measurement with a number of users
- Request per seconds, Transaction per seconds, Hits per second
- Performance measurement with a number of users
- Network Usage
- CPU / Memory Usage
- Web Server (request and response per seconds)
- Performance measurement under load

☞ Test Strategy for Scalability testing

Test Strategy for Scalability Testing differs in terms of the type of application is being tested. If an application accesses a database, testing parameters will be testing the size of the database in relation to the number of users and so on.

☞ Prerequisites for Scalability Testing

1. **Load Distribution Capability :** Check whether the load test tool enables the load to be generated from multiple machines and controlled from a central point.
2. **Operating System :** Check what operating systems do the load generation agents and load test master run under
3. **Processor :** Check what type of CPU is required for the virtual user agent and load test master
4. **Memory :** Check how much memory would be enough for the virtual user agent and load test master.

GQ. How to do Scalability Testing ?

1. Define a process that is repeatable for executing scalability tests throughout the application life-cycle
2. Determine the criteria for scalability
3. Shortlist the software tools required to run the load test
4. Set the testing environment and configure the hardware required to execute scalability tests
5. Plan the test scenarios as well as Scalability Tests
6. Create and verify visual script
7. Create and verify the load test scenarios
8. Execute the tests
9. Evaluate the results
10. Generate required reports

☞ **Scalability Test Plan**

- Before you actually create the tests, develop a detailed test plan. It is an important step to ensure that the test conforms as per the application requirement.
- Following are the attributes for creating a well-defined Test Plan for Scalability Testing.
  1. **Steps for Scripts :** The test script should have a detailed step that determines the exact actions a user would perform.
  2. **Run-Time Data :** The test plan should determine any run-time data that is required to interact with the application
  3. **Data Driven Tests :** If the scripts need varying data at run-time, you need to have an understanding of all the fields that require this data.

3.7.4 Compatibility Testing

GQ. Write a note on : Compatibility Testing.

- Compatibility testing verifies whether the application/Product is compatible with all the hardware/software platforms or not.
- The application should be **compatible with all browsers, operating systems, mobile devices.**
- **Example :** The application might show all the images perfectly in one browser whereas, in another browser, the images are shown distorted. So, it's important to verify the application on all the browsers and operating systems that the customer has asked for in his requirement.

3.7.5 Security Testing

Security testing is one of the important testing methods as security is a crucial aspect of the Product. It is done to verify if the application is secured or not.

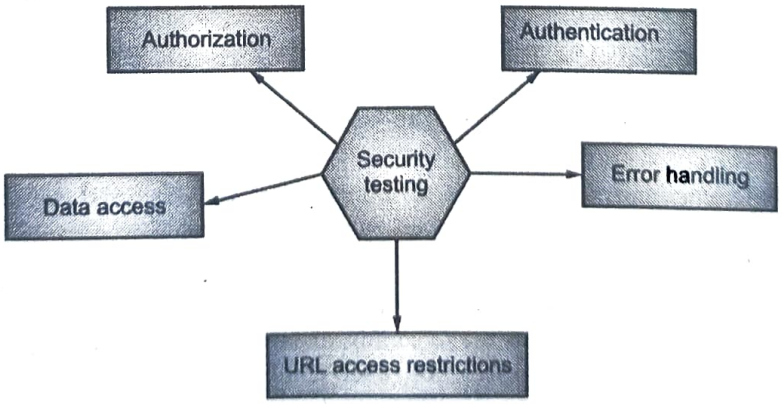


Fig. 3.7.2 : Security Testing

Few examples of what is being tested while performing security testing are :

1. Security testing verifies that only **authorized** users should be able to access data.
2. **Authentication** is verified i.e. the user should be able to log in with his credentials. An incorrect username or password should not allow the user to log in.
3. **Secured and unsecured pages** in the application should be verified.
4. For financial sites, **the session should timeout** after a few minutes if no activity is being performed.

3.7.6 Cookies Testing

GQ. Write a note on Cookies Testing with example.

The Basics of Cookies

- We'll start by discussing what cookies are and how they operate.
- When you understand how cookies operate, it will be much easier for you to comprehend the test cases for testing cookies. How can cookies end up on your computer's hard drive? Also, how can we change our Cookie preferences?

GQ. What Exactly Is a Cookie ?

- A cookie is a little piece of data that a web server stores in a text file on the user are hard disc.
- The web browser then uses this information to obtain information from that machine.

GQ. What is the purpose of cookies ?

- Cookies are simply the user's identification, and they are used to monitor where the user traveled across the website pages. Stateless communication exists between a web browser and a web server.
- As an example
- If you visit the domain http://www.example.com/1.html your web browser will simply ask the example.com web server for page 1.html. If you write the page as http://www.example.com/2.html the next time, a fresh request will be made to the example.com web server for sending the 2.html page, and the webserver has no idea who the prior page 1.html was provided to.
- What if you want to see the history of this user's interactions with the webserver? Somewhere, the user state and interaction between a web browser and a web server must be maintained. This is when the cookie comes into play. Cookies are used to keep user interactions with websites going.

GQ. What exactly is Cookie Testing ?

- Cookie Testing is a form of software testing that examines cookies produced in your web browser.
- A cookie is a little piece of information that the web server stores in a text file on the user's (client's) hard disc.
- Each time the browser requests a page from the server, this information is transmitted back to the server.
- Cookies often include customized user data or information that is used to communicate between websites. The screenshot below displays cookies on various websites.
- To put it another way, cookies are nothing more than a user's identification and are used to monitor where the user travelled across the website's pages.
- A cookie's aim is to allow users and websites to communicate quickly. Cookies can be used to provide a shopping cart, a personalized online experience, user tracking, marketing, user sessions, and other applications.

GQ. What Is the Function of Cookies ?

- The HTTP protocol, which is used to communicate information files on the internet, is used to keep cookies.
- HTTP protocols are classified into two kinds. There are two HTTP protocols: stateless HTTP and stateful HTTP. The stateless HTTP protocol does not maintain track of previously visited web pages. While the Stateful HTTP protocol does preserve some history of prior web browser and web server interactions, cookies employ this protocol to maintain user interactions.
- When a user visits a site or page that uses a cookie, the little code contained inside that HTML page (usually a call to some language script to write the cookie, such as cookies in JAVAScript, PHP, or Perl) creates a text file.
- When a user returns to the same page or site, this cookie is retrieved from the disc and used to identify the user's second visit to that domain. The expiration time is set when the cookie is created. This time is set by the program that will use the cookie.
- In general, two sorts of Cookies are stored on the user's computer.
  1. **Session Cookies :** This cookie remains active as long as the browser that set it is open. This session cookie is erased when we exit the browser. The cookie can sometimes be configured to expire after a 20-minute session.
  2. **Persistent Cookies :** These are cookies that are stored on the user's computer indefinitely and can last months or years.

GQ. What is the Cookie's Content ?

The cookie is made up of three major components.

- The name of the server from which the cookie was sent
- Lifetime of Cookies
- A monetary value. This is generally a one-of-a-kind number created at random.

GQ. Where do cookies get saved ?

- Any web page program that writes a cookie saves it in a text file on the user's hard drive. The location of the cookies is determined by the browser. Cookies are stored in various routes by different browsers.
- Cookies, for example, maybe seen in the browser settings in Mozilla Firefox. To see it, go to Tools -> Options -> Privacy and then "Remove Individual Cookies."
- Cookies are stored on the location "C:\Documents and Settings\Default User\Cookies" in the Internet Explorer browser.

☞ **How to Put Cookies to the Test**

The following is an important checklist and step-by-step guide on how to test cookies on a website –

- Cookies must be disabled – Disable all cookies and try to utilize the site's main features.
- Cookies tainted – manually modify the cookie in a notepad and replace the parameters with some arbitrary ones.
  1. **Encrypting cookies :** Passwords and usernames, for example, should be encrypted before being transferred to our computer.
  2. **Cookie testing with a variety of browsers :** Check that your internet page is correctly writing cookies on a separate browser as intended.

☞ **Examining the removal from your web application page**

- **Cookie rejection on a case-by-case basis :** Delete all of the websites' cookies and see how the website behaves.
- **Cookies are accessible :** Cookies created by one website should not be accessible to other websites.

- **Cookies should not be used excessively :** If the program under test is a public website, cookies should not be used excessively.
- **Experimenting with various settings :** Testing should be done thoroughly to ensure that the website works correctly with various cookie settings.
- **Separately categorize cookies :** Cookies should not be classified in the same category as viruses, spam, or malware.

☞ **Specific Test for multi-environment sites**

- A test that is specific Check if the same Cookies are allowed in all environments for multi-environment sites.
- The usage of wildcards in the Cookie path might be the reason (so-called super cookies). If this is a need, certain access difficulties may arise as a result of the alternative encryption key being utilized (for.Net it is a machine key that usually is unique unless specified otherwise).
- These are some of the most important test scenarios to keep in mind while evaluating website cookies.
- You may create many test cases from these test cases by combining them in different ways. If you have an alternative application situation, please share it in the comments section below.

☞ **Cookies can be used in the following ways**

1. **To implement the shopping cart :** Cookies are used to keep an online ordering system running smoothly. Cookies keep track of what the user wishes to purchase. What if a consumer adds certain goods to their shopping cart and then decides not to buy them this time because of whatever reason and shuts the browser window? When the same user returns to the buy page, he will be able to see all of the goods he put in his shopping basket on his previous visit.
2. **Personalized sites :** When a person views a certain website, they are asked which other pages they do not wish to see. User preferences are saved in a cookie, and those pages are not displayed to the user until he is online.
3. **User tracking :** Counting the number of unique visitors who are online at any one moment.
4. **Advertising :** Some businesses use cookies to display advertising on user computers. These adverts are controlled by cookies. When and how should advertisements be shown? What is the user's point of view? What keywords do they look upon the site? Cookies may be used to keep track of all of these things.
5. **User sessions :** Using a user ID and password, cookies may monitor user sessions to a certain domain.

☞ **Drawbacks of Cookies**

- While writing a Cookie is a great way to keep users engaged, if the user has set their browser to warn them before writing any Cookies or has completely disabled cookies, the site containing the Cookie will be completely disabled and unable to perform any operations, resulting in a loss of site traffic. This may be turned off or on in your browser's settings. For Google Chrome, for example, go to Settings -> Advanced -> Content Settings -> Cookies. You may apply a cookie policy to all websites or set it up for specific ones.
- In addition to browser settings, changes in EU and US regulations require developers to notify users that cookies are being used on their websites.
- Compliance with such new rules should be included in test scenarios for specific areas.
  1. **Too many Cookies :** If you are writing too many cookies on every page navigation and the user has enabled the option to warn before writing the Cookie, this may turn away users from your site.
  2. **Security Concerns :** Personal information about users is sometimes saved in Cookies, and if the Cookie is hacked, the hacker can access your personal information. Even damaged cookies can be read by several websites, posing security risks.
  3. **Sensitive Information :** Some websites may write and keep sensitive information about you in cookies, which is not permitted owing to privacy concerns. This should be sufficient to understand what Cookies are.

3.7.7 Session Testing

GQ. Write a note on session testing.

- **Session-based testing** is a software test method that aims to combine accountability and exploratory testing to provide rapid defect discovery, creative on-the-fly test design, management control and metrics reporting.
- The method can also be used in conjunction with scenario testing. Session-based testing was developed in 2000 by Jonathan and James Bach.
- Session-based testing can be used to introduce measurement and control to an immature test process and can form a foundation for significant improvements in productivity and error detection.
- Session-based testing can offer benefits when formal requirements are not present, incomplete, or changing rapidly.

☞ Elements of session-based testing

Mission

- The mission in Session Based Test Management identifies the purpose of the session, helping to focus the session while still allowing for exploration of the system under test.
- According to Jon Bach, one of the co-founders of the methodology, the mission tells us "what we are testing or what problems we are looking for.

Charter

- A charter is a goal or agenda for a test session. Charters are created by the test team prior to the start of testing, but they may be added or changed at any time.
- Often charters are created from a specification, test plan, or by examining results from previous sessions.

Session

- An uninterrupted period of time spent testing, ideally lasting one to two hours.
- Each session is focused on a charter, but testers can also explore new opportunities or issues during this time.
- The tester creates and executes tests based on ideas, heuristics or whatever frameworks to guide them and records their progress. This might be through the use of written notes, video capture tools or by whatever method as deemed appropriate by the tester

☞ Session report

The session report records the test session. Usually this includes:

- Charter.
- Area tested.
- Detailed notes on how testing was conducted.
- A list of any bugs found.
- A list of issues (open questions, product or project concerns)
- Any files the tester used or created to support their testing
- Percentage of the session spent on the charter vs investigating new opportunities.
- Percentage of the session spent on:
  - Testing - creating and executing tests.
  - Bug investigation / reporting.
  - Session setup or other non-testing activities.
- Session Start time and duration.

3.7.8 Recovery Testing

GQ. Write a note on Recovery Testing.

- Recovery testing is done to verify that the data is recovered if any fault occurs; application crashes or power goes off.
- To test the recovery of data, the software is forcefully failed to check if data is recovered successfully. **E.g.** Printer can be disconnected to check if the system hangs or can be shut down to check data loss.

3.7.9 Installation Testing

- Installation testing is one of the most important testings as Installation is the very first interaction of the user with the product and it's important that the user does not face difficulty in installing the same.
- The software can be installed via CD, Internet, and network location.

Example : If an installation is done through the Internet, then test cases should be included for:

- Bad network speed
- Broken connection.
- Size and approximate time taken.
- Concurrent installation/downloads.
- The Installation of Software requires the use of License keys. Thus different types of licensing are:
  - Node-Locked
  - Floating
  - Named User
  - Temp/Evaluation Licenses

3.7.10 Ad hoc Testing

GQ. Explain Ad hoc Testing.

- **Ad hoc Testing** is an informal or unstructured software testing type that aims to break the testing process in order to find possible defects or errors at an early possible stage.
- Ad hoc testing is done randomly and it is usually an unplanned activity which does not follow any documentation and test design techniques to create test cases.
- Ad hoc Testing does not follow any structured way of testing and it is randomly done on any part of application.
- Main aim of this testing is to find defects by random checking.

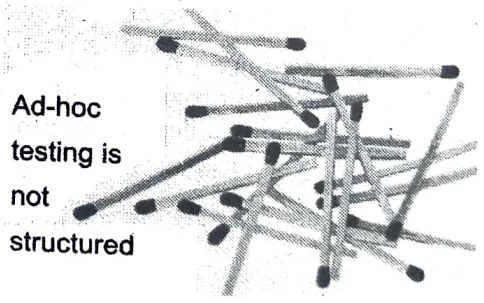


Fig. 3.7.3

- Adhoc testing can be achieved with the Software testing technique called **Error Guessing.** Error guessing can be done by the people having enough experience on the system to "guess" the most likely source of errors.
- This testing requires no documentation/ planning /process to be followed.
- Since this testing aims at finding defects through random approach, without any documentation, defects will not be mapped to test cases. This means that, sometimes, it is very difficult to reproduce the defects as there are no test steps or requirements mapped to it.

GQ. When execute Adhoc Testing ?

- Ad hoc testing can be performed when there is limited time to do elaborative testing.
- Usually adhoc testing is performed after the formal test execution. And if time permits, ad hoc testing can be done on the system.
- Ad hoc testing will be effective only if the tester is knowledgeable of the System Under Test.

☞ **Types of Adhoc testing**

There are different types of Adhoc testing and they are listed as below :

Buddy Testing	Two buddies mutually work on identifying defects in the same module. Mostly one buddy will be from development team and another person will be from testing team. Buddy testing helps the testers develop better test cases and development team can also make design changes early. This testing usually happens after Unit Testing completion.
Pair testing	Two testers are assigned modules, share ideas and work on the same machines to find defects. One person can execute the tests and another person can take notes on the findings. Roles of the persons can be a tester and scriber during testing. *Comparison Buddy and Pair Testing:* Buddy testing is combination of unit and System Testing together with developers and testers but Pair testing is done only with the testers with different knowledge levels. (Experienced and non-experienced to share their ideas and views)
Monkey Testing	Randomly test the product or application without test cases with a goal to break the system.

☞ **Best practices of Adhoc testing**

Following best practices can ensure effective Adhoc Testing

1. **Good business knowledge**
   - Testers should have good knowledge of the business and clear understanding of the requirements- Detailed knowledge of the end to end business process will help find defects easily.
   - Experienced testers find more defects as they are better at error guessing.
2. **Test Key Modules**
   - Key business modules should be identified and targeted for ad-hoc testing.
   - Business critical modules should be tested first to gain confidence on the quality of the system.
3. **Record Defects**
   - All defects need to be recorded or written in a notepad.
   - Defects must be assigned to developers for fixing. For each valid defect, corresponding test cases must be written and must be added to planned test cases.
   - These Defect findings should be made as lesson learned and these should be reflected in our next system while we are planning for test cases.

3.7.11 Risk Based Testing

GQ. Explain Risk Based Testing with example.

- **Risk based testing is basically a testing done for the project based on risks.** Risk based testing uses risk to prioritize and emphasize the appropriate tests during test execution.
- In simple terms – Risk is the probability of occurrence of an undesirable outcome. This outcome is also associated with an impact. Since there might not be sufficient time to test all functionality, Risk based testing involves testing the functionality which has the highest impact and probability of failure.

- Risk-based testing is the idea that we can organize our testing efforts in a way that reduces the residual level of product risk when the system is deployed.
  - Risk-based testing starts early in the project, identifying risks to system quality and using that knowledge of risk to guide testing planning, specification, preparation and execution.
  - Risk-based testing involves both mitigation – testing to provide opportunities to reduce the likelihood of defects, especially high-impact defects – and contingency – testing to identify work-around to make the defects that do get past us less painful.
  - Risk-based testing also involves measuring how well we are doing at finding and removing defects in critical areas.
  - Risk-based testing can also involve using risk analysis to identify proactive opportunities to remove or prevent defects through non-testing activities and to help us select which test activities to perform.
- The goal of risk-based testing cannot practically be – a risk-free project. What we can get from risk-based testing is to carry out the testing with best practices in risk management to achieve a project outcome that balances risks with quality, features, budget and schedule.

GQ. How to perform risk based testing ?

1. Make a prioritized list of risks.
2. Perform testing that explores each risk.
3. As risks evaporate and new ones emerge, adjust your test effort to stay focused on the current crop.

3.7.12 I18N Testing

GQ. Explain I18N Testing.

- Internationalization testing is a non-functional testing technique. It is a process of designing a software application that can be adapted to various languages and regions without any changes.
- Localization, internationalization and globalization are highly interrelated.

1. Localization (l10n)

- Localization is the process of adapting a product or content to a specific locale or market.
- Localization is sometimes written as **l10n**, where 10 is the number of letters between l and n.
- The aim of localization is to give a product the look and feel of having been created specifically for a target market, no matter their language, culture, or location.

2. Internationalization (i18n)

- Internationalization is a design process that ensures a product (usually a software application) can be adapted to various languages and regions without requiring engineering changes to the source code.
- Think of internationalization as readiness for localization.
- Internationalization is often written **i18n**, where 18 is the number of letters between i and n in the English word.

3. Globalization (g11n)

- Globalization, in the context of the language industry, refers to a broad range of processes necessary to prepare and launch products and activities internationally.
- Sometimes written **g11n**, globalization is an all-encompassing concept which applies to activities such as multilingual communication, global-readiness of products and services, and processes and policies related to international trade, commerce, education, and more.

3.7.13 L1ON Testing

GQ. Explain L1ON Testing.

- The world is shrinking with technology, and more and more enterprises are rolling out products across geographies to address global customer base. However, enterprises need the confidence that their product will meet the language and functionality requirements of the local users in the specific regions.
- At the same time, they also need the right approach to scale operations optimally across geographies and deliver product in the shortest possible time-to-market.
- The acceptance of an application rests primarily on how its users perceive it. Critical to that success is the applications' ability to integrate seamlessly with the native language and the cultural landscape of the target market.
- MWIDM brings you Globalization Testing Services to help rapid product rollout across geographies.
- We have built an exhaustive QA methodology to identify specifically potential issues and problems in the globalization process.
- We have specific local checklists for various countries to ensure exhaustive coverage during localization testing.
- We leverage our proven experience in successful globalization testing engagements, skilled multi-lingual QA talent pool, and globalization testing best practices to ensure quick testing for scalability of products.
- The globalization testing includes localization (L10N) and internationalization (I18N) testing. The various aspects of L10N and I18N testing are as shown in Fig. 3.7.4.

Fig. 3.7.4

☞ **Our Globalization Testing Differentiators**

- Rich experience in globalization testing supporting multi-country rollouts, giving you predictable results
- Localization-specific test optimization checklist and best practices to ensure maximum test coverage
- Specialized multi-lingual QA team with specific local knowledge for various countries
- Ability to reduce testing cycle time and enable quicker time-to-market for your products

3.7.14 Compliance Testing

GQ. Explain Compliance Testing.

☞ **Definition – What is Compliance Testing ?**

- ***"Compliance testing"*** also known as Conformance testing is a non-functional testing technique which is done to validate, whether the system developed meets the organization's prescribed standards or not.
- There is a separate category of testing known as "Non-Functional Testing".
- This is basically a kind of an audit which is done on the system to check if all the specified standards are met or not. To ensure that the compliances are met, sometimes a board of regulators and compliance expert people are established in every organization. This board puts a check whether the development teams are meeting the standards of the organization or not.
- The teams do an analysis to check that the standards are properly enforced and implemented. The regulatory board also works simultaneously to improve the standards, which will, in turn, lead to better quality.
- Compliance testing is also known as Conformance testing. The standards normally used by the IT industry, are basically defined by the large organizations like IEEE (International institute of electrical and electronics engineers) or W3C (World Wide Web Consortium), etc.
- It can also be carried out by an independent/third party company which specializes in this type of testing and service.

☞ **Objectives**

Objectives of compliance testing include

- Determining that the development and maintenance process meets the prescribed methodology.
- Ensures whether the deliverables of each phase of the development meets the standards, procedures, and guidelines.
- Evaluate the documentation of the project to check for completeness and reasonableness

GQ. When to use Compliance Testing ?

- It is solely the management's call. If they want, they have to enforce sufficient tests to validate the degree of compliance to the methodology and identify the violators. But it may be possible that lack of compliance is due NOT understanding the methodology or they are misunderstood.
- Management should ensure that the teams have a proper and clear understanding of the standards, procedures, and methodology. They can arrange proper training for the team if needed.
- It may be possible that the standards are not published properly or maybe that the standards itself are of poor quality. In such a situation, efforts should be made either to rectify it or to adopt a new methodology.
- It is important that the compliance check should be made right from the inception of the project than at the later stage because it would be difficult to correct the application when the requirement itself is not adequately documented.

GQ. How to do a compliance check ?

- Doing Compliance check is quite straight forward. A set of standards and procedures are developed and documented for each phase of the development lifecycle.
- Deliverables of each phase need to compare against the standards and find out the gaps. This can be done by the team through the inspection process, but I would recommend an independent team to do it.
- After the end of the inspection process, the author of each phase should be given a list of non-compliant areas that needs to correct.
- The inspection process should again be done after the action items are worked upon, to make sure that the non-conformance items are validated and closed.

☞ **#Exemplar/Case Studies**

1. Case Study : Manual Testing (Online Marketing Software Platform)

Background

- The Client – INTECH Consultancy, is one of growing IT Company in India that provides software solution for Logistics and Marine for Hong Kong based clients, it also provides other services to small and medium businesses. They also providing the customer support for their shipment based products.
- The main product for client is based on Port Operation Management which contains shipment and vessel operations.

Business Objective

- The primary objective of KiwiQA team was to provide the solution for manual testing in their product.
- The main purpose in manual testing is to understand the product and gathering the requirements of it, on later stage to prepare the business scenarios or the test cases for the product, logging the issues in Test Management tool, prepare test data in spreadsheet for the test execution.
- Apart from this objective, team provide the other solutions like regression testing, smoke testing for each releases and suggesting the improvements in product.

Challenges	KiwiQA Approach	Value Delivered
• Understanding the port operation management flow.	• Team gathers the requirement from client and take understanding of the port operations.	• Team prepared around 3800 + test cases (75 % functional and 25 % Usability/UI) and logged around 1500 + bugs in the product.
• One of the major challenge is functional complexity as port operation is very unfamiliar domain in IT industries.	• Team studied about the port industries from various online websites and gathered the information about port operations.	• Team has compared and understand the client product with the gathered documents online.
• Another challenge is due to having frequent releases because of large amount if changes in requirements and very tight deadlines.	• Team used traceability matrix for the issue tracking and working extra hours to achieve the testing deadlines.	• Team has achieved almost 90 % to deliver the build with about 95 % accuracy in every release with very tight deadlines.

Solution Background and Engagement Details- The Outcome

- Our manual testing solution was delighted and given the confidence to the Customer where it had functional improvements, efforts and accuracy.
- In earlier stages it was a bit tricky to work on the project and challenging as well to understand the complete flow of port operation management.
- On later stages once the team gathered the requirements and understood the product, the test execution approach gets better and the execution time decreased.
- Our solution has gained client confidence and thus strengthens the relations between us and which in turn made client to give more work over a longer period of time.
- Link:https://www.360logica.com/blog/case-study-manual-testing-online-marketing-software-platform/

2. Case Study : Decision Table Testing (transferring money online to an account which is already added and approved.)

- Decision Table Testing is a software testing methodology used to test system behavior for various input combinations.
- In this systematic approach, the several input combinations and their corresponding system behavior are represented in tabular form.
- The decision table is also called a Cause-Effect table, as the causes and effects for comprehensive test coverage are captured in this table.
- Decision Table testing is a commonly used black-box testing technique and is ideal for testing two or more inputs that have a logical relationship.

GQ. What is a Decision Table ?

- A decision table is the tabular representation of several input values, cases, rules, and test conditions.
- The Decision table is a highly effective tool utilized for both requirements management and complex software testing. Through this table, we can check and verify all possible combinations of testing conditions.
- The testers can quickly identify any skipped needs by reviewing the True(T) and False(F) values assigned for these conditions.

☞ **Advantages Of Decision Table Testing**

1. Decision tables are one of the most effective and full-proof design testing techniques.
2. Testers can use decision table testing to test the results of several input combinations and software states.
3. It gives the developers to state and analyzes complex business rules.
4. Decision table testing is the most preferred black box testing and requirements management.
5. A decision table is used for modeling complex business logic. They can first be converted to test cases and test scenarios through decision table testing.
6. This technique provides comprehensive coverage of all test cases that can significantly reduce the re-work on writing test cases and test scenarios.
7. Decision tables guarantee coverage of all possible combinations of condition values which are called completeness property.
8. Decision tables can be used iteratively. The table results created in the first testing iteration can be used for the next and so on.

9. Decision tables are easy to understand, and everyone can use and implement this design and testing method, scenarios and test cases without prior experience.
10. Multiple conditions, scenarios and results can be viewed and analyzed on the same page by both developers and testers.

GQ. Give short note on : Decision Table Examples.

A decision table is a tabular representation of inputs vs cases, rules and test conditions.

Example 1 : In this example, we see how to create the decision table for a login screen that asks for UserId and Password.

The condition here is that the user will be redirected to the homepage if he enters the correct user name and password, and an error message will be displayed if the input is wrong.

Conditions	Rule 1	Rule 2	Rule 3
Username (T/F)	F	T	F
Password (T/F)	F	F	T
Output (E/H)	E	E	E

Legend

T – Correct username or password
F – Wrong username or password
E – Error message is displayed.
H – Home screen is displayed.

☞ **Decision Table Interpretation**

- **Case 1 :** Username and Password both are wrong, and the user is shown an error message.
- **Case 2 :** Username is correct, but the password is wrong, and the user is shown an error message,
- **Case 3 :** The username is wrong, but the password is correct, and the user is shown an error message.
- **Case 4 :** Username and password both are correct, and the user is taken to the homepage.

☞ **Test Scenarios Possible For This Decision Table**

1. Enter correct username, correct password, and click on login. The expected result is that the user should navigate to the homepage.
2. Enter correct username, wrong password, and click on login. The expected result is that the user should get an error message.
3. Enter the wrong username, correct password, and click on login. The expected result is that the user should get an error message.
4. Enter the wrong username, wrong password, and click on login. The expected result is that the user should get an error message.

Example 2 : In this example, we consider the decision table and test scenarios for an Upload screen.

There is a dialogue box that will ask the user to upload a photo with the following conditions:

The file must be in the .jpg format.

The file size must be less than 32kb.

The image resolution must be 137*177.

If any one of the above conditions fails, the system will display the corresponding error messages about the issue. If all conditions are satisfied, the photo will be uploaded successfully.

Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Format	.JPG	.JPG	.JPG	.JPG	Not.JPG	Not.JPG	Not.JPG	Not.JPG
Size	< 32 kb	< 32 kb	>= 32 kb	>=32 kb	< 32 kb	< 32 kb	>= 32 kb	>= 32 kb
Resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded successfully	Error message due to resolution mismatch	Error message due to size mismatch	Error message due to size and resolution mismatch	Error message due to format mismatch	Error message due to format and resolution mismatch	Error message due to format and size mismatch	Error message due to format, size and resolution mismatch

For these conditions of the decision table, we can formulate eight different test cases or input scenarios to cover all the possibilities.

☞ Scope of Decision Table Testing

- When data is complex, and every combination needs to be tested, decision tables can become huge.
- You can intelligently reduce the number of varieties in each possibility to only choose the interesting and impactful ones. This approach is called Collapsed Decision Table Testing.
- In this technique, redundant conditions that are irrelevant to the outcome are removed, and different outputs are produced. An additional layer of analysis is added to the test design so that the tester can perform more effective testing.
- Decision tables are a robust specification-based testing technique that can work for many scenarios.
- The tabular and the graphical representation is very beneficial for all stakeholders and non-technical members to understand easily.
- The project team members can instantly obtain detailed insights about the problem at hand through illustrative examples and real-life scenarios.
- By moving to the next level of collapsed decision-making table, the management can realize the effectiveness and efficiency of this testing technique.

...Chapter Ends

□□□